MAP2212 - EP5 Estimativa de uma integral com base em geradores MCMC

Vinícius da Costa Collaço - 11811012 Nikolas Lukin - 5381328

13 de junho de 2022

Instituto de Matemática e Estatística

Universidade de São Paulo

Junho de 2022

Sumário

1 Introdução

Esse relatório apresenta a solução do quinto exercício programa (EP5) proposto na matéria MAP2212/2022 (Laboratório de Computação e Simulação) do curso de bacharelado de matemática aplicada e computacional (BMAC) do instituito IME-USP. O objetivo deste EP é estimar a função verdade definida pela integral da distribuição de Dirichlet,

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

através de uma função aproximada *U(v)* obtida pela integral condensada da massa de probabilidade[3] de $f(\theta|x, y)$ no domínio *T(v)* e que não ultrapassa um nível v pré-estabelecido (cut-off), 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 é um vetor de observações, y é um vetor 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[1, 6, 2, 5],

2 Estrutura do algoritmo de MCMC

A idéia por trás do Método de Monte Carlo usando Cadeias de Markov é gerar amostras de pontos para o cálculo de uma integral em locais onde a função a ser integrada seja mais relevante. A localização destas amostras no domínio da função tendem a se convergir como em uma Cadeia de Markov. Inicialmente, algoritmo de MCMC cria amostras randômicas a partir de uma distribuição multivariada. Neste trabalho usamos a distribuição multivariada normal para gerar amostras de θ com média zero e matriz de variância

definida por,

$$Var(X_{i,j}) = \begin{cases} \frac{\alpha_i(\alpha_0-\alpha_i)}{\alpha_0^2(\alpha_0+1)} & i = j \\ \frac{-\alpha_i\alpha_0}{\alpha_0^2(\alpha_0+1)} & i \neq j \end{cases} \tag{5}$$

onde, $\alpha_0 = \sum_{i=1}^{k} \alpha_i$ Esse vetor é adicionado à um valor inicial de θ_i para tornar-se um candidato de amostra, o qual é validado pela verificação de pertencer ao domínio de um simplex:

- Soma dos elementos do vetor é unitária
- Os elementos do vetor são estritamente positivos

Caso as condições de domínio estejam satisfeitas, o algoritmo decide por aceitar ou rejeitar esse candidato (θ_{i+1}com base no critério de balanço detalhado do algoritmo de Metrópolis-Hastings. Segundo este algoritmo, as amostras serão aceitas com base na probabilidade relativa na função a qual se deseja integrar. Ela será aceita caso um número aleatório uniforme em [0,1] seja menor que $\alpha(\theta_{i+1}, \theta_i)$, o qual é definido por,

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

Caso contrário, θ_{i+1} continuará com o valor de θ_i. Após gerados os n valores da amostra, inicia-se o calculo da função potencial $p(\theta_i \mid x, y)$ correspondente à cada amostra,

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

Finalmente a lista de potenciais gerados para cada amostra é ordenada e normalizada pela função Gamma. Esta lista é condensada em uma lista menor com k bins, os quais cada um condensa a informação de n/k pontos. Finalmente, o algoritmo percorre esta última lista e localiza a posição i correspondente ao cut-off v imposto na entrada. O valor calculado da integral condensada *U(v)* é obtido por,

$$\begin{cases} U(v) = 0, & v \leq min(f(\theta \mid x, y)) \\ U(v) = \frac{i}{n} & min(f(\theta \mid x, y)) \leq v \leq max(f(\theta \mid x, y)) \\ U(v) = 1, & v \geq max(f(\theta \mid x, y)) \end{cases} \tag{8}$$

Resumidamente, o algoritmo está estruturado nas seguintes funções:

- Declaração de variáveis e estruturação de objetos
- Cálculo da matriz de covariância

- Gerador de amostras
- Cálculo da função potencial das amostras
- Ordenação e normalização da lista
- Condensação da lista em bins
- Cálculo da probabilidade acumulada para um dado v

2.1 Definindo o valor inicial e de aquecimento da cadeia

Para valor inicial utilizou-se o valor central do simplex, ou seja, o vetor [1/3, 1/3, 1/3] e para aquecimento da cadeia é necessário o descarte das amostras iniciais, observou-se empiricamente que com uma queima de 1000 valores a cadeia se comportava como o esperado, definindo então 1000 como valor de queima.

