Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

O método da Análise de Componentes Independentes no problema da Separação Cega de Fontes

Marcos Henrique Castello

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr Marcelo Gomes de Queiroz

2023

Agradecimentos

Gostaria de iniciar esta monografia agradecendo a minha família por todos os sacrifícios feitos nesses meus anos de vida para me oferecer uma boa educação. Sem o apoio de minha mãe, meu pai e meu irmão eu não estaria aqui hoje. Gostaria também de agradecer o professor Marcelo por todo o apoio prestado durante este ano, por ter me apresentado ao assunto tratado aqui e por ter me guiado neste trabalho.

Resumo

Marcos Henrique Castello. **O método da Análise de Componentes Independentes no problema da Separação Cega de Fontes**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Em Processamento de Sinais Digitais, o problema da Separação Cega de Fontes trata de estimar o conteúdo de fontes geradoras de sinais a partir de suas misturas, sem o conhecimento de como tais misturas são formadas. O método da Análise de Componentes Indepentes aparece como uma forma de resolvê-lo analisando os sinais como variáveis aleatórias, e com isso utilizando conceitos da Teoria da Informação que passam a ser aplicáveis ao contexto, no caso os conceitos de Negentropia e Informação mútua. Neste trabalho o desenvolvimento deste método quando aplicado a este problema é apresentado, culminando na formação de um algoritmo que propõe realizar a tarefa estabelecida, obtendo resultados práticos. Com isso é feita uma implementação deste algoritmo, chamado FastICA, que possibilita três formas de estimar as fontes a partir das observações. Com o código, são realizados testes para comparar as performances estatísticas e computacionais entre cada uma das três formas citadas e também com uma implementação externa provida pela biblioteca da linguagem de programação python scikit-learn.

Palavras-chave: Processamento de Sinais Digitais. Análise de Componentes Independentes. Negentropia. Informação mútua. FastICA.

Abstract

Marcos Henrique Castello. **The method of Independent Component Analysis on the problem of Blind Source Separation**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

In Digital Signal Processing, the problem of Blind Source Separation is about estimating the content of signal generating sources from their mixtures, without knowledge on how these mixtures are formed. The method of Independent Component Analysis appears as a way of solving this problem analysing the signals as random variables, and with that using concepts of Information Theory that become applicable to the context, in this case the concepts of Negentropy and Mutual Information. In this work the development of such method when applied to this problem is presented, leading to the creation of an algorithm that proposes to solve this task, obtaining practical results. With that an implementation of such algorithm, called FastICA, is made in a way that enables three kinds of estimatives of the sources. With the code, tests are carried out to compare the statistical and computational performances between each one of the three forms and also with an external implementation provided by the python programming language library scikit-learn.

Keywords: Digital Signal Processing. Independent Component Analysis. Negentropy. Mutual Information. FastICA.

Lista de figuras

Sumário

Introdução

Um problema conhecido em processamento de sinais digitais é o chamado em inglês Blind Source Separation, que pode ser entendido a partir do seguinte exemplo: Suponha que dentro de uma sala fechada existam três fontes geradoras de som posicionadas em lugares quaisquer e três receptores que fazem registros dos sinais que chegam até eles. É possível notar nesta situação que cada receptor observará uma mistura diferente dos sinais gerados pelas três fontes sonoras, que são influenciadas pelas distâncias dos receptores às fontes. Essa situação é descrita pelo sistema de equações a seguir:

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t) + a_{13}s_3(t) \tag{1}$$
$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t) + a_{23}s_3(t) \tag{2}$$
$$x_3(t) = a_{31}s_1(t) + a_{32}s_2(t) + a_{33}s_3(t) \tag{3}$$

