Uma implementação eficiente em Python de um algoritmo de separação de fontes sonoras

Natanael Luciano de Matos
Universidade Federal do Rio de Janeiro (UFRJ)
natanael@dme.ufrj.br

Hugo Tremonte de Carvalho
Universidade Federal do Rio de Janeiro (UFRJ)
hugo@dme.ufrj.br

Resumo: Aqui apresentamos uma implementação eficiente em Python de um algoritmo já previamente proposto de separação às cegas de sinais. Separação de sinais consiste em obter os elementos que o geram. Para estruturação do algoritmo foi utilizada uma abordagem Bayesiana em conjunto com métodos Markov Chain Monte Carlo (MCMC). Resultados são apresentados utilizando os áudios retornados pelo programa implementado, estes mostram a capacidade do modelo de separar os sinais, além da robustez da matriz de mistura por conta dos métodos MCMC utilizados.
Palavras-chave: Inferência Bayesiana. Markov Chain Monte Carlo. Separação de Fontes Sonoras. Compiladores *just-in-time.* Transformada Discreta do Cosseno.

An Efficient Python Implementation of a Blind Source Separation Algorithm

Abstract: Here we present an efficient implementation, in Python programming language, of a previously proposed algorithm of blind source separation. Source separation consists of obtaining the elements that additively compose the observed signal. For structuring the algorithm, a Bayesian approach was employed together with Markov Chain Monte Carlo (MCMC) methods. Results are presented using the audios returned by the implemented program, showing the model's ability to separate the signals, in addition to the robustness of the mixing matrix due to the MCMC methods used.
Keywords: Bayesian Inference. Markov Chain Monte Carlo. Sound Source Separation. Just-in-time compilators. Discrete Cosine Transform.

1. Introdução

Separação de fontes sonoras consiste em decompor um sinal de áudio em componentes mais fundamentais. Existem duas principais estruturas para algoritmos de separação, sendo essas: separação às cegas, onde não se tem qualquer informação a priori sobre os elementos que compõem o sinal; e separação informada, onde além do sinal, é conhecida alguma informação de sua estrutura subjacente. Dentre estas, existem algumas técnicas tais como Inferência Bayesiana, Fatoração de Matrizes Não-negativas (MÜLLER, 2015, Cap. 8), dentre outras. Alguns dos usos mais comuns dos métodos de separação de áudio são: Análise forense, aplicativos de identificação de áudio, separação de áudio estéreo e detecção de sinais.

Neste trabalho, trabalharemos com separação às cegas utilizando uma modelagem Bayesiana. Além disso, daremos foco para os chamados casos subdeterminados, onde o número de réplicas do sinal m é menor que o número de fontes n.

Realizamos aqui uma implementação eficiente, na linguagem Python, do algoritmo proposto em (FEVOTTE; GODSILL, 2006). Para isso, será utilizada uma coleção de amostras no tempo que serão misturadas e então submetidas ao algoritmo. Após processo de separação, a eficiência do mesmo será avaliada apenas perceptiva e qualitativamente.

2. Fundamentação teórica

O objetivo de um algoritmo de separação de fontes é processar um ou mais sinais de áudio de forma a se obter suas componentes significativas. Note que este é naturalmente um problema mal posto, visto que a definição de "componentes significativas" é ambígua: em um quarteto de cordas podemos pensar que tais unidades fundamentais são o som gerado por cada instrumento ou grupo de instrumentos, o som gerado por cada corda, dentre outras possibilidades. Neste trabalho não iremos tratar de tais questões, mas serão impostas condições estatísticas nos sinais analisados, de modo a garantir certa independência no que nos será retornado.

Recordando que nosso objetivo é realizar uma implementação eficiente, na linguagem Python, do algoritmo proposto em (FEVOTTE; GODSILL, 2006), façamos primeiramente uma revisão do modelo bayesiano nele apresentado.

2.1 Inferência Bayesiana

No escopo da Inferência Estatística, a abordagem Bayesiana, além de analisar os dados à luz de algum modelo probabilístico, incorpora conhecimento prévio sobre os parâmetros que se deseja estimar. Mais precisamente, seja $\boldsymbol{z} \in R^n$ o conjunto de observações e $\boldsymbol{\theta} \in R^d$ os parâmetros, de modo que $p(\boldsymbol{z} \mid \boldsymbol{\theta})$ representa a verossimilhança; denote por $p(\boldsymbol{\theta})$ a distribuição *a priori*, que codifica o conhecimento prévio acima mencionado. Pelo Teorema de Bayes, temos que

$$p(\boldsymbol{\theta} \mid \boldsymbol{z}) = \frac{p(\boldsymbol{z} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{z})} \propto p(\boldsymbol{z} \mid \boldsymbol{\theta})p(\boldsymbol{\theta}),$$

onde $p(\boldsymbol{\theta}|\boldsymbol{z})$ é a distribuição *a posteriori*, condicionada nos dados observados.

