Universidade Federal do Rio de Janeiro
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

Programa de Engenharia de Sistemas e Computação

CPS767 - Algoritmos de Monte Carlo e Cadeias de Markov

Prof. Daniel Ratton Figueiredo

2ª Lista de Exercícios

Luiz Henrique Souza Caldas
email: lhscaldas@cos.ufrj.br

9 de abril de 2025

Questão 1: Cauda do dado

Considere um icosaedro (um sólido Platônico de 20 faces) honesto, tal que a probabilidade associada a cada face é 1/20. Considere que o dado será lançado até que um número primo seja observado, e seja Z a variável aleatória que denota o número de vezes que o dado é lançado. Responda às perguntas abaixo:

1. Determine a distribuição de Z, ou seja $P[Z = k], k = 1, 2, \ldots$. Que distribuição é esta?

Resposta:

Os números primos são 2, 3, 5, 7, 11, 13, 17, 19. Portanto, a probabilidade de obter um número primo em um lançamento é $p = \frac{8}{20} = \frac{2}{5}$. Consequentemente, a probabilidade de não obter um número primo em um lançamento é $1 - p = \frac{3}{5}$. Assim, a distribuição de Z é dada por:

$$\boxed{P[Z = k] = (1-p)^{k-1}p = \left(\frac{3}{5}\right)^{k-1}\left(\frac{2}{5}\right)}$$

para $k = 1, 2, \ldots$. Esta é uma distribuição geométrica com parâmetro $p = \frac{2}{5}$. ✓

2. Utilize a desigualdade de Markov para calcular um limitante para $P[Z \geq 10]$.

Resposta:

A desigualdade de Markov afirma que, para uma v.a. $Z > 0$ e $a > 0$, temos:

$$P[Z \geq a] \leq \frac{E[Z]}{a}$$

Para calcular $E[Z]$, utilizamos a fórmula da média de uma distribuição geométrica:

$$E[Z] = \frac{1}{p} = \frac{1}{\frac{2}{5}} = \frac{5}{2}$$

Assim, aplicando a desigualdade de Markov com $a = 10$, obtemos:

$$P[Z \geq 10] \leq \frac{E[Z]}{10} = \frac{\frac{5}{2}}{10} = \underline{\frac{1}{4} = 0.25}$$

Portanto, $P[Z \geq 10] \leq 0.25$. Isso significa que a probabilidade de o número de lançamentos do dado ser maior ou igual a 10 é menor ou igual a 25%. ✓

3. Utilize a desigualdade de Chebyshev para calcular um limitante para $P[Z \geq 10]$.

Resposta:

A média e variância da distribuição geométrica são $\mu = \frac{1}{p} = \frac{5}{2}$ e $\sigma^2 = \frac{1-p}{p^2} = \frac{15}{4}$. Aplicando Chebyshev:

$$P[|Z - \mu| \geq k\sigma] \leq \frac{1}{k^2}$$

Subtraindo μ de ambos os lados em $P[Z \geq 10]$, temos:

$$P[Z \geq 10] = P[Z - \mu \geq 10 - \tfrac{5}{2}] = P[Z - \mu \geq \tfrac{15}{2}] \leq P[|Z - \mu| \geq k\sigma]$$

Fazendo $k\sigma = \frac{15}{2}$, então

$$k = \frac{15}{2} \cdot \frac{2}{\sqrt{15}} = \sqrt{15}$$

Assim, temos:

$$\boxed{P[Z \geq 10] \leq \frac{1}{15} \approx 0.0667}$$

✓

4. Calcule o valor exato de $P[Z \geq 10]$ (dica: use probabilidade complementar). Compare os valores obtidos.

Resposta:

$$P[Z \geq 10] = 1 - P[Z \leq 9] = 1 - \sum_{k=1}^{9} P[Z = k]$$

$$P[Z \geq 10] = 1 - \frac{2}{5}\sum_{k=1}^{9}\left(\frac{3}{5}\right)^{k-1} = 1 - \frac{2}{5} \cdot \frac{1 - \left(\frac{3}{5}\right)^9}{1 - \frac{3}{5}} = \left(\frac{3}{5}\right)^9$$

$$\boxed{P[Z \geq 10] = \left(\frac{3}{5}\right)^9 \approx 0{,}0101}$$

Comparando os valores:

- Markov: $P[Z \geq 10] \leq 0{,}25$
- Chebyshev: $P[Z \geq 10] \leq 0{,}0667$
- Valor exato: $P[Z \geq 10] \approx 0{,}0101$

Ambas as desigualdades fornecem limites conservadores, sendo Chebyshev mais ajustado que Markov. O valor exato é o mais preciso.

✓

Questão 2: Pesquisa eleitoral

Você leu no jornal que uma pesquisa eleitoral com 1500 pessoas indicou que 40% dos entrevistados prefere o candidato A enquanto 60% preferem o candidato B. Determine a margem de erro desta pesquisa usando uma confiança de 95%. O que você precisou assumir para calcular a margem de erro?

