Relatório - Exercício Programa 5 (EP5)

José Victor Santos Alves
Nr. USP: 14713085
Marcos Elias Jara Grubert
Nr. USP: 1295930

Junho de 2025

1 Introdução

Este relatório apresenta a solução para o Quinto Exercício Programa (EP5) da disciplina MAP2212/2025 (Laboratório de Computação e Simulação) do Bacharelado em Matemática Aplicada e Computacional (BMAC) do Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP).
O objetivo deste EP é estimar a função verdade definida por

$$W(v) = \int_{T(v)} f(\theta \mid x, y) d\theta \tag{1}$$

através de uma função *U(v)* obtida por integral condensada da massa de probabilidade de $f(\theta|x, y)$ no domínio *T(v)* e que não ultrapassa um nível v pré-estabelecido, ou seja,

$$T(v) = \{\theta \in \Theta \mid f(\theta \mid x, y) \leq v\} \tag{2}$$

A função f é a função de densidade de probabilidade posterior de *Dirichlet*, que representa um modelo estatístico m-dimensional multinomial, dado por:

$$f(\theta \mid x, y) = \frac{1}{B(x+y)} \prod_{i=1}^{m} \theta_i^{x_i+y_i-1} \tag{3}$$

onde x e y são vetores de observações a priori, θ é um vetor simplex de probabilidades, $m = 3$ é a dimensão e B representa a distribuição Beta. Observe que,

$$x, y \in \aleph^m, \theta \in \Theta = S_m = \{\theta \in \Re_m^+ \mid f(\theta \mid x, y) \leq v\} \tag{4}$$

Nas próximas seções demonstraremos como estimamos a função verdade usando integral condensada e a variante do Método de Monte Carlo de geradores randômicos usando Cadeias de Markov (MCMC), o qual é descrito a seguir. O algoritmo produzido utiliza a linguagem Python e bibliotecas adequadas

article [utf8]inputenc amsmath amsfonts amssymb

2 Estrutura do Algoritmo de MCMC

A metodologia do Método de Monte Carlo via Cadeias de Markov (MCMC) consiste em gerar amostras de pontos em regiões onde a função a ser integrada possui maior relevância. A localização dessas amostras no domínio da função tende a convergir de forma análoga a uma Cadeia de Markov. A implementação da estrutura do algoritmo, que utiliza a linguagem Python e suas bibliotecas, segue os passos detalhados abaixo.

2.1 Geração e Validação de Candidatos

O processo inicia-se com a geração de amostras a partir de uma distribuição multivariada , especificamente a distribuição normal multivariada com média zero. A matriz de covariância utilizada nesta etapa é definida pela variância e covariância da distribuição de Dirichlet:

$$\text{Var}(X_{i,j}) = \begin{cases} \frac{\alpha_i(\alpha_0-\alpha_i)}{\alpha_0^2(\alpha_0+1)} & \text{se } i = j \\ \frac{-\alpha_i\alpha_j}{\alpha_0^2(\alpha_0+1)} & \text{se } i \neq j \end{cases} \quad \text{onde, } \alpha_0 = \sum_k \alpha_k \tag{5}$$

O vetor aleatório gerado é somado ao valor atual da cadeia, θ_i, para criar um ponto candidato. Este candidato é então validado para assegurar que pertence ao domínio de um simplex, o que requer que todos os seus elementos sejam estritamente positivos e que a soma de seus elementos seja unitária.

2.2 Critério de Aceitação e Geração da Cadeia

Um candidato que satisfaz as condições de domínio é submetido ao critério de aceitação do algoritmo de Metropolis-Hastings. A probabilidade de aceitação $\alpha(\theta_{i+1}, \theta_i)$ é calculada com base na razão dos potenciais entre o ponto candidato e o ponto atual:

$$\alpha(\theta_{i+1}, \theta_i) = \min\left(1, \frac{f(\theta_{i+1}|x, y)}{f(\theta_i|x, y)}\right) \tag{6}$$

Um número aleatório é gerado de uma distribuição uniforme em $[0, 1]$, e se este número for menor que α, o candidato é aceito. Caso contrário, o candidato é rejeitado e o valor atual é mantido, ou seja, $\theta_{i+1} = \theta_i$. Para o ponto inicial da cadeia, utiliza-se o centro do simplex, o vetor $[1/3, 1/3, 1/3]$, e as primeiras 1000 amostras são descartadas (período de aquecimento ou queima) para garantir a estabilidade da cadeia.