A análise da distribuição *a posteriori* geralmente é muito difícil ou até mesmo impossível de se realizar analiticamente. Sendo assim é necessária a utilização de métodos computacionais tais como Markov Chain Monte Carlo (MCMC) para realizar tal procedimento.

Métodos MCMC consistem em construir uma cadeia de Markov cuja distribuição estacionária coincida com a distribuição de interesse (nesse caso, a *posteriori*), de modo que amostrar de tal processo estocástico, à longo prazo, irá retornar amostras suficientemente próximas da distribuição alvo.

Alguns dos algoritmos de MCMC mais utilizados são o Amostrador de Gibbs (RORBERT; CASELLA, 2010, Cap. 7) e o algoritmo de Metropolis-Hastings (RORBERT; CASELLA, 2010, Cap. 6) descritos a seguir.

2.2 Amostrador de Gibbs

Assuma que haja uma partição do vetor paramétrico $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_k\}$ de modo que as distribuições condicionais abaixo sejam conhecidas e fáceis de amostrar:

$$p(\boldsymbol{\theta}_i | \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_{i+1}, \dots, \boldsymbol{\theta}_k, \boldsymbol{z}),$$

para $i = 1, \dots, k$. Nesse caso, pode-se mostrar que o procedimento no Algoritmo 1 abaixo gera uma cadeia de Markov com as propriedades desejadas. A quantidade denotada por "*sobra*" representa o número de passos gastos antes que a cadeia de Markov alcance a distribuição estacionária.

Algoritmo 1: Pseudo-código do amostrador de Gibbs.

ENTRADA	Distribuições condicionais completas dos parâmetros $p(\theta_i \mid \theta_1, \dots, \theta_{i-1}, \theta_{i+1}, \dots, \theta_k)\ \forall i \in \{1, \dots, k\}$, um conjunto de dados $\boldsymbol{z}$, e o número de iterações K.
SAÍDA	Amostras da distribuição estacionária da cadeia
INICIALIZE	Parâmetros da distribuição $\boldsymbol{\theta}^{(0)} = \{\boldsymbol{\theta}_1^{(0)}, \dots, \boldsymbol{\theta}_k^{(0)}\}$
	PARA $i \in \{1, \dots, K + sobra\}$ FAÇA
	PARA $j \in \{1, \dots, k\}$ FAÇA
	ATUALIZE $\theta_j^{(i)} \sim p\left(\theta_j \mid \theta_1^{(i)}, \theta_2^{(i)}, \dots, \theta_{j-1}^{(i)}, \theta_{j+1}^{(i-1)}, \dots, \theta_k^{(i-1)}, \boldsymbol{z}\right)$
RETORNE	Vetor de parâmetros $\boldsymbol{\theta}^{(m+sobra)} = (\boldsymbol{\theta}_1^{(m+sobra)}, \boldsymbol{\theta}_2^{(m+sobra)}, \dots, \boldsymbol{\theta}_k^{(m+sobra)})$ para $m = 1, \dots, K$.

2.3 Algoritmo de Metropolis-Hastings

Seja $p(\boldsymbol{\theta})$ a distribuição da qual se deseja obter amostras, $Q(\boldsymbol{\theta}|\boldsymbol{y})$ uma função densidade de probabilidade qualquer, e $f(\boldsymbol{\theta})$ uma função tal que:

$$p(\boldsymbol{\theta}) \propto f(\boldsymbol{\theta}).$$

Então o procedimento descrito no Algoritmo 2 cria uma cadeia de Markov cuja distribuição estacionária é $p(\boldsymbol{\theta})$.

Algoritmo 2: Pseudo-código do Algoritmo de Metropolis-Hastings

ENTRADA — Distribuição de interesse $p(\boldsymbol{\theta})$, função $f(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta})$, e $Q(\boldsymbol{\theta}|\boldsymbol{y})$ uma função densidade de probabilidade qualquer.

SAÍDA — Amostras da distribuição de interesse.

INICIALIZE — Escolha um $\boldsymbol{y} = \boldsymbol{\theta}^{(0)}$ qualquer.