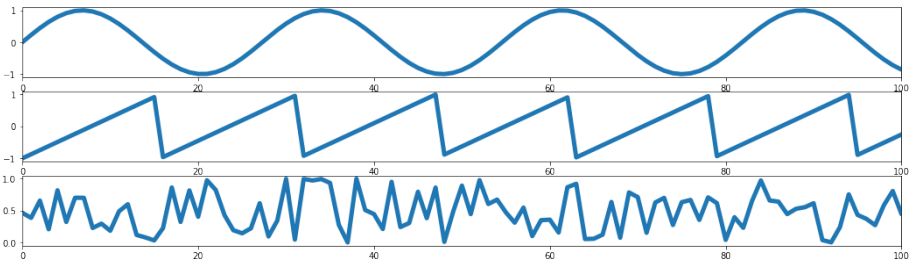

Figura 1: *Exemplo de sinais sem ruído que podem representar as fontes sonoras descritas anteriormente.*

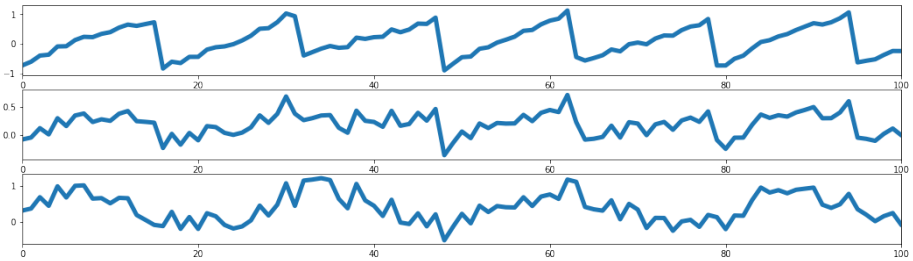

Figura 2: *Exemplo de sinais sem ruído que podem representar as misturas descritas anteriormente e feitas de acordo com o sistema de equações referido.*

Um outro modo de formular este problema é através da forma matricial das equações

anteriores. Generalizando a situação apresentada, sendo $\boldsymbol{X}$ um vetor $\boldsymbol{N}$-dimensional, cujas componentes são compostas a partir de combinações lineares das componentes de um vetor $\boldsymbol{M}$-dimensional $\boldsymbol{S}$, com coeficientes desconhecidos dados por uma matriz $\boldsymbol{A}$. O problema da Separação Cega de Fontes é a obtenção de $\boldsymbol{S}$ a partir de $\boldsymbol{X}$.

$$\boldsymbol{X}(\boldsymbol{t}) = \boldsymbol{A}\boldsymbol{S}(\boldsymbol{t}) \tag{4}$$

Neste trabalho considera-se que $\boldsymbol{N} = \boldsymbol{M}$, e portanto $\boldsymbol{A}$ é uma matriz quadrada. Além disso, a matriz é tratada como se fosse sempre inversível, o que, de acordo com A. Hyvärinen and Oja, 2000, ao avaliar as matrizes obtidas experimentalmente a partir de situações como a descrita anteriormente, na grande maioria dos casos é verdade. Com isso, é possível analisar o problema a partir da seguinte equação:

$$\boldsymbol{S}(\boldsymbol{t}) = \boldsymbol{W}\boldsymbol{X}(\boldsymbol{t}) \tag{5}$$

Onde $\boldsymbol{W} = \boldsymbol{A}^{-1}$. A partir deste momento, neste trabalho tanto S($\boldsymbol{t}$) quanto X($\boldsymbol{t}$) passarão a ser interpretadas como variáveis aleatórias multidimensionais centralizadas, e com isso os índices $\boldsymbol{t}$ representam "sorteios" dos valores que elas podem assumir, e então deixarão de ser utilizados nas notações (assim como é feito nos trabalhos de A. Hyvärinen and Oja, 2000, Aapo Hyvärinen *et al.*, 2001). Dessa forma, é possível determinar $\boldsymbol{S}$ a partir de $\boldsymbol{X}$ encontrando valores específicos de $\boldsymbol{W}$ que satisfaçam condições especiais. No caso, assume-se que as componentes de $\boldsymbol{S}$ são independentes, e então encontrar valores dos elementos da matriz que fazem com que as misturas $\boldsymbol{W}\boldsymbol{X}$ tenham componentes o mais independentes possível resulta na melhor estimativa de $\boldsymbol{S}$.

O raciocínio apresentado anteriormente é o que caracteriza o Método das Componentes Independentes (ICA, do inglês Independent Component Analysis), que será explorado pelo resto deste trabalho. Como apresentado por Comon, 1994, a Análise de Componentes Independentes de um vetor aleatório consiste na busca por uma transformação linear que minimiza a dependência estatística entre os seus componentes.

