

# Relatório Exercício Computacional 5

Leonardo Heidi Almeida Murakami - NUSP: 11260186

leonardo.murakami@usp.br

Instituto de Matemática e Estatística - Universidade de São Paulo

Neste trabalho utilizaremos de um modelo estatístico multinomial para aproximarmos uma função que possuímos alta dificuldade em calcular, sendo esta, neste exercício, a função verdade do modelo estatístico que será mais profundamente descrito abaixo.

## I. Introdução e Conceitos

### A. Modelo Estatístico

Consideraremos o modelo estatístico  $m$ -dimensional (para este exercício, utilizaremos  $m = 3$ ) Multinomial, com observações  $x$ , priori  $y$  e parâmetro  $\theta$ , onde

$$x, y \in N^m \quad (1)$$

$$\theta \in \Theta = S_m = \{\theta \in R_m^+ | \theta^T \mathbf{1} = 1\} \quad (2)$$

$$m = 3 \quad (3)$$

Com função de densidade potencial igual a

$$f(\theta|x, y) = \prod_{i=1}^m \theta_i^{x_i+y_i-1} \quad (4)$$

Que prova-se extremamente semelhante a uma distribuição de Dirichlet, com  $x+y$  tornando-se o  $\alpha$  da distribuição, que será a aproximação que utilizaremos para obtermos valores aleatórios desta distribuição.

Além disso, também temos a função que define o conjunto de corte de nosso modelo, tal que

$$T(v) = \{\theta \in \Theta | f(\theta|x, y) \leq v\}, v \geq 0 \quad (5)$$

que será utilizada para calcularmos a função verdade (objetivo desse exercício computacional)

$$W(v) = \int_{T(v)} f(\theta|x, y) d\theta \quad (6)$$

Onde  $W(v)$  é a massa da probabilidade a posteriori dentro de  $T(v)$ , ou seja, a massa de probabilidade onde a densidade de probabilidade  $f(\theta|x, y)$ , não excede o limite  $v$

### B. Método de Aproximação

Para aproximarmos a função  $W$ , utilizaremos  $k$  pontos de corte, que variam entre 0 e o limite superior de  $f(\theta)$  e geraremos  $n$  pontos  $\theta$  aleatórios distribuídos de acordo com nossa função definida acima e utilizaremos isto para definir  $k - 1$  "bins" de corte com pesos semelhantes (ou seja, que possuem mesma fração de pontos gerados, de modo que a fração de pontos dentro deste intervalo seja  $\frac{1}{k}$ ). Utilizaremos a fração de pontos ate aquele intervalo como uma aproximação para  $W(v)$

## C. Aproximação de $W(v)$

### 1. Gerando o espaço de thetas aleatórios

Assim como pedido pelo professor, utilizaremos um gerador de amostras de *Markov Chain Monte Carlo* (MCMC) baseado no conhecimento da potencial de Dirichlet. Como kernel do nosso MCMC utilizaremos a Normal Multivariada  $N(0, \Sigma)$  onde a matriz de covariância é calculada baseada no conhecimento *a priori* dos parâmetros de nossa Dirichlet (ou seja, nosso  $\alpha = X + Y$ )

Para calcularmos a matriz de covariância, utilizamos as formulas da própria distribuição de Dirichlet, onde

$$Var(X_i) = \frac{\alpha_i(\alpha_0 - \alpha_i)}{\alpha_0^2(\alpha_0 + 1)} \quad (7)$$

$$Cov(X_i, X_j) = \frac{-\alpha_i \alpha_j}{\alpha_0^2(\alpha_0 + 1)} \quad (8)$$

Onde geraremos uma matriz do tipo

$$\begin{bmatrix} Var(\alpha_1) & Cov(\alpha_1, \alpha_2) & Cov(\alpha_1, \alpha_3) \\ Cov(\alpha_2, \alpha_1) & Var(\alpha_2) & Cov(\alpha_2, \alpha_3) \\ Cov(\alpha_3, \alpha_1) & Cov(\alpha_3, \alpha_2) & Var(\alpha_3) \end{bmatrix} \quad (9)$$

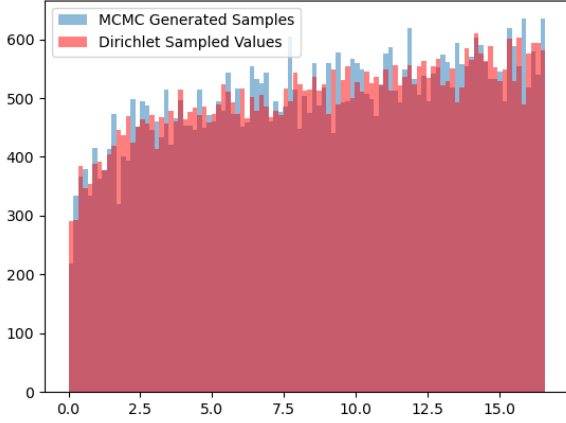
Que será utilizada no nosso kernel, uma Normal Multivariada ( $N(0, \Sigma)$ ).