Resposta:

Seja X_i uma variável aleatória i.i.d. que representa a preferência do entrevistado i, com:

$$X_i = \begin{cases} 1 & \text{se prefere o candidato A} \\ 0 & \text{se prefere o candidato B} \end{cases}$$

A média amostral de X_i é dada por:

$$M_n = \frac{1}{n}\sum_{i=1}^{n} X_i = 0.4$$

Como $E[M_n] = \mu$, onde $\mu = p$ é a média da distribuição de Bernoulli, temos, pela desigualdade de Chebyshev:

$$P[|M_n - p| \geq k\sigma_{M_n}] \leq \frac{1}{k^2}$$

onde $\sigma_{M_n} = \sqrt{\frac{\sigma^2}{n}}$ é o desvio padrão da média amostral, sendo $\sigma^2 = p(1-p)$ a variância da distribuição de Bernoulli.

Fazendo a margem de erro $\epsilon = k\sigma_{M_n}$, temos:

$$k = \frac{\epsilon}{\sigma_{M_n}} = \frac{\epsilon\sqrt{n}}{\sigma} \Rightarrow P[|M_n - p| \geq \epsilon] \leq \frac{\sigma^2}{\epsilon^2 n}$$

Aplicando o complementar, temos:

$$P[|M_n - p| < \epsilon] = 1 - P[|M_n - p| \geq \epsilon] \geq 1 - \frac{\sigma^2}{\epsilon^2 n} = \beta$$

onde β é o nível de confiança que queremos.

Resolvendo para ϵ:

$$\epsilon = \sqrt{\frac{\sigma^2}{(1-\beta)n}} = \sqrt{\frac{p(1-p)}{(1-\beta)n}}$$

Assumindo que n é grande o suficiente para $M_n = \mu = p = 0.4$, temos:

$$\epsilon = \sqrt{\frac{0.4(1-0.4)}{(1-0.95)1500}} = \sqrt{\frac{0.4(0.6)}{0.05 \cdot 1500}} = \sqrt{\frac{0.24}{75}} = \sqrt{0.0032} \approx \underline{0.0565}$$ ✓

Questão 3: Sanduíches

Você convidou 64 pessoas para uma festa e agora precisa preparar sanduíches para os convidados. Você acredita que cada convidado irá comer 0, 1 ou 2 sanduíches com probabilidades 1/4, 1/2 e 1/4, respectivamente. Assuma que o número de sanduíches que cada convidado irá comer é independente de qualquer outro convidado. Quantos sanduíches você deve preparar para ter uma confiança de 95% de que não vai faltar sanduíches para os convidados?

Resposta:

Seja X_i a variável aleatória que representa o número de sanduíches que o convidado i irá comer, com distribuição de probabilidade dada por $p_1 = \frac{1}{4}$, $p_2 = \frac{1}{2}$ e $p_3 = \frac{1}{4}$, onde p_i é a probabilidade de o convidado comer i sanduíches. O valor esperado e a variância de X_i são dados por:

$$\mu = E[X_i] = \sum_{i=0}^{2} i \cdot p_i = 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} = 0 + \frac{1}{2} + \frac{2}{4} = 1$$

$$E[X_i^2] = \sum_{i=0}^{2} i^2 \cdot p_i = 0^2 \cdot \frac{1}{4} + 1^2 \cdot \frac{1}{2} + 2^2 \cdot \frac{1}{4} = 0 + \frac{1}{2} + \frac{4}{4} = 0 + \frac{1}{2} + 1 = \frac{3}{2}$$

$$\sigma^2 = Var[X_i] = E[X_i^2] - (E[X_i])^2 = \frac{3}{2} - 1^2 = \frac{3}{2} - 1 = \frac{1}{2}$$

Como vimos na questão anterior, aplicando a desigualdade de Chebyshev, temos:

$$P[|M_n - \mu| \geq \epsilon] \leq \frac{\sigma^2}{\epsilon^2 n} = \beta$$

onde $M_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ é a média amostral e ϵ e β são a margem de erro e a confiança desejadas. Resolvendo para ϵ, temos:

$$\epsilon = \sqrt{\frac{\sigma^2}{(1-\beta)n}} = \sqrt{\frac{0.5}{(1-0.95)64}} = \sqrt{\frac{0.5}{0.05 \cdot 64}} = \sqrt{\frac{0.5}{3.2}} = \sqrt{0.15625} \approx 0.395$$

Porém,

$$P[|M_n - \mu| \geq \epsilon] = P[M_n \in [\mu - \epsilon, \mu + \epsilon]] = P[M_n \in [0.605, 1.395]]$$