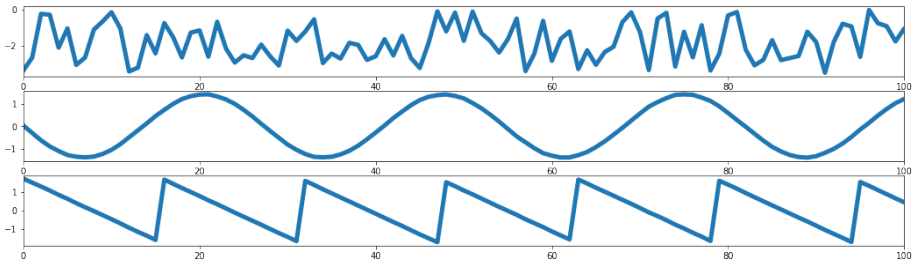

Figura 3: *Componentes estimadas aplicando-se o algoritmo FastICA, que será apresentado em capítulos posteriores, mas que tem como base o raciocínio empregado na explicação da ICA. É importante notar que somente é possível determinar as componentes até um múltiplo das originais, bem como também não é possível determinar a ordem original delas, como descrito por A. Hyvärinen and Oja, 2000.*

O resto deste trabalho será dividido da seguinte forma: A próxima seção apresentará o desenvolvimento matemático do método, introduzindo os conceitos da Teoria da In-

formação de entropia diferencial, negentropia e informação mútua, mostrando a relação entre essas propriedades e a independência estatística que se busca alcançar. Em seguida os conceitos apresentados serão utilizados para exibir o algoritmo FastICA, que busca maximizar a independência entre as componentes de um vetor aleatório $\boldsymbol{X}$. Com isso será mostrada uma implementação do algoritmo feita neste trabalho bem como uma avaliação da acurácia e da performance computacional utilizando sinais artificiais sem ruído. Por fim haverá uma discussão sobre a eficácia da implementação e das limitações que ela possui.

Capítulo 1

Desenvolvimento matemático

Neste capítulo, iremos definir conceitos que permitem encontrar uma forma de realizar o proposto pelo método da Análise de Componentes Independetes para solucionar o problema da Separação Cega de Fontes, definidos anteriormente. Esta forma foi apresentada nos trabalhos Hyvarinen, 1999 e Aapo Hyvärinen *et al.*, 2001. Os conceitos seguem definições atribuídas em Comon, 1994.

De agora em diante assume-se que todos os vetores aleatórios apresentados possuem matrizes de covariância inversíveis e média finita.

1.1 Entropia diferencial

A entropia diferencial de um vetor aleatório $X = (x_1, x_2, \dots, x_N)$ com função densidade de probabilidade $p_X(.)$ é definida como:

$$H(X) = -\int p_X(\epsilon) log(p_X(\epsilon)) d\epsilon \tag{1.1}$$

Em particular, a entropia diferencial de um vetor aleatório que apresenta uma distribuição de probabilidade Gaussiana de matriz de covariância C possui uma entropia diferencial da forma:

$$H(X) = \frac{N}{2}(log(2\pi) + 1) + \frac{1}{2}(log(|C|)) \tag{1.2}$$

Onde $|C|$ representa o determinante da matriz C. Da mesma forma, para uma váriavel aleatória de distribuição de probabilidades Gaussiana (x) com desvio padrão σ a entropia diferencial apresenta a forma:

$$H(x) = \frac{1}{2}(log(2\pi\sigma^2) + 1) \tag{1.3}$$

1.2 Negentropia

A negentropia de um vetor aleatório $X = (x_1, x_2, \dots, x_N)$ é definida da forma:

$$J(X) = H(X_G) - H(X) \tag{1.4}$$

Onde X_G é um vetor aleatório que segue uma distribuição de probabilidade Gaussiana e que possui a mesma matriz de covariância que X.