PARA $i \in \{1, \dots, K + sobra\}$ FAÇA

AMOSTRE

$$\boldsymbol{\theta}' \sim Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(i-1)})$$
$$u \sim uniforme(0,1)$$

CALCULE

$$\alpha = min\left\{1, \frac{f(\boldsymbol{\theta}')}{f(\boldsymbol{\theta}^{(i-1)})} \frac{Q(\boldsymbol{\theta}^{(i-1)}|\boldsymbol{\theta}')}{Q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{(i-1)})}\right\}$$

SE $u \leqslant \alpha$ FAÇA $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}'$

SE $\alpha < u$ FAÇA $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)}$

RETORNE — Vetor de parâmetros $\boldsymbol{\theta}^{(m+sobra)}$ para $m = 1, \dots, K$.

2.4 Modelo

Nesta seção será exposto o modelo proposto em (FEVOTTE; GODSILL, 2006). Está sendo suposto que as observações seguem um modelo linear instantâneo, ou seja, os sinais observados no tempo t são combinações lineares ruidosas das fontes no mesmo tempo t de modo que para $t = 1, \dots, N$ tem–se:

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{s}_t + \boldsymbol{n}_t,$$

onde $\boldsymbol{x}_t = \left[x_{1,t,\dots,} x_{n,t}\right]^T$ é o vetor contendo as observações dos sinais no tempo t, $\mathbf{A}$ é a matriz de mistura, $\boldsymbol{s}_t = \left[s_{1,t,\dots,} s_{n,t}\right]^T$ é o vetor dos coeficientes da fonte no tempo t, e $\boldsymbol{n}_t = \left[n_{1,t,\dots,} n_{n,t}\right]^T$ é o vetor de ruídos. De forma matricial tem–se:

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{n}.$$

Sendo assim, o objetivo do algoritmo é estimar os coeficientes das fontes $\boldsymbol{s}$, e a matriz de mistura $\boldsymbol{A}$, a partir das observações no vetor $\boldsymbol{x}$.

2.4.1 Hipóteses sobre esparsidade

Nesta seção, serão discutidas as suposições feitas sobre as fontes e os ruídos. Será utilizada uma base onde as observações e os coeficientes das fontes são representados de forma esparsa, ou seja, apenas uma pequena proporção dos seus coeficientes é diferente de zero. Sendo assim, precisamos de uma transformação que leve o sinal do domínio do tempo para um que o represente de tal forma.

Seja $\boldsymbol{\Phi}_{n\times n}$ uma matriz invertível que define uma transformação para a base desejada. Então é possível escrever o vetor das observações, $\boldsymbol{x}$, como:

$$\begin{aligned}\tilde{\boldsymbol{x}} &= \boldsymbol{x\Phi} \\ &= (\boldsymbol{As} + \boldsymbol{n})\boldsymbol{\Phi} \\ &= \boldsymbol{A\tilde{s}} + \tilde{\boldsymbol{n}}.\end{aligned}$$

Dessa forma, como $\boldsymbol{\Phi}$ é uma matriz de mudança de base, calcular os coeficientes das fontes nessa nova base, onde os sinais são esparsos, é equivalente a calculá-los no domínio do tempo. Por conta disso, todo o processo de estimação de parâmetros é feito no domínio da transformada.

Um dos motivos de se realizar essa mudança de base é que toda a informação mais significativa do sinal será comprimida em algumas poucas entradas, e também não haverá perda de informação do sinal original, fazendo com que os algoritmos de separação possam trabalhar com menos dados, assim diminuindo a chance de erro numérico bem como o custo computacional envolvido.

Tome por exemplo dois vetores, um vetor contendo milhares de entradas onde precisamos realizar diversas operações com pontos flutuantes para cada entrada, e outro vetor esparso que de alguma representa, sem perda, toda a informação do primeiro. Dessa forma fica claro que a chance de erro numérico é menor ao utilizar o vetor esparso pelo simples fato de ter menos entradas relevantes sobre as quais iremos realizar as operações, que também serão em menor número.

2.4.2 Transformada Discreta Cosseno (DCT)

A DCT (AHMED; NATARAJAN; RAO, 1974) será escolhida para realizar a transformação para uma base onde sinais musicais são representados de forma esparsa[1]. Essencialmente, a DCT expressa uma sequência finita de pontos/dados como uma soma de funções cosseno com diferentes frequências. A DCT é utilizada em processamento de sinais e compressão de arquivos, um dos seus usos mais correntes sendo no formato JPEG. A transformação de um sinal pela DCT é dada pela seguinte equação:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right], k = 0, \dots, N-1,$$

com N sendo o tamanho do sinal e X_k a k-ésima componente da DCT. É possível mostrar que a DCT é uma mudança de base ortogonal.

