Exercício Programa 5

Ana Paula Lopes Cavalcante
Gabriel Tetsuo Haga

Junho 2021

1 Introdução

Esse relatório tem como objetivo, apresentar a função feita para a geração de amostras de variáveis aleatórias $\theta_i = (\theta_{i_1}, \theta_{i_2}, \theta_{i_1})$ que seguem uma distribuição de Dirichlet. Essa função é usada para substituir, dentro do código feito para o Exercício Programa 4 (EP4), a função `dirichlet` do módulo `random` da biblioteca `numpy`. Além disso a função se baseia no algoritmo de Metropolis-Hastings, um algoritmo que utiliza a ideia de cadeias de Markov no seu funcionamento.

Assim apresenta-se, primeiramente, uma breve explicação teórica das cadeias de Markov e do algoritmo de Metropolis-Hatings. Após isso, apresenta-se a implementação do algoritmo para criar a função e sua implementação dentro do código feito para o EP4. E ao final, é feito uma conclusão sobre os resultados obtidos.

2 Teoria

2.1 Cadeias de Markov

Suponha um sistema S que possui m estados possíveis, denotados por $x \in \{1,2,..,m\}$, essse sistema evolui com o passar da variável $t = 1,2,\ldots$, em geral é chamada de tempo. E denomina-se $P_{i,j}$, a probabilidade de transição do estado i para o estado j. Pode-se definir a matriz de transição P como $P = (P_{i,j})_{m\times m}$, para ficar mais claro pode-se utilizar um $m = 3$, assim a matriz de transição ficaria:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix}$$

Se as probabilidades de transição $P_{i,j}$, $i = 1,2,..,m$ e $j = 1,2,..,m$ forem dependentes unicamente do estado atual i do sistema denomina-se esse sistema de uma cadeia de Markov. Assim, dado que o sistema está em um estado i é possível determinar a probabilidade de ele mudar para um certo estado j. A ideia das cadeias de Markov é bastante utilizada na área de *Machine Learning*.

2.2 Algoritmo Metropolis-Hastings

Com essa ideia de cadeias de Markov pode-se implementar um algoritmo que tem como objetivo gerar variáveis aleatórias x_i que seguem a distribuição g. A ideia por trás desse algoritmo é semelhante ao método de Aceitação e Rejeição que também tem o intuito de gerar variáveis aleatórias que seguem um certa distribuição g. Para tal é necessário uma função que gera variáveis aleatórias que seguem uma distribuição Q. Abaixo mostra-se o esquema do algoritmo:

Inicializa x_0; e define $t = 0$
para $t = 1,2,\ldots,n$ **faça**
  $y \sim Q(.|x_t)$ ▷ *Gera y seguindo a distribução Q*
  $u \sim U(0,1)$ ▷ *Gera u seguindo uma distribução uniforme*
  se $u \leq \alpha(x_t,y)$ **então** ▷ *Condição para aceitação*
    $x_{t+1} = y$
  senão
    $x_{t+1} = x_t$ ▷ *Se rejeitado mantém* x_t
  fim se
fim para

Define-se $\alpha(X_t,Y)$, a taxa de aceitação de Metropolis, da seguinte forma:

$$\alpha(X_t,Y) = \min\left(1, \frac{g(Y)Q(X_t|Y)}{g(X_t)Q(Y|X_t)}\right) \tag{1}$$

Contudo, como nesse caso será utilizado a normal multivariada para a função Q, então a expressão (1) pode ser reduziada a seguinte fórmula:

$$\alpha(X_t,Y) = \min\left(1, \frac{g(Y))}{g(X_t)}\right) \tag{2}$$

3 Implementação

Nessa seção discutiremos apenas a implementação do método de Metrópolis-Hastings para a geração de amostras com distribuição de Dirichlet, já que as outras partes referentes ao funcionamento do programa já foram explicados no relatório do EP4, mudando apenas essa parte.

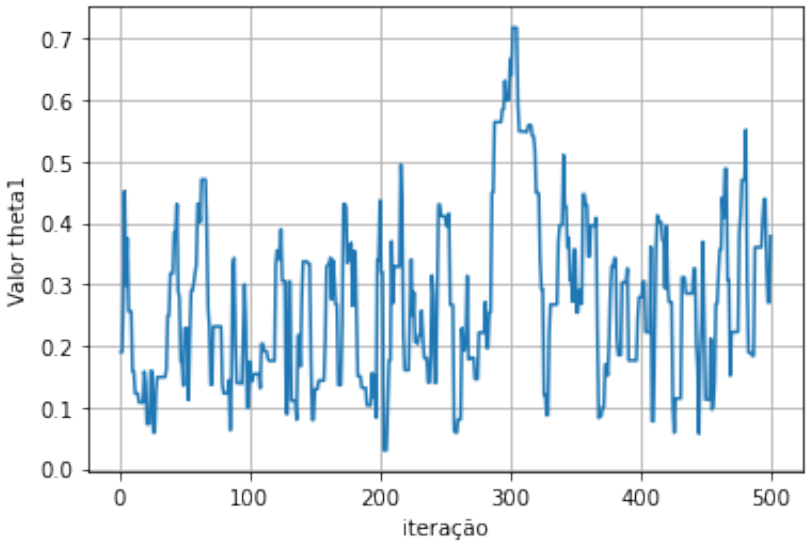
A função responsável pela geração do vetor θ se chama `thetamcmc(n,x,y)`. Após ter recebido como parâmetros os vetores x e y e o número de valores a serem gerados, a função se utiliza de x e y para calcular as variâncias e covariâncias da Dirichlet para por na matriz de covariância, dadas pelas fórmulas:

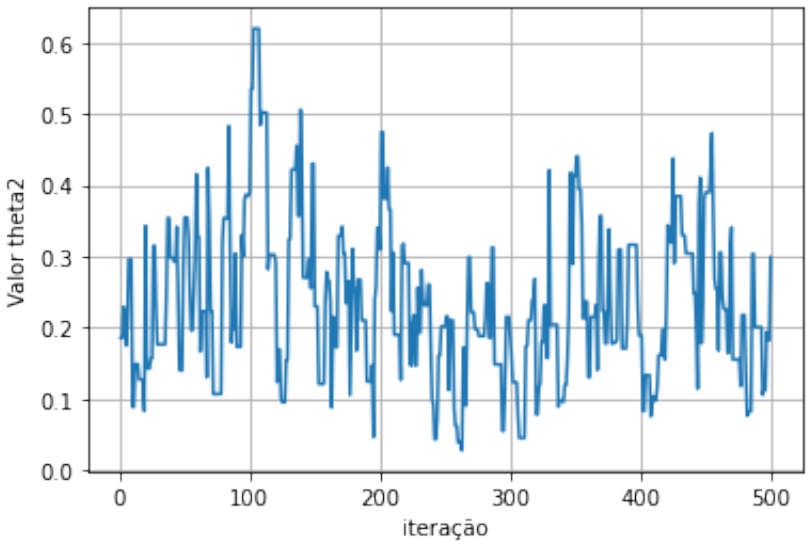
$$Var[X_i|\alpha] = \frac{\alpha_i(\alpha_0 - \alpha_i)}{\alpha_0^2(\alpha_0 + 1)}$$

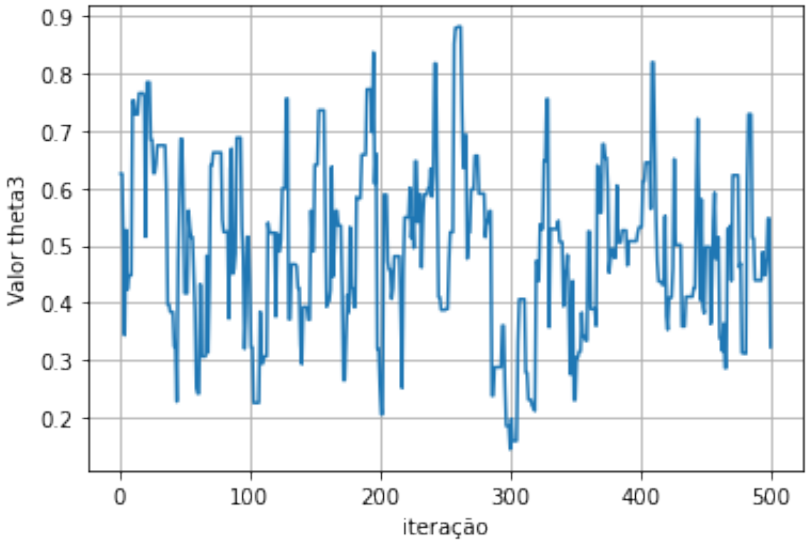
$$Cov[X_i X_j|\alpha] = \frac{-\alpha_i \alpha_j)}{\alpha_0^2(\alpha_0 + 1)}$$

Sendo α igual a soma de x e y, α_i o elemento i da soma e α_0 a soma dos elementos de α.

Começamos com um vetor θ fixo qualquer, escolhido sem rigor. Depois definimos o *burn in* como 200, a partir de inspeção visual. Veja os gráficos gerados para os elementos do vetor θ:







Vemos que é seguro dizer que um *burn in* de 200 é o suficiente para o algoritmo "se esquecer"do vetor inicial, nos três valores do vetor.

Então criamos uma matriz (n+200) por 3, inicialmente preenchida por zeros, mas que depois serão substituídos pelas amostras aceitas pelo algoritmo.

Finalmente damos início a um loop `for` para gerar as amostras. Consideramos o vetor θ antigo como a média da normal multivariada e a matriz de covariâncias como variância constante. A partir disso geramos o novo vetor θ.

Sendo $\theta = [\theta_1, \theta_2, \theta_3]$ calculamos $\theta_3 = 1$ - $(\theta_1 + \theta_2)$. Após termos calculado o novo vetor, checamos se ele será aceito na amostra. Com ajuda da função `g(theta, params)`, que retorna -1 caso algum dos elementos do vetor for negativo (pode ocorrer por causa da normal multivariada) e retorna o valor calculado usando a função de probabilidade da Dirichlet caso contrário, calculamos *alpha*, o mínimo entre 1 e $\frac{g(x_j)}{g(x_i)}$, sendo x_j o vetor θ novo e x_i o vetor θ antigo. Geramos um valor u uniformemente distribuído entre 0 e 1, se u for menor que *alpha* aceitamos o novo vetor, que se torna o vetor antigo e é incluído na amostra, caso contrário, repetimos o vetor antigo na amostra.

No final do processo devolvemos n vetores, pois descontamos aqueles do *burn in*.

O resto do código se mantém igual, apenas trocando o gerador de vetores com distribuição dirichlet da biblioteca `numpy` pelo gerador de Markov Chain Monte Carlo descrito acima. As outras funções são explicadas no relatório do EP4.

4 Conclusão

Nessa seção discute-se os resultados obtidos pelo código. Após a análise da função, concluímos que o tempo máximo para o retorno do resultado é cerca de 30 minutos, mas pode ser menos dependendo do número de pontos necessários para calcular o valor da integral com um erro menor que 0.0005.

Além disso, comparando os resultados obtidos com esse programa e com o programa anterior, chegamos a valores bem próximos. Com isso acreditamos que a função gera estimativas boas para o valor da função $W(v)$.