Assim, temos 95% de confiança de que o máximo da média de sanduíches que os convidados irão comer é 1.395. Multiplicando isso pelo número de convidados, temos:

$$\text{Número de sanduíches} \geq 1.395 \cdot 64 \approx 89.28$$

Portanto, devemos preparar 90 sanduíches para ter 95% de confiança de que não vai faltar sanduíches para os convidados. ✓

Questão 4: Graus improváveis

Considere o modelo de grafo aleatório de Erdős-Rényi (também conhecido por $G(n, p)$), onde cada possível aresta de um grafo rotulado com n vértices ocorre com probabilidade p, independentemente. Responda às perguntas abaixo:

1. Determine a distribuição do grau do vértice 1 (em função de n e p).

Resposta:

O vértice 1 pode ou não ter aresta ligada de um só outro a 1 vértice, com probabilidade p. Modelando isso temos uma variável aleatória $Y \sim Bernoulli(p)$.

O grau do vértice é dado pela soma dessas v.a. de Bernoulli $X = \sum_{i=1}^{n-1} Y_i$, que, por definição, é uma v.a. Binomial com parâmetros $n-1$ e p, ou seja, $X \sim Binomial(n-1, p)$. Assim, temos:

$$P(X = k) = \binom{n-1}{k} p^k (1-p)^{(n-1)-k}$$

onde $k = 0, 1, \ldots, n-1$.

2. Determine o valor γ (em função de n e p) tal que com alta probabilidade $(1 - 1/n)$ o grau observado no vértice 1 é menor ou igual a γ.

Resposta:

$$P[X \leq \gamma] = 1 - P[X > \gamma] \geq 1 - \frac{1}{n}$$

$$P[X > \gamma] \leq \frac{1}{n}$$

Aplicando a desigualdade de Chernoff, temos:

$$P[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2\mu}{3}}$$

Fazendo $\delta = \frac{\lambda}{\mu}$, temos:

$$P[X \geq \mu + \lambda] \leq e^{-\frac{\lambda^2}{3\mu}}$$

Para uma v.a $X \sim Binomial(n-1, p)$, $\mu = E[X] = (n-1)p$. Com isso, temos:

$$P[X \geq \mu + \lambda] \leq e^{-\frac{\lambda^2}{3(n-1)p}} = \frac{1}{n}$$

Resolvendo para λ, temos:

$$\lambda = \sqrt{3(n-1)p\ln(n)}$$

Assim, temos:

$$\gamma = \mu + \lambda = \underline{(n-1)p + \sqrt{3(n-1)p\ln(n)}}$$

Questão 5: Calculando uma importante constante

Seja X_i uma sequência i.i.d. de v.a. contínuas uniformes em $[0, 1]$. Seja V o menor número k tal que a soma das primeiras k variáveis seja maior do que 1. Ou seja, $V = \min\{k \mid X_1 + \cdots + X_k \geq 1\}$.

1. Escreva e implemente um algoritmo para gerar uma amostra de V.

Resposta:

- Pseudo-código:

```
soma = 0;
contador = 0;
enquanto soma < 1 faça
    x = amostra de U[0, 1];
    soma = soma + x;
    contador = contador + 1;
retorne contador;
```

- Implementação em Python:

```python
def amostra_V():
    soma = 0
    contador = 0
    while soma < 1:
        x = random()
        soma += x
        contador += 1
    return contador
```

2. Escreva e implemente um algoritmo de Monte Carlo para estimar o valor esperado de V.

Resposta:

- Definição do estimador: Pela Lei dos Grandes Números, temos que, para n suficientemente grande, a média amostral converge para o valor esperado:

$$E[V] \approx \frac{1}{n} \sum_{i=1}^{n} V_i$$

onde V_i é a amostra de V obtida na iteração i.

- Pseudo-código:

```
n = N; // número de amostras
soma = 0;
para i = 1 até n faça
    soma = soma + amostra_V();
estimador = soma / n;
retorne estimador;
```

- Implementação em Python:

```python
def estimador_E(n):
    soma = 0
    for i in range(n):
        x = amostra_V()
        soma += x
    return soma / n
```

3. Trace um gráfico do valor estimado em função do número de amostras. Para qual valor seu estimador está convergindo?

Resposta:

Código para gerar o gráfico:

```
def grafico_E(n_max):

    n_values = list(range(1, n_max + 1))
    E_values = []

    soma = 0
    for n in n_values:
        x = amostra_V()
        soma += x
        E_values.append(soma / n)

    plt.plot(n_values, E_values, label='E[V] a cada n')
    label = f'E[V] = {E_values[-1]}'
    plt.axhline(y=E_values[-1], color='r', linestyle='--',
       label=label)
    plt.title('Estimador de E[V] em função de n')
    plt.xlabel('n')
    plt.ylabel('E[V]')
    plt.grid(True)
    plt.legend()
    plt.show()
```

Para 10^4 amostras, o estimador converge para aproximadamente $E[V] \approx 2.718$, como visto na figura abaixo.