[1] O trabalho tomado como referência utiliza uma modificação da DCT, conhecida na literatura por MDCT, amplamente empregada em algoritmos de compressão de áudio. Aqui, optamos por utilizar a DCT por ter uma implementação mais simples.

2.4.2 Modelando esparsidade estatisticamente

Assumimos que os coeficientes das fontes, no domínio da DCT $\tilde{\boldsymbol{s}}$, dados os parâmetros α_i e λ_i são i.i.d. com distribuição t de Student com α_i graus de liberdade e λ_i um parâmetro de escala, denotada por $t(\alpha_i, \lambda_i)$, definida como:

$$p(\tilde{s}_{i,k} \mid \alpha_i, \lambda_i) = \frac{\Gamma(\frac{\alpha_i + 1}{2})}{\lambda_i \sqrt{\alpha_i \pi} \Gamma(\frac{\alpha_i}{2})} (1 + \frac{1}{\alpha_i} (\frac{\tilde{s}_{i,k}}{\lambda_i})^2)^{-(\boldsymbol{\alpha_i + 1})/2}.$$

É um resultado conhecido (CASELLA; BERGER, 2001) que uma distribuição t de Student pode ser expressa como uma combinação de distribuições normais e inversas gama, dada da seguinte forma:

$$p(\tilde{s}_{i,k} \mid \alpha_i, \lambda_i) = \int_0^{+\infty} \mathcal{N}(\tilde{s}_{i,k} \mid 0, v_{i,k}) \mathcal{IG}(v_{i,k} \mid \frac{\alpha_i}{2}, \frac{2}{\alpha_i \lambda_i^2}) dv_{i,k},$$

onde $\mathcal{N}(x \mid \mu, v)$ e $\mathcal{IG}(x \mid \gamma, \beta)$ são densidades das distribuições Normal e Inversa-Gama respectivamente. Claro que para fazer essa nova representação é preciso pagar um preço, que é a inserção de parâmetros auxiliares, $v_{i,k}$.

Ressaltamos que a hipótese de que os coeficientes $\tilde{\boldsymbol{s}}$ são independentes é apenas uma hipótese facilitadora, entretanto isso não significa que o algoritmo não será capaz de separar sinais cujas fontes são correlacionadas.

2.4.3 Propriedades do ruído

Estamos assumindo que $\tilde{\boldsymbol{n}}$ é um ruído Gaussiano com matriz de covariância $\sigma^2 I_m$, sendo σ^2 desconhecido e I_m denota a matriz identidade de tamanho $m \times m$. Note que se $\boldsymbol{\Phi}$ é uma matriz ortogonal (ou seja, $\boldsymbol{\Phi} = \boldsymbol{\Phi}^T$), então o vetor de ruídos $\boldsymbol{n}$ terá a mesma distribuição de $\tilde{\boldsymbol{n}}$. Este é o caso da DCT, conforme mencionado acima.

2. Distribuição dos parâmetros

Utilizando as equações da Seção 2.4, obtêm-se as distribuições condicionais completas dos parâmetros do modelo. Aqui as contas foram omitidas, mas podem ser encontradas em (FEVOTTE; GODSILL, 2006). Defina os seguintes vetores e matrizes:

$$\boldsymbol{v}_k = [v_{1,k}, \ldots, v_{n,k}], \ \boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$$
$$\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n], \ \boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]$$
$$\boldsymbol{\theta} = [\tilde{\boldsymbol{s}}, \boldsymbol{A}, \sigma, \boldsymbol{v}, \boldsymbol{\alpha}, \boldsymbol{\lambda}].$$

Daqui em diante denotamos por $\boldsymbol{\theta}_{-k}$ o $\boldsymbol{\theta}$ vetor sem a componente de índice ou nome k. Então a distribuição *a posteriori* é dada da seguinte forma:

$$p(\boldsymbol{\theta} | \tilde{\boldsymbol{x}}) = p(\tilde{\boldsymbol{x}} | \boldsymbol{\theta}) p(\tilde{\boldsymbol{s}} \mid \boldsymbol{v}) p(\boldsymbol{v} \mid \boldsymbol{\alpha}, \boldsymbol{\lambda}) p(\boldsymbol{\alpha}) p(\boldsymbol{\lambda}) p(\boldsymbol{A}) p(\sigma).$$

Estamos assumindo a priori de Jeffreys para $\boldsymbol{\lambda}$ e σ, e uma priori uniforme para $\boldsymbol{A}$ e $\boldsymbol{\alpha}$. Listamos abaixo as distribuições condicionais completas de cada bloco de $\boldsymbol{\theta}$, bem como explicitamos a verossimilhança do modelo.