3 Definindo a quantidade de iterações n

Utilizamos o mesmo método de cálculo da quantidade de pontos n que serão utilizados na simulação que no EP4. Para tanto, considera-se a aproximação assintótica de uma distribuição Bernoulli para determinar a quantidade de pontos dentro de um bin necessários para satisfazer a resolução e o erro máximo tolerável.

3.1 Erro (ε)

Define-se por erro (ε) a precisão esperada do cálculo, ou seja, o maior desvio tolerável entre o valor obtido por aproximação $U(v)$ e o valor exato da função verdade $W(v)$. Neste trabalho, foi considerado $\varepsilon = 0,0005$

3.2 Número de bins (k)

Para efeitos práticos os dados da simulação são condensados em bins[3]. A quantidade k de bins da distribuição discreta de probabilidades deve ser tal a garantir uma resolução que seja maior que o erro ϵ, definido em 10

$$W(v_j) - W(v_{j-1}) = \frac{1}{k} \geq \epsilon \tag{9}$$

Portanto, para que a resolução seja maior que o erro estipulado,

$$k \geq \frac{1}{\epsilon} \implies k \geq 2000 \tag{10}$$

Para melhorar a resolução, considerou-se utilizar o dobro do mínimo de bins necessários, ou seja,

$$k = 4000$$

3.3 Distribuição dos dados na amostra

Inicialmente, supõe-se que os dados estão distribuídos segundo uma distribuição de Bernoulli dentro dos bins. No entanto, caso o tamanho da amostra seja suficientemente grande, pelo teorema do limite central podemos aproximar esta Bernoulli por uma normal, obtendo:

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

[4]

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

A estimativa para a quantidade total de pontos n a serem simulados é portanto,

$$n = \frac{\sigma^2 {Z_\gamma}^2}{\varepsilon^2} \tag{11}$$

O resultado obtido em 11 será utilizado para definir o número de pontos necessários para resultar no erro mínimo exigido. Na modelagem normal da distribuição de dados do problema, consideramos um intervalo de confiança $\gamma = 95\%$, escolhido arbitrariamente. Obtêm-se assim $Z_\gamma = 1,96$ para $N(0,1)$,

3.4 Variância amostral

Para o cálculo da variância, considera-se que os pontos se distribuem segundo uma distribuição de Bernoulli dentro de um bin com probabilidade igual à resolução do bin ($= \frac{1}{k}$). Com o valor de $k = 4000$ variância é determinada por,

$$\sigma^2 = \frac{\epsilon}{2}(1 - \frac{\epsilon}{2}) \approx \frac{\epsilon}{2} \tag{12}$$

3.5 Valor final do n

O valor de n é obtido através das equações em 10 11 12 resultando em,

$$n = k \cdot \frac{1,96^2 \cdot \frac{\epsilon}{2}}{\epsilon^2} \geq 15.366.400$$

Porém observando os valores extremos dentro de um único bin empiricamente, observou-se que não havia diferença significante reduzindo em cerca de 10 vezes o tamanho da amostra. Então optou-se por reduzir amostra final cerca de 7 vezes, para aumentar a eficiência do código e ainda assim manter uma boa acurácia, obtendo no final.