constante de Euler

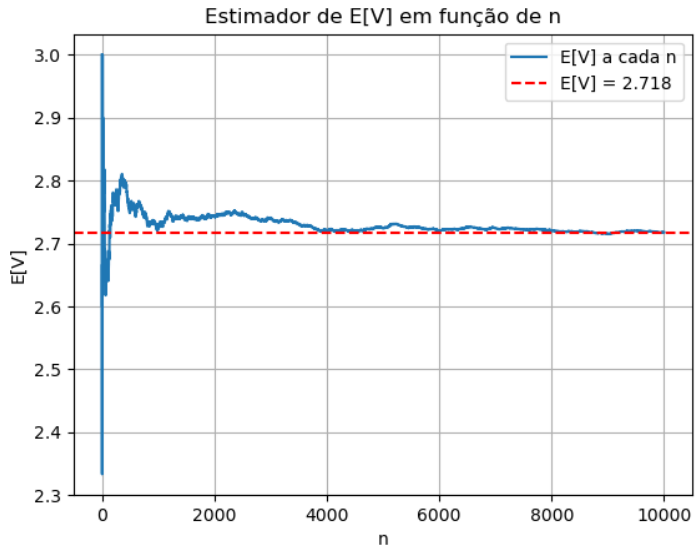

Figura 1: Estimador de $E[V]$ em função de n

Questão 6: Transformada inversa

Mostre como o método da transformada inversa pode ser usado para gerar amostras de uma v.a. contínua X com as seguintes distribuições:

1. Distribuição exponencial com parâmetro $\lambda > 0$, cuja função densidade é dada por $f_X(x) = \lambda e^{-\lambda x}$, para $x \geq 0$.

Resposta:

A função de distribuição acumulada (CDF) da distribuição exponencial é dada por:

$$F_X(x) = \int_0^x f_X(t)dt = \int_0^x \lambda e^{-\lambda t} dt = -\left[e^{-\lambda t}\right]_0^x$$

$$F_X(x) = 1 - e^{-\lambda x}, \quad x \geq 0$$

Para gerar amostras, igualamos a CDF a uma variável aleatória uniforme $U \sim \text{Unif}(0, 1)$:

$$U = 1 - e^{-\lambda x} \Rightarrow e^{-\lambda x} = 1 - U \Rightarrow -\lambda x = \ln(1 - U) \Rightarrow x = -\frac{1}{\lambda}\ln(1 - U)$$

Como $1 - U$ também é uma distribuição uniforme em $[0, 1]$:

$$\boxed{x = -\frac{1}{\lambda}\ln(U)}$$ ✓

2. Distribuição de Pareto com parâmetros $x_0 > 0$ e $\alpha > 0$, cuja função densidade é dada por $f_X(x) = \frac{\alpha x_0^\alpha}{x^{\alpha+1}}$, para $x \geq x_0$.

Resposta:

Analogamente ao item anterior:

$$F_X(x) = \int_{x_0}^x f_X(t)dt = \int_{x_0}^x \frac{\alpha x_0^\alpha}{t^{\alpha+1}} dt = -\left[\frac{x_0^\alpha}{t^\alpha}\right]_{x_0}^x$$

$$F_X(x) = 1 - \left(\frac{x_0}{x}\right)^\alpha, \quad x \geq x_0$$

$$U = 1 - \left(\frac{x_0}{x}\right)^\alpha \Rightarrow \left(\frac{x_0}{x}\right)^\alpha = 1 - U \Rightarrow x = \frac{x_0}{(1 - U)^{1/\alpha}}$$

Como $1 - U$ também é uma distribuição uniforme em $[0, 1]$:

$$\boxed{x = \frac{x_0}{(U)^{1/\alpha}}}$$ ✓

Questão 7: Contando domínios na Web

Quantos domínios web existem dentro da UFRJ? Mais precisamente, quantos domínios existem dentro do padrão de nomes `http://www.[a-z](k).ufrj.br`, onde $[a-z](k)$ é qualquer sequência de caracteres de comprimento k ou menor? Construa um algoritmo de Monte Carlo para estimar este número.

1. Descreva a variável aleatória cujo valor esperado está relacionado com a medida de interesse. Relacione analiticamente o valor esperado com a medida de interesse.

Resposta:

Seja Ω o conjunto de todas as palavras com até k letras minúsculas (de a a z). O número total de possíveis nomes é:

$$|\Omega| = \sum_{i=1}^{k} 26^i$$

Queremos estimar quantos desses nomes realmente correspondem a domínios válidos da forma `www.[palavra].ufrj.br`.

Para isso, definimos uma variável aleatória X da seguinte forma: sorteamos uma palavra $\omega \in \Omega$ de forma uniforme. Se o domínio `www.`ω`.ufrj.br` existe, definimos $X = |\Omega|$; caso contrário, definimos $X = 0$. Ou seja,

$$X = |\Omega| \cdot I_{\{\text{domínio existe}\}},$$

onde I é a função indicadora: ela vale 1 se o domínio existe, e 0 caso contrário.