Verossimilhança:

$$p(\tilde{\mathbf{x}} \mid \boldsymbol{\theta}) = p(\tilde{\mathbf{x}} \mid \mathbf{A}, \tilde{\mathbf{s}}, \sigma)$$
$$= \prod_{k=1}^{N} \mathcal{N}(\tilde{\mathbf{x}}_k \mid \mathbf{A}\tilde{\mathbf{s}}_k, \sigma^2 \mathbf{I}_m)$$

- Condicional completa de $\tilde{\boldsymbol{s}}$:

$$p(\tilde{\boldsymbol{s}} \mid \boldsymbol{\theta}_{-\tilde{s}}, \tilde{\boldsymbol{x}}) = \prod_{k=1}^{N} \mathcal{N}(\tilde{\boldsymbol{s}}_k \mid \boldsymbol{\mu}_{s_k}, \boldsymbol{\Sigma}_{s_k})$$
$$\Sigma_{s_k} = ((1/\sigma^2)\boldsymbol{A}^T\boldsymbol{A} + \mathrm{diag}(\boldsymbol{v}_k)^{-1})^{-1}$$
$$\boldsymbol{\mu}_{\bar{s}_k} = (1/\sigma^2)\boldsymbol{\Sigma}_{\bar{s}_k}\boldsymbol{A}^T\tilde{\boldsymbol{x}}_k$$

- Condicional completa de σ^2:

$$(\sigma^2 \mid \boldsymbol{\theta}_{-(\boldsymbol{A},\sigma)}, \boldsymbol{x}) \sim \mathcal{IG}(\alpha_\sigma, \beta_\sigma)$$
$$\alpha_\sigma = \frac{(N-n)m}{2} \text{ e}$$

$$\beta_\sigma = \frac{2}{\sum_{i=1}^{m}\sum_{k=1}^{N}\tilde{\boldsymbol{x}}_{i,k}{}^2 - (\sum_{k=1}^{N}\tilde{\boldsymbol{x}}_{i,k})(\sum_{k=1}^{N}\tilde{\boldsymbol{s}}_k\tilde{\boldsymbol{s}}_k^T)^{-1}(\sum_{k=1}^{N}\tilde{\boldsymbol{x}}_{i,k}\tilde{\boldsymbol{s}}_k)}$$

- Condicional completa de $\boldsymbol{A}$:

$$p(\boldsymbol{A} \mid \boldsymbol{\theta}_{-\boldsymbol{A}}, \tilde{\boldsymbol{x}}) = \prod_{i=1}^{m} \mathcal{N}(\boldsymbol{r}_i \mid \boldsymbol{\mu}_{\boldsymbol{r}_i}, \boldsymbol{\Sigma}_{\boldsymbol{r}})$$
$$\boldsymbol{\Sigma}_{\boldsymbol{r}} = \sigma^2(\boldsymbol{s}_k\tilde{\boldsymbol{s}}_k^T)^{-1} \text{e } \mu_{r_i} = (1/\sigma^2)\Sigma_{\mathrm{r}}\tilde{x}_{i,k}\tilde{\boldsymbol{s}}_k$$

- Condicional completa de $\boldsymbol{v}$:

$$p(\boldsymbol{v} \mid \boldsymbol{\theta}_{-v}, \tilde{\boldsymbol{x}}) = \prod_{k=1}^{N}\prod_{i=1}^{n} \mathcal{IG}(v_{i,k} \mid \gamma_{v_i}, \beta_{v_{i,k}})$$

$$\text{com } \gamma_{v_i} = (\alpha_i + 1)/2 \text{ e } \beta_{v_{i,k}} = 2/(\tilde{s}_{i,k}^2 + \alpha_i\lambda_i^2)$$