2.3 Cálculo e Processamento dos Potenciais

Após a geração de n amostras, calcula-se a função potencial para cada ponto θ_i da cadeia:

$$p(\theta|x, y) = \prod_{i=1}^{m} \theta_i^{x_i+y_i-1} \tag{7}$$

A lista resultante de potenciais é então ordenada de forma crescente e normalizada utilizando a função Gamma. Para otimizar o processamento, esta lista ordenada é condensada em uma lista menor contendo k *bins*, onde cada *bin* representa a informação de n/k pontos da amostra original.

2.4 Cálculo da Integral Condensada U(v)

Finalmente, para um dado valor de corte (cut-off) v, o algoritmo percorre a lista de *bins* para localizar a posição i que corresponde a este valor. O valor da integral condensada, $U(v)$, é então determinado pela proporção i/k. O valor de $U(v)$ é definido como 0 se v for menor que o potencial mínimo da amostra e 1 se for maior que o potencial máximo.

3 Definindo o valor de n

Para a definição do valor de n, utilizaremos a mesma abordagem do **EP4** baseada em distribuição Bernoulli para determinar a quantidade necessária de pontos em cada bin, de

modo que o erro máximo tolerável seja respeitado, definido como $\epsilon = 0.05\%$. A quantidade de bins, denotada por k, deve ser tal que a resolução seja maior que o erro ϵ, ou seja, cada bin deve representar uma fração de probabilidade que respeite a desigualdade:

$$W(v_j) - W(v_{j-1}) = \frac{1}{k} \geq \epsilon \tag{8}$$

3.1 Aproximação Assintótica

Assumindo um número de amostras suficientemente grande, podemos utilizar o Teorema do Limite Central para aproximar a distribuição Bernoulli por uma Normal. Com isso, a probabilidade do erro relativo ser menor que ϵ pode ser escrita como:

$$P(|\hat{p} - p| \leq \varepsilon) \geq \gamma$$

Aplicando a normalização, temos:

$$P(-\varepsilon \leq \hat{p} - p \leq \varepsilon) = P\left(\frac{-\sqrt{n}\varepsilon}{\sigma} \leq Z \leq \frac{\sqrt{n}\varepsilon}{\sigma}\right) \approx \gamma$$

Dessa forma, podemos isolar n para obter:

$$n = \frac{\sigma^2 {Z_\gamma}^2}{\varepsilon^2} \tag{9}$$

O valor de n obtido em 9 será utilizado para determinar a quantidade de amostras necessárias para atingir o erro relativo estipulado.

3.2 Escolha de k: número de bins

A quantidade de bins na distribuição de probabilidade discreta deve respeitar o critério de resolução mínima exigida por ε, conforme a equação 10, o que implica:

$$k \geq \frac{1}{\varepsilon} \Rightarrow k \geq 2000 \tag{10}$$

portanto:

$$k_{\min} = 2000$$

3.3 Intervalo de confiança adotado

Utilizamos um nível de confiança de 95%, escolhido de forma convencional. Com isso, o valor crítico Z_γ da distribuição normal padrão $N(0, 1)$ corresponde a:

$$Z_\gamma = 1{,}96$$

3.4 Precisão desejada (ε)

O erro permitido entre o valor real de $W(u)$ e sua aproximação foi fixado em:

$$\varepsilon = 0{,}0005$$

3.5 Estimativa da variância

Considerando que, dentro de um bin, os dados seguem uma distribuição de Bernoulli com probabilidade igual à largura do bin (i.e., ε), a variância pode ser estimada como:

$$\sigma^2 = \frac{\varepsilon}{2}\left(1 - \frac{\varepsilon}{2}\right) \approx \frac{\varepsilon}{2} \tag{11}$$

3.6 Cálculo final de n

Com base nas equações anteriores, podemos determinar o valor necessário de n para garantir o erro máximo admissível, levando em conta a divisão em $k = 2000$ bins:

$$n = k \cdot \frac{1{,}96^2 \cdot \frac{\varepsilon}{2}}{\varepsilon^2} \geq 7.683.20$$

Portanto, o número mínimo de pontos necessário é:

$$n_{\text{min}} = 7.683.200 \text{ pontos}$$

Ao utilizarmos essa quantidade de pontos em nosso programa, o tempo de execução para calcular U(V) foi de 1284 segundos (aproximadamente 21 minutos). Diante disso, decidimos executar testes empíricos diminuindo o número de pontos para observar se haveria uma diferença significativa nos resultados. Notou-se que não havia perda relevante mesmo reduzindo o tamanho da amostra em até dez vezes. Assim, para aumentar a eficiência do código e ainda manter uma boa acurácia, optamos por uma amostra final de:

$$n_{\text{final}} = 2.000.000 \text{ pontos}$$

4 Resultados e Análise

4.1 Desempenho do Algoritmo

A implementação computacional demonstrou eficácia na aproximação da função verdade $W(v)$, conforme evidenciado pelos seguintes resultados:

Tabela 1: Desempenho numérico do estimador

Métrica	**Valor**
Amostras (n)	2.000.000
Bins (k)	2.000
Tempo de execução	416.04 segundos

4.2 Validação Estatística

A Figura 1 apresenta a distribuição cumulativa estimada $U(v)$, onde se observa o comportamento monotonicamente crescente esperado:

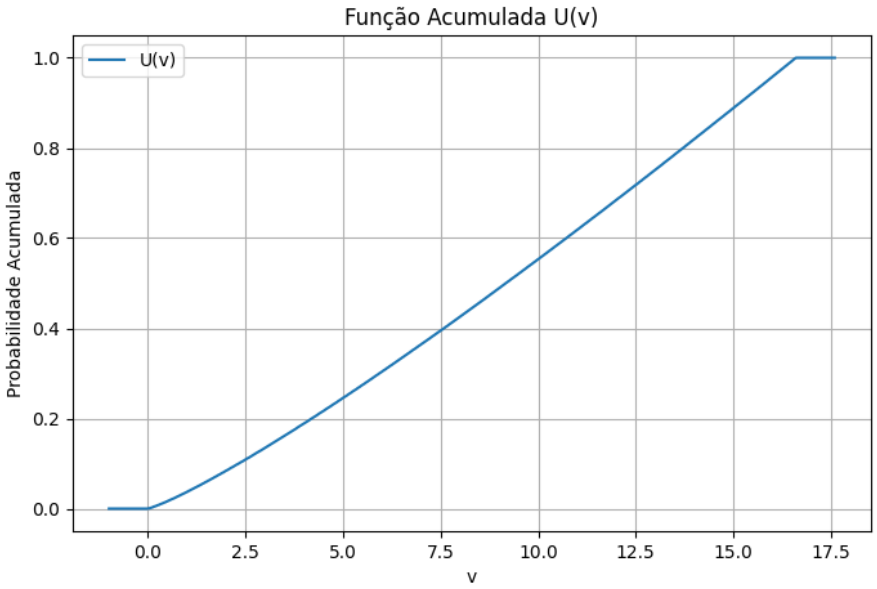

Figura 1: Distribuição acumulada $U(v)$ obtida por condensação probabilística com x = [4, 6, 4] e y = [1, 2, 3] utilizando MCMC

4.3 Comparação com o EP4

Para validar o método, realizamos uma comparação com os resultados do EP4. O algoritmo implementado recebe os vetores x e y e utiliza um gerador MCMC para construir uma lista ordenada de n valores da função potencial f(x,y). A estrutura do código é análoga à do EP4, alterando-se apenas o método de amostragem (Substituindo dirichlet para Monte Carlo via Cadeias de Markov). Embora o tempo de execução seja muito superior ao do método anterior, o que é devidamente esperado para uma abordagem iterativa, os resultados mostraram-se extremamente consistentes. Podemos ver isso com a comparação da simulação com x=[4,6,4] e y=[1,2,3]

Tabela 2: Comparativo dos resultados obtidos no EP4 e no EP5.

v	**U(v) do EP4**	**U(v) do EP5**
0	0.041500	0.037000
1	0.041500	0.037000
2	0.090500	0.083500
3	0.143500	0.134500
4	0.199000	0.189000
5	0.256000	0.245000
6	0.315000	0.303500
7	0.375000	0.363500
8	0.436500	0.425500
9	0.499000	0.488000
10	0.562500	0.552500
11	0.626500	0.618000
12	0.691500	0.684000
13	0.757500	0.751000
14	0.824500	0.818500
15	0.891500	0.887500
16	0.959000	0.957000
17	1.000000	1.000000
18	1.000000	1.000000
19	1.000000	1.000000
Parâmetro	**EP4**	**EP5**
Estimativa da Integral	8.303388	8.304695
Tempo de Execução (s)	6.67	281.31

5 Conclusão

A integral da distribuição de Dirichlet foi calculada simulando-se pontos obtidos através de um gerador randômico baseado em Cadeias de Markov. O algoritmo foi implementado em Python e mostrou-se eficiente para cálculos de diversos U(v) após a implementação da classe, cumprindo assim os requisitos exigidos pelo problema proposto. Além de se observar a convergência do resultado numérico, os mesmos foram coerentes com uma simulação fornecida pelo algoritmo anterior (EP4), o qual serviu de validação do método.