Nesse modelo, a esperança de X é:

$$\mathbb{E}[X] = |\Omega| \cdot p,$$

onde p representa a probabilidade de que uma palavra sorteada aleatoriamente de Ω corresponda a um domínio que realmente existe. Como estamos sorteando de forma uniforme, essa probabilidade é igual à razão entre o número de domínios válidos (D) e o tamanho total de Ω:

$$p = \frac{D}{|\Omega|}$$

Portanto:

$$\mathbb{E}[X] = D$$

Assim, o valor esperado de X é igual ao número de domínios válidos D. ✓

2. Implemente o método de Monte Carlo para gerar amostras e estimar a medida de interesse. Para determinar o valor de uma amostra, você deve consultar o domínio gerado para determinar se o mesmo existe (utilize uma biblioteca web para isto).

Resposta:

- Definição do estimador:

  Seja M_n a média amostral de X após n amostras, dada por:

  $$M_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

  Pela Lei dos Grandes Números, temos que, para n suficientemente grande, a média amostral converge para o valor esperado. Assim, o estimador de D é dado por:

  $$D = \mathbb{E}[X] \approx M_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

- Código em Python:

```python
# Gerador de palavra aleatória com 1 até k letras minúsculas
def gerar_palavra_aleatoria(k):
    tamanho = random.randint(1, k)
    return ''.join(random.choices(string.ascii_lowercase, k=
        tamanho))

# Verifica se o domínio existe consultando o DNS
def dominio_existe(nome):
    try:
        socket.gethostbyname(f"www.{nome}.ufrj.br")
        return True
    except socket.gaierror:
        return False

# Estimativa Monte Carlo de domínios válidos
def estimar_dominios_validos(k, n_amostras):
    omega = sum([26**i for i in range(1, k+1)])
    soma = 0
    for i in range(n_amostras):
        palavra = gerar_palavra_aleatoria(k) # Gera palavra aleat
            ória de tamanho k
        existe = dominio_existe(palavra) # Verifica se o domínio
            existe
        x = omega if existe else 0
        soma += x
    return soma / n_amostras
```

3. Assuma que $k = 4$. Seja $\hat{w}_n$ o valor do estimador do número de domínios após n amostras. Trace um gráfico em escala semi-log (eixo-x em escala log) de $\hat{w}_n$ em função de n para

$n = 1, \ldots, 10^5$ (ou mais, se conseguir). O que você pode dizer sobre a convergência de $\hat{w}_n$?

Resposta:

Para traçar o gráfico, a função `estimador_dominios_validos` foi modificada para salvar em uma lista o valor de $\hat{w}_n$ a cada iteração. O gráfico foi gerado utilizando a biblioteca Matplotlib.

```python
# Estimativa Monte Carlo de domínios válidos salvando E_n a cada n
def estimar_dominios_validos_com_grafico(k, n_amostras):
    omega = sum([26**i for i in range(1, k+1)])
    soma = 0
    E_n = []
    for i in range(n_amostras):
        palavra = gerar_palavra_aleatoria(k) # Gera palavra aleatória
            de tamanho k
        existe = dominio_existe(palavra) # Verifica se o domínio
           existe
        x = omega if existe else 0
        soma += x
        E_n.append(soma / (i + 1))
    return E_n
```

O gráfico abaixo mostra a convergência do estimador $\hat{w}_n$ em função de n. A medida de interesse, o número de domínios válidos, converge para ≈ 3232 à medida que o número de amostras aumenta. Isso indica que o estimador é consistente e converge para o valor esperado. ✓

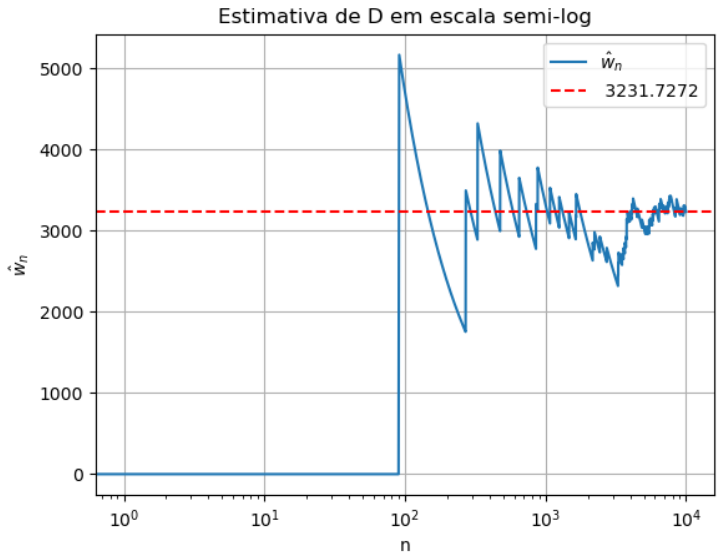

Figura 2: Estimador de D em função de n

Questão 8: Rejection Sampling

Considere o problema de gerar amostras de uma v.a. $X \sim \text{Binomial}(1000, 0.2)$.