- Condicional completa de $\boldsymbol{\alpha}$:

$$\boldsymbol{p}(\boldsymbol{\alpha} \mid \boldsymbol{\theta}_{-\boldsymbol{\alpha}}, \tilde{\boldsymbol{x}}) \propto$$
$$\prod_{\boldsymbol{i=1}}^{\boldsymbol{n}} \frac{P_i^{-(\boldsymbol{\alpha_i}/2+1)}}{\Gamma(\frac{\alpha_i}{2})^N}(\frac{\alpha_i\lambda_i^2}{2})^{\alpha_i N/2}\exp(-\frac{\alpha_i\lambda_i^2}{2}R_i)p(\alpha_i)$$

$$\text{com } R_i = \sum_{\boldsymbol{k=1}}^{\boldsymbol{N}} \boldsymbol{1/v_{i,k}} \text{ e } P_i = \prod_{\boldsymbol{k=1}}^{\boldsymbol{N}} \boldsymbol{v_{i,k}}.$$

- Condicional completa de λ_i^2:

$$\left(\lambda_i^2 \mid \theta_{-\lambda}, \tilde{\boldsymbol{x}}\right) \sim \mathcal{G}\left(\gamma_{\lambda_i}, \beta_{\lambda_i}\right)$$
$$\text{com } \gamma_{\lambda_i} = (\alpha_i N)/2 \text{ e } \beta_{\lambda_i} = 2/(\alpha_i R_i).$$

4. O algoritmo de separação

Como foi falado na Seção 2.2, para utilizar o Amostrador de Gibbs é preciso conhecer todas as distribuições condicionais completas dos parâmetros, entretanto é possível contornar essa limitação caso haja uma exceção. Isso será feito utilizando o algoritmo de Metropolis-Hastings para obter uma amostra da distribuição desconhecida, e essa será utilizada para atualizar o valor do parâmetro dentro do Amostrador de Gibbs. O Algoritmo 3 descreve como esse processo é realizado para o problema de separação de fontes descrito nesse trabalho.

Algoritmo 3: Pseudo-código do Algoritmo de Separação de fontes

ENTRADA	Distribuições condicionais completas dos parâmetros $p(\theta_i \mid \theta_1, \dots, \theta_{i-1}, \theta_{i+1}, \dots, \theta_k)\ \forall i \in \{1, \dots, k\}$, um conjunto de dados $\boldsymbol{z}$, e o número de iterações K.
	Distribuição de interesse $p(\boldsymbol{\theta})$, função $f(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta})$, e $Q(\boldsymbol{\theta}\mid\boldsymbol{y})$ uma função densidade de probabilidade qualquer.
	Vetor de parâmetros $\boldsymbol{\theta} = [\tilde{\boldsymbol{s}}, \boldsymbol{A}, \sigma, \boldsymbol{v}, \boldsymbol{\alpha}, \boldsymbol{\lambda}]$
SAÍDA	Matriz de mistura $\boldsymbol{A}$ e vetor $\tilde{\boldsymbol{s}}$ dos coeficientes das fontes.
INICIALIZE	Parâmetros da distribuição $\boldsymbol{\theta}^{(0)} = \{\tilde{\boldsymbol{s}}^{(0)}, \dots, \boldsymbol{\lambda}^{(0)}\}$.
ENQUANTO	O amostrador não tiver convergido
	PARA $j \in \{1, \dots, K + sobra\}$ FAÇA:
	ATUALIZE $\theta_j^{(i)} \sim p\left(\theta_j \mid \theta_1^{(i)}, \theta_2^{(i)}, \dots, \theta_{j-1}^{(i)}, \theta_{j+1}^{(i-1)}, \dots, \theta_k^{(i-1)}, \boldsymbol{z}\right)$
	$A^{(i)} \sim p(A \mid \tilde{s}^{(i-1)}, \sigma^{2^{(i-1)}}, V^{(i-1)}, \lambda^{(i-1)}, \alpha^{(i-1)})$
	$\tilde{s}^{(i)} \sim p(\tilde{s} \mid, A^{(i)}, \sigma^{2^{(i-1)}}, V^{(i-1)}, \lambda^{(i-1)}, \alpha^{(i-1)})$
	$\sigma^{2^{(i)}} \sim p(\sigma^2 \mid, A^{(i)}, \tilde{s}^{(i)}, V^{(i-1)}, \lambda^{(i-1)}, \alpha^{(i-1)})$
	$V^{(i)} \sim p(V \mid A^{(i)}, \tilde{s}^{(i)}, \sigma^{2^{(i)}}, \lambda^{(i-1)}, \alpha^{(i-1)})$
	$\lambda^{(i)} \sim p(\lambda^{(i)} \mid A^{(i)}, \tilde{s}^{(i)}, \sigma^{2^{(i)}}, V^{(i)}, \alpha^{(i-1)})$
	Use MH para amostrar $\alpha' \sim p(\alpha \mid \boldsymbol{\theta}_{-\alpha}, \tilde{\boldsymbol{x}})$
	Atualize $\alpha^{(i)} = \alpha'$
RETORNE	Matriz de mistura $\boldsymbol{A}$ e vetor $\tilde{\boldsymbol{s}}$ de coeficientes das fontes.