Utilizamos um *burn-in* de 1000 iterações, número arbitrariamente escolhido para não possuímos quaisquer vieses quanto ao valor inicial utilizado pelo nosso MCMC, mas não grande o suficiente para causar impactos em nossa performance.

Como método de aproximação utilizaremos o Metropolis-Hasting, utilizando o kernel como passo aleatório entre os valores aleatório gerados e uma aceitação de

$$\min(1, \frac{f(\theta_{n-1} + N(0, \Sigma))}{f(\theta_{n-1})}) \quad (10)$$

Para verificar se o MCMC estava gerando amostras condizentes com a distribuição de Dirichlet passada, verificamos se a distribuição potencial de um numero grande de números gerados estavam próximas entre si, como visto na imagem a baixo.



### 2. Ajustando as bordas dinamicamente

Para que tenhamos bins com peso semelhantes, ou seja, todos próximos a  $1/k$  ajustaremos dinamicamente as bordas de cada bin. Para isso, ordenaremos todo o nosso espaço de  $f(\theta|x, y)$  em ordem crescente e realizaremos  $k$  cortes nesse array, de modo que cada pedaço possuirá  $n/k$  membros do vetor de  $f(\theta|x, y)$ , pegamos então as bordas direitas destes vetores e igualamos a cada ponto  $v$ . Deste modo teremos todos os bins balanceados.

### 3. Calculando uma aproximação para $W(v)$

Dado que temos todos os bins ajustados e os seus limites calculados precisamos agora criar uma função que corretamente simula o comportamento de  $W(v)$ .

Como indicado no enunciado do exercício, a fração de  $f(\theta)$  presentes até um ponto  $v_j$  é uma aproximação de  $W(v_j)$ , chamaremos essa função aproximada de  $U(v)$ , o que fazemos, para um dado ponto  $v$ , será somar as frações, ou seja, somar os bins (com suas fração de pontos) até que a borda de um bin seja maior que o  $v$  passado para a função

## II. Implementação e Testes

O algoritmo foi implementado na linguagem Python na versão 3.8.8, organizado em 1 arquivo: `statisticalModel.py`.

A álgebra com vetores, assim como a geração de números aleatórios a partir de uma distribuição (no caso a Gamma) foi feita através da biblioteca *NumPy*. O esqueleto do código pode ser encontrado na seção abaixo

### A. Arquivos do Projeto

Abaixo segue um breve resumo do conteúdo dos códigos fonte e interfaces. **`statisticalModel.py`**: Implementa todo o algoritmo. Separado em duas classes principais. Apresentarei apenas os métodos públicos das classes neste documento.

```
class Dirichlet(alpha: List[int]):
    def sample() -> List[float]:
```

```
class StatisticalModel(
    observations: List[int],
    prior: List[int],
    cut_off_points: int,
    theta_n_size: int=30000,
):
    def calculate_fraction_of_sets() ->
        Dict[(str, float)]:

    def U(v) -> float:
```

### B. Algoritmo

Explicando um pouco melhor cada método mais relevante do programa.

- *Dirichlet()*

- `.sample_metropolis_hasting()`: retorna uma lista com os valores de theta da distribuição de Dirichlet que modela nosso problema baseado no algoritmo do Metropolis-Hasting, explicado brevemente anteriormente (I.C.2)

- *StatisticalModel()*

- `.calculate_fraction_of_sets()`: calcula a fração de pontos do espaço de theta criado em cada bin do nosso modelo e retorna um dicionário que mostra o limite inferior e superior de cada bin como chave e a fração nele presente como valor.
- `._adjust_bounds_for_balanced_bins()`: método protegido que é chamado na instanciação da classe. Ele é responsável pelo ajuste dinâmico dos sets de corte.
- `.U(v)`: método que aproxima  $W(v)$  dado um valor de  $v$  e retorna o valor aproximado

## III. Conclusão

Após diversos testes com diversos valores diferentes de  $k$  e  $n$ , obtivemos um valor agradável de erro por volta de  $k = 50000$  e  $n = 300000$ .

Para estes valores o desvio padrão de um mesmo valor rodado diversas vezes com um modelo instanciado com variáveis aleatórias diferentes e independentes entre modelos, atingimos um desvio padrão menor que 0.05, desejado pelo enunciado do exercício.

Além disso, para iniciar o modelo estatístico (com o set dinâmico dos limites de cada *bin*) temos uma demora de aproximadamente 40.29 segundos, onde, após a instanciação da classe, demoramos um tempo infinitamente menor (2.57 nano segundos, calculado usando o Google Colabs com a utilidade `%timeit`) para obtermos um valor de  $U(v) \approx W(v)$ , isso utilizando variáveis de teste que não necessariamente representam o real valor a ser testado.