1. Descreva uma proposta simples de função de probabilidade para gerar amostras de X usando Rejection Sampling. Calcule a eficiência dessa proposta.

Resposta:

A v.a. X, com distribuição binomial com parâmetros $n = 1000$ e $p = 0.2$, possui média $\mu = np = 200$ e variância $\sigma^2 = np(1-p) = 160$ (desvio padrão ≈ 12.65). A distribuição tem forma aproximadamente simétrica, com a maior parte da massa concentrada no intervalo $\mu \pm 4\sigma \approx [150, 250]$.

Seja $q(x)$ a função de probabilidade da distribuição-alvo (binomial) e $p(x)$ a função de densidade da distribuição proposta. Para aplicar Rejection Sampling, escolhemos uma distribuição uniforme discreta no intervalo $[150, 250]$:

$$p(x) = \begin{cases} \frac{1}{101}, & x \in [150, 250] \\ 0, & \text{caso contrário} \end{cases}$$

Para garantir que $q(x) \leq c \cdot p(x)$ para todo x, tomamos c como:

$$c = \max_{x \in [150,250]} \frac{q(x)}{p(x)} = 101 \cdot \max_{x \in [150,250]} q(x)$$

Como a binomial atinge seu valor máximo (moda) em $x = (n+1)p \approx 200$, temos:

$$q(200) = \binom{1000}{200} (0.2)^{200} (0.8)^{800} \approx 0.028$$

Logo, a constante é:

$$c = 101 \cdot 0.028 \approx \underline{2.828}$$

A eficiência do método é o inverso da constante de rejeição:

$$\text{Eficiência} = \frac{1}{c} \approx \frac{1}{2.828} \approx \underline{0.353}$$

Ou seja, cerca de 35.3% das amostras da proposta são aceitas. Isso indica que a escolha da proposta não é muito eficiente.

2. Lembrando que a distribuição Binomial tem a forma de sino, centrada em sua média, proponha outra função de probabilidade para gerar amostras de X usando Rejection Sampling. Calcule a eficiência dessa proposta e compare com a eficiência acima. O que você pode concluir?

Resposta:

A v.a. X, com distribuição binomial com parâmetros $n = 1000$ e $p = 0.2$, possui média $\mu = np = 200$ e variância $\sigma^2 = np(1-p) = 160$ (desvio padrão ≈ 12.65). A distribuição é aproximadamente simétrica e concentrada em torno da média.

Seja $q(x)$ a função de probabilidade da distribuição-alvo (binomial) e $p(x)$ a função de densidade da distribuição proposta. Para aplicar Rejection Sampling, escolhemos como proposta a distribuição normal contínua $\mathcal{N}(200, 160)$, avaliada em pontos inteiros.

A densidade da proposta é dada por:

$$p(x) = \frac{1}{\sqrt{2\pi \cdot 160}} \exp\left(-\frac{(x-200)^2}{2 \cdot 160}\right)$$

Para garantir que $q(x) \leq c \cdot p(x)$ para todo x, tomamos c como:

$$c = \max_x \left(\frac{q(x)}{p(x)}\right)$$

Para calcular c, precisamos encontrar o valor máximo de $\frac{q(x)}{p(x)}$ no intervalo de interesse, que não nescessariamente ocorre na média das duas distribuições, uma vez que o formato das caudas das distribuições são diferentes. Para isso, foi feito um código em Python que calcula o valor máximo de $\frac{q(x)}{p(x)}$ em um dado intervalo:

```python
# Parâmetros
n = 1000
p = 0.2
mu = n * p
sigma = np.sqrt(n * p * (1 - p))

# Intervalo de interesse
x = np.arange(150, 250)  # Valores inteiros de 150 a 250

# Distribuição Binomial (discreta)
q = binom.pmf(x, n, p)

# Distribuição Gaussiana (contínua, discretizada nos mesmos pontos)
p_vals = norm.pdf(x, mu, sigma)

# Encontrar a constante c (máximo da razão q/p)
ratio = q / p_vals
c_index = np.argmax(ratio)
c = ratio[c_index]
print(f"Constante c encontrada: {c}")
print(f"Ponto de máximo da razão: x = {x[c_index]}")
print(f"Eficiência: {1/c*100:.2f}%")
```

Resposta (continuação):

Para título de comparação entre este item e o anterior, foi escolhido o mesmo intervalo, $[150, 250]$ (aproximadamente 4 desvios padrão em torno da média) O valor máximo da razão $\frac{q(x)}{p(x)}$ encontrado neste intervalo foi em $x = 249$, com valor $c \approx 1.3879$. Assim, a eficiência do método é:

$$\text{Eficiência} = \frac{1}{c} \approx \frac{1}{1.3879} \approx 0.7205$$

Ou seja, cerca de 72.05% das amostras da proposta são aceitas. Isso indica que a escolha da proposta é mais eficiente do que a proposta anterior, que tinha eficiência de 35.3%. Portanto, a distribuição normal é uma escolha melhor para gerar amostras da binomial nesse caso.