4.1 Numba

Como o objetivo deste trabalho é uma implementação eficiente, em *Python*, do algoritmo proposto em (FEVOTTE; GODSILL, 2006), não é recomendado utilizar *listas* para construir os vetores dos parâmetros, mesmo que sua implementação seja mais fácil. Isso se dá porque listas em *Python* são guardadas de forma não sequencial na memória, ou seja, sempre que precisamos acessar um dado na lista muito tempo é gasto para o computador alcançá-lo, causando um grande impacto na eficiência do código.

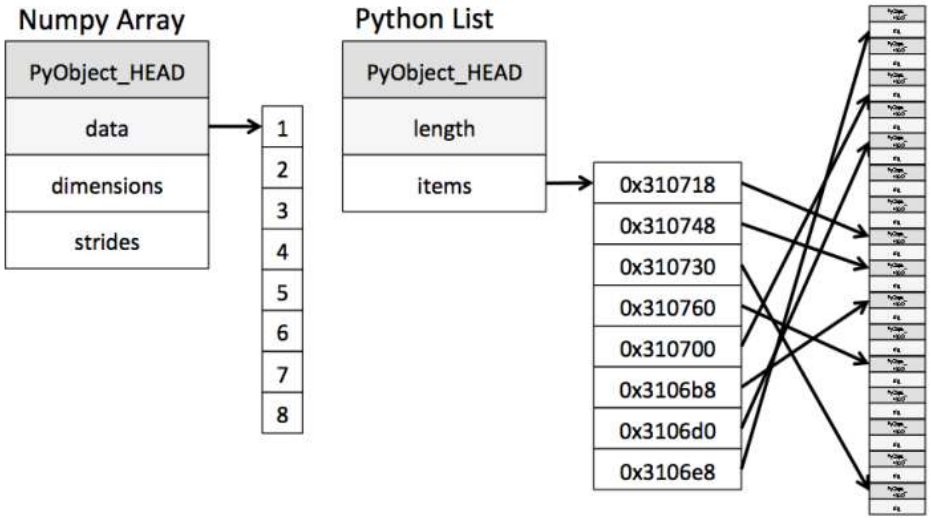

Figura 11: Comparação entre um array do Numpy e uma lista em Python.
(fonte: jakevdp.github.io/images/array_vs_list.png)

Para contornar essa limitação, foram utilizados os vetores da biblioteca *Numpy*, esses sendo armazenados de forma sequencial na memória, e para otimizar os laços presentes nos métodos MCMC, utilizou-se a biblioteca *Numba*.

O *Numba* é uma biblioteca que traduz funções em *Python* para código de máquina fazendo com que as funções traduzidas rodem em velocidades comparáveis a C e Fortran. Tais ferramentas são conhecidas como compiladores *just-in-time* (JIT).

5. Resultados

Inicialmente foi realizada uma tradução do código disponibilizado em (FEVOTTE; GODSILL, 2006), da linguagem Matlab/Octave para *Python*. Entretanto, ao analisar o código-fonte original, verificou-se algumas discrepâncias entre as distribuições dos parâmetros na implementação do algoritmo e o modelo teórico proposto. Por conta disso, a presente implementação em *Python* foi feita somente a partir do referencial teórico de acordo com as distribuições descritas na Seção 2.4. Tais códigos estão disponíveis em (MATOS, 2021).

5.1 Experimentos

Como critério de convergência para o algoritmo de separação, estaremos utilizando a seguinte métrica:

$$\left| max_{i,j}\left(\boldsymbol{A}^{(k)} - \boldsymbol{A}^{(k-1)}\right)_{i,j} \right| < \varepsilon,$$

para algum ε suficientemente pequeno.

Aqui não verificaremos rigorosamente se a cadeia de Markov convergiu de fato para a sua distribuição estacionária, uma vez que esta análise é complexa e não é

o foco deste trabalho. Tendo isso em mente, a análise da eficiência/convergência do algoritmo de separação foi feita de forma empírica ao analisar perceptivamente cada saída do programa.