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

4 Resultados e discussão

Para a simulação construiu-se um algoritmo que recebe como dados de entrada os vetores x e y, constrói e ordena uma lista com n entradas crescentes de $f(\theta|x, y)$ obtidas de simulações aleatórias de θ_i advindas de um gerador gerador baseado no MCMC como descrito anteriormente. Verificou-se que o algoritmo funcionou conforme o esperado, apesar de demorar mais para rodar que o anterior. Uma comparação dos resultados do algoritmo do EP4 com este algoritmo está mostrada a seguir para validação dos resultados desta simulação com $x = [4, 6, 4]$ e $y = [1, 2, 3]$.

v	U(v) EP4	U(v) EP5
0	0.00000	0.00000
1	0.04125	0.03700
2	0.09050	0.08300
3	0.14325	0.13450
4	0.19875	0.18875
5	0.25575	0.24575
6	0.31450	0.30425
7	0.37475	0.36450
8	0.43625	0.42600
9	0.49875	0.48900
10	0.56200	0.55350
11	0.62625	0.61850
12	0.69150	0.68475
13	0.75725	0.75150
14	0.82375	0.81925
15	0.89100	0.88800
16	0.95875	0.95750
17	1.00000	1.00000

Tabela 1: Validação de *U(v)*

A comparação dos gráficos resultante de *U(v)* de ambos EP4 e EP5 está ilustrada na Figura 1:

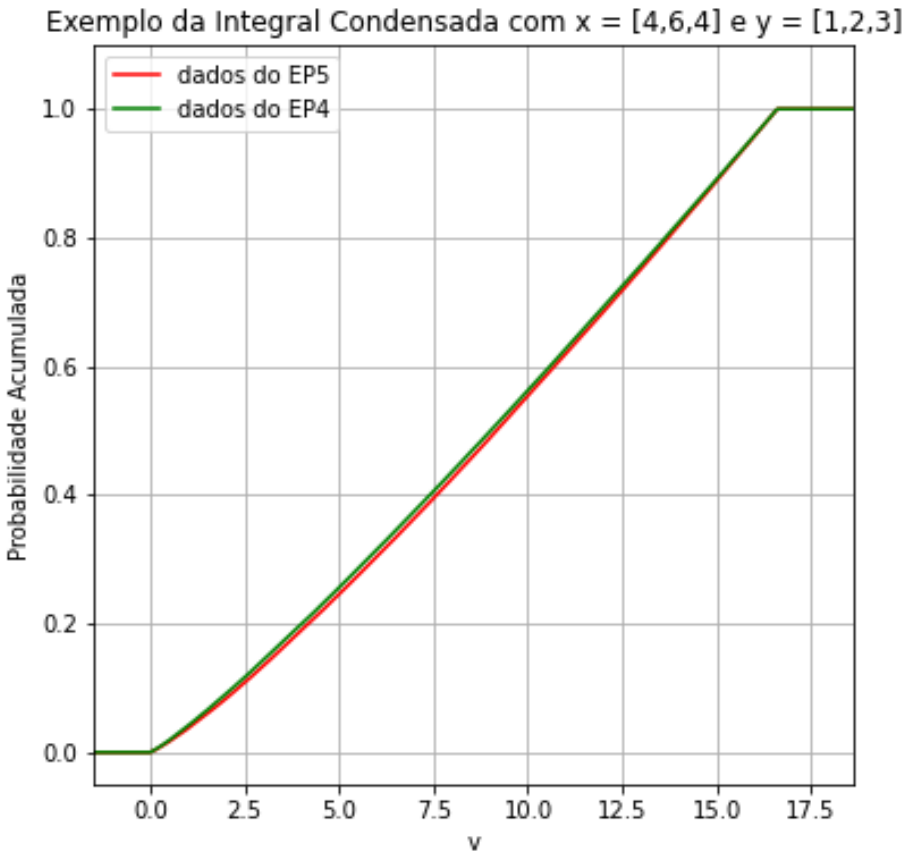

Figura 1: Gráfico de $U(v)$

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.

Referências

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] Stan Kaplan and James C Lin. An improved condensation procedure in discrete probability distribution calculations. *Risk Analysis*, 7(1):15–19, 1987.

[4] Wilton de O. Bussab Pedro A. Morettin. *Estatística Básica,* 6^a *Edição*. 2010.

[5] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[6] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.