Na figura abaixo é possível visualizar a comparação entre as duas distribuições.

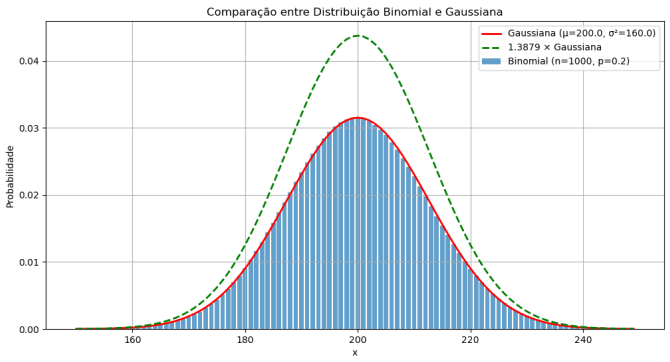

Figura 3: Comparação entre as distribuições Binomial e Normal

Questão 9: Integração de Monte Carlo e Importance Sampling

Considere a função $g(x) = e^{-x^2}$ e a integral de $g(x)$ no intervalo $[0, 1]$.

1. Implemente um método de Monte Carlo simples para estimar o valor da integral.

Resposta:

- **Definição do estimador:** Seja a integral de $g(x)$ definida por:

$$I = \int_0^1 g(x)dx$$

Seja X uma v.a. uniforme em $[0, 1]$. O valor esperado de $g(X)$ é dado por:

$$E[g(X)] = \int_0^1 f_X(x)g(x)dx = \int_0^1 \frac{1}{1-0}g(x)dx = I$$

Pelo método de Monte Carlo, temos, para $n \to \infty$ amostras:

$$E[g(X)] \approx M_n = \frac{1}{n}\sum_{i=1}^{n} g(X_i)$$

onde M_n é a média amostral de $g(X)$ e X_i é uma amostra de X.

Assim, para n suficientemente grande, o estimador de I é dado por:

$$\hat{I}_n = \frac{1}{n}\sum_{i=1}^{n} g(X_i)$$

- **Código em Python:**

```python
# Calculando a integral pelo Método de Monte Carlo
def monte_carlo_integration(a, b, n):
    soma=0
    for i in range(n):
        x = np.random.uniform(a, b)
        y = g(x)
        soma += y
    return soma/n
```

O valor obtido para $n = 10^3$ foi $\hat{I}_n = 0.759248$. Para avaliar esse resultado, a integral também foi calculada pela Regra de Simpson, com $n = 10^6$, que retornou o valor $I = 0.746824$. O erro relativo foi de aproximadamente 1.66%, o que demonstra que o método de Monte Carlo é eficaz para estimar a integral, mesmo com um número relativamente pequeno de amostras.

2. Intuitivamente, muitas amostras de $g(x)$ vão ter valores muito baixos. Dessa forma, utilize Importance Sampling para melhorar a qualidade do estimador do valor da integral. Em particular, utilize a função de densidade $h(x) = Ae^{-x}$ definida em $[0, 1]$ onde A é o valor da constante de normalização. Mostre como gerar amostras de $h(x)$.

Resposta:

- **Definição do estimador com *Importance Sampling*:**

$$E_h\left[\frac{g(X)}{h(X)}\right] = \int_0^1 \frac{g(x)}{h(x)} h(x)dx = \int_0^1 g(x)dx = I$$

Aplicando Monte Carlo:

$$E_h\left[\frac{g(X)}{h(X)}\right] \approx M_n = \frac{1}{n}\sum_{i=1}^{n} \frac{g(X_i)}{h(X_i)}$$

Logo, o estimador da integral I é:

$$\hat{I}_n = \frac{1}{n}\sum_{i=1}^{n} \frac{g(X_i)}{Ae^{-X_i}} = \underline{\frac{1}{nA}\sum_{i=1}^{n} g(X_i)e^{X_i} \quad |}$$