Foram realizados, ao todo, dez experimentos para verificar a eficácia do algoritmo, sendo eles:

- Experimentos 1-4:

Foi recriado o experimento *"Robustness to Mixing Matrix"* apresentado em (FEVOTTE; GODSILL, 2006). Esse consiste em analisar a capacidade de separação do algoritmo para matrizes de mistura com a seguinte estrutura:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 & 1 \\ \tan\psi_1 & \tan\psi_2 & \tan\psi_3 \end{bmatrix}.$$

A razão de fazer isso é que testaremos a capacidade do algoritmo de separar amostras que são obtidas por receptores colocados de forma circular ao redor da fonte sonora. Lembrando que o número de amostras do sinal, e o número de fontes que foram passados como parâmetro ao algoritmo são, respectivamente, dois e três. Além disso, na criação das observações foi introduzido ruído gaussiano aditivo com desvio padrão $\sigma = 0{,}01$. Para tal cenário, utilizaram-se as seguintes sequências de valores para os ângulos $\psi_1, \psi_2, e\ \psi_3$:

Tabela 1: Valores dos ângulos utilizados nos experimentos 1-4.

ψ_1	ψ_2	ψ_3
-45º	15º	75º
-45º	66º	75º
-45º	-36º	75º
57º	66º	75º

Após analisarmos cada áudio retornado pelo programa, é seguro afirmar a capacidade do mesmo de lidar com essa situação. Os áudios utilizados para gerar as amostras, bem como as fontes retornadas pelo programa se encontram em (MATOS, 2021).

- Experimentos 5-7:

O foco destes experimentos é determinar se o algoritmo é capaz de separar sinais dependentes, uma vez que o modelo assume independência dos sinais das fontes. Para isso foram utilizadas três versões da quinta sinfonia de Beethoven, uma em MIDI, uma regida por Karajan, e a outra por Bernstein. Além disso, também é verificado se o programa tem capacidade de separar sinais dependentes ao aumentarmos o número de observações passadas ao modelo, tendo um sistema super-determinado ($m \geq n$).

Foi uma surpresa observar que mesmo para sistema subdeterminado ($m < n$) o algoritmo foi capaz de realizar a separação. Os áudios utilizados e obtidos nestes experimentos se encontram em (MATOS, 2021).

- Experimento 8-10:

O objetivo desse experimento é analisar a capacidade do algoritmo de lidar com dados contendo diferentes intensidades de ruídos. Para isso utilizamos o mesmo conjunto de áudio dos experimentos 1-4 para gerar observações contendo ruídos com desvios-padrão $\sigma = 0.01, 0.1, 0.5$ e sistema determinado ($m = n$).

Para os dois primeiros valores de σ, o algoritmo foi capaz de estimar satisfatoriamente as fontes que formam o sinal, mas para o último valor, o programa foi capaz obter alguma informação sobre as fontes, mas seu desempenho foi afetado pela quantidade elevada de ruído. Os áudios utilizados e obtidos nesse experimento se encontram em (MATOS, 2021).

Conclusões

Neste trabalho revisitamos o algoritmo de separação de fontes proposto em (FEVOTTE; GODSILL, 2006), bem como realizamos uma implementação eficiente na linguagem *Python*, lançando mão de compiladores *just-in-time* para ganhar eficiência computacional, reduzindo o tempo de cada iteração do algoritmo MCMC de 3 minutos para 4,3 segundos, em média. Também verificamos, empiricamente, a satisfatoriedade dos resultados obtidos. Próximos passos podem ser uma implementação utilizando a MDCT, ou tratando o modelo Bayesiano através de Inferência Variacional, em vez de MCMC.

Referências

AHMED, Nasir; NATARAJAN, T.; RAO, Kamisetty. Discrete Cosine Transform. *IEEE Transactions on Computers*, v.C-23, n.1, 1974, pp. 90-93.

CASELLA, George; BERGER, Roger. *Statistical Inference*. 2nd Edition. Pacific Grove, Boston: Cengage Learning, 2001.

FEVOTTE, Cedric.; GODSILL, Simon. A Bayesian Approach for Blind Separation of Sparse Sources, *IEEE Transactions on Audio, Speech, and Language Processing*, v.14, n.6, 2006, pp. 2174-2188.

MATOS, Natanael. Audio_Separation: Projeto de iniciação científica sobre abordagem bayesiana para separação de fontes sonoras. 2021. Disponível em <https://github.com/Natanael-Luciano/Audio_Separation>, Acessado em: 05 out. 2021.

MÜLLER, Meinard. *Fundamentals of Music Processing*: Audio, Analysis, Algorithms, Applications. New York: Springer, 2015.

ROBERT, Christian; CASELLA, George. *Introducing Monte Carlo Methods with R*. New York: Springer, 2010.