- **Determinação da constante de normalização:**

Para determinar a constante de normalização A, precisamos garantir que a integral de $h(x)$ no intervalo $[0, 1]$ seja igual a 1, ou seja:

$$\int_0^1 h(x)dx = \int_0^1 Ae^{-x}dx = A\left[-e^{-x}\right]_0^1 = A(1 - e^{-1}) = 1$$

$$A = \frac{1}{1 - e^{-1}} \approx \underline{1.581977 \quad |}$$

- **Gerando amostras a partir da transformada inversa:**

A função de distribuição acumulada (CDF) de $h(x)$ é dada por:

$$H(x) = \int_0^x h(t)dt = \int_0^x Ae^{-t}dt = -A\left[e^{-t}\right]_0^x = A(1 - e^{-x})$$

Igualando a CDF a uma variável aleatória uniforme $U \sim \text{Unif}(0, 1)$:

$$U = A(1 - e^{-x}) \Rightarrow 1 - e^{-x} = \frac{U}{A} \Rightarrow e^{-x} = 1 - \frac{U}{A} \Rightarrow -x = \ln\left(1 - \frac{U}{A}\right)$$

$$x = -\ln\left(1 - \frac{U}{A}\right) = -\ln\left(\frac{A - U}{A}\right) = \ln\left(\frac{A}{A - U}\right) = \underline{\ln(A) - \ln(A - U) \quad |}$$

Resposta (continuação):

- **Código em Python:**

```python
def is_integration(a, b, n):
    A = 1/(1 - np.exp(-1))
    soma=0
    for i in range(n):
        U = np.random.uniform(a, b)
        x = np.log(A)-np.log(A-U)
        soma += g(x) * np.exp(x)
    return soma/(n*A)
```

O valor obtido para $n = 10^3$ foi $\hat{I}_n = 0.744836$. Para avaliar esse resultado, a integral também foi calculada pela Regra de Simpson, com $n = 10^6$, que retornou o valor $I = 0.746824$. O erro relativo foi de aproximadamente 0.27%, o que demonstra que o método de Monte Carlo com Importance Sampling também é eficaz para estimar a integral, mesmo com um número relativamente pequeno de amostras.

3. Compare os dois métodos. Trace um gráfico do erro relativo de cada um dos estimadores em função do número de amostras. Ou seja, $|\hat{I}_n - I|/I$ onde I é o valor exato da integral e $\hat{I}_n$ é o valor do estimador com n amostras, para $n = 10^1, 10^2, \ldots, 10^6$.

Resposta:

Código em Python:

A função abaixo foi implementada para calcular o erro relativo de cada um dos estimadores a cada iteração:

```python
def plot_function(n, I):
    A = 1/(1 - np.exp(-1))
    soma_mc=0
    soma_is=0
    erro_mc=[]
    erro_is=[]
    for i in range(n):
        # Gerando um número aleatório U entre 0 e 1
        U = np.random.uniform(0, 1)
        # Calculando a integral pelo método de Monte Carlo
        x_mc = U
        soma_mc += g(x_mc)
        I_mc = soma_mc/(i+1)
        erro_mc.append(abs(I-I_mc)/I)
        # Calculando a integral pelo método de Monte Carlo com
            importance sampling
        x_is = np.log(A)-np.log(A-U)
        soma_is += g(x_is) * np.exp(x_is)
        I_is = soma_is/((i+1)*A)
        erro_is.append(abs(I-I_is)/I)
```

Resposta (continuação):

No gráfico abaixo, é possível obseervar que o método de Monte Carlo com Importance Sampling converge bem mais rápido do que o método de Monte Carlo simples, evidenciando a maior eficiência do método.

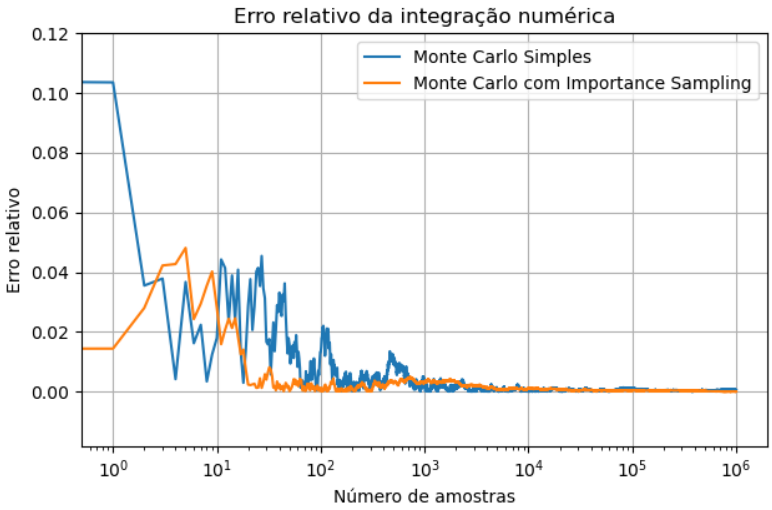

Figura 4: Erro relativo dos estimadores em função de n

Códigos

Os códigos utilizados para a resolução dos exercícios estão disponíveis no repositório do GitHub: `https://github.com/lhscaldas/CPS767_MCMC/`