Uma propriedade importante da negentropia que vai influenciar na construção do algoritmo futuramente é a de que, de acordo com Comon, 1994, a negentropia é invariável quando aplicadas mudanças de coordenadas ou multiplicações por constantes no vetor em que se calcula (X no caso de (1.4)).

1.3 Informação mútua

A informação mútua de um vetor aleatório $X = (x_1, x_2, \dots, x_N)$ pode ser definida, de acordo com Aapo Hyvärinen *et al.*, 2001, a partir das entropias de $x_1, x_2, ..., x_N$ e de X:

$$I(x_1, x_2, \dots, x_N) = \sum_{i=1}^{N} H(x_i) - H(X) \tag{1.5}$$

Como apresentado em Comon, 1994, a informação mútua é nula se e somente se as componentes de X são independetes e é estritamente positiva caso contrário. É possível definir esta quantidade a partir da negentropia de X, utilizando conceitos apresentados anteriormente. Para isso, partimos de (1.4):

$$J(X) = H(X_G) - H(X) \tag{1.6}$$

$$J(X) - \sum_{i=1}^{N} J(x_i) = H(X_G) - H(X) - (\sum_{i=1}^{N} H(x_{G_i}) - \sum_{i=1}^{N} H(x_i)) \tag{1.7}$$

$$J(X) - \sum_{i=1}^{N} J(x_i) = I(X) + H(X_G) - \sum_{i=1}^{N} H(x_{G_i}) \tag{1.8}$$

$$I(X) = J(X) - \sum_{i=1}^{N} J(x_i) - H(X_G) + \sum_{i=1}^{N} H(x_{G_i}) \tag{1.9}$$

$$I(X) = J(X) - \sum_{i=1}^{N} J(x_i) - [\frac{N}{2}(1 + log(2\pi)) + \frac{1}{2} log(|C_X|)] + \sum_{i=1}^{N} [\frac{1}{2}(log(2\pi\sigma_i^2) + 1)] \tag{1.10}$$

Com isso, assumindo que as variáveis aleatórias $(x_1, \dots, x_N)$ são não correlacionadas, a matriz de covariância de X, C_X, se torna diagonal, e logo seu determinante é calculado como o produto das variâncias de suas componentes. Dessa forma, (1.10) pode ser

simplificada:

$$I(X) = J(X) - \sum_{i=1}^{N} J(x_i) \tag{1.11}$$

A partir desta equação, é possível formular a Análise de Componentes Independentes como sendo a maximização das negentropias das componentes do vetor de observações, pois assim a informação mútua das componentes é minimizada, seguindo o raciocínio apresentado anteriormente.

Assim, a Análise de Componentes Independentes de um vetor aleatório X pode ser definida como uma transformação invertível $S = WX$ onde W é determinada de forma que a informação mútua das componentes transformadas s_i é minimizada. A partir dessa definição será desenvolvido um algoritmo que visa calcular o melhor valor de W que satisfaça essa condição.

Capítulo 2

O algoritmo FastICA

No capítulo anterior concluiu-se que é possível definir o Método da Análise de Componentes Independentes no problema que está sendo estudado a partir de uma transformação invertível $S = WX$ onde W é tal que a informação mútua das componentes de S é minimizada. Neste capítulo será apresentado um algoritmo, chamado FastICA, que se dispõe a aplicar este método a partir de uma matriz de valores que é interpretada como o vetor aleatório de observações X discutido anteriormente. Além disso será apresentado um pré-processamento que é necessário para a aplicação deste algoritmo (como afirmado em Hyvarinen, 1999), que recebe o nome de esferamento.

2.1 Esferamento

Como apresentado anteriormente, o esferamento consiste em uma etapa de pré-processamento que é necessária para a execução do FastICA. Um vetor aleatório Z é dito esferado se todas as suas componentes são não correlacionadas entre si e suas variâncias valem 1. Em outras palavras, Z é dito esferado se $C_Z = I$, onde C_Z é a matriz de covariância de Z e I é a matriz identidade.

É possível fazer com que qualquer vetor aleatório cuja matriz de covariância possua apenas autovalores positivos passe a ser esferado. Para isso é necessário notar que para qualquer vetor aleatório X, C_X é positiva semidefinida, ou seja, para qualquer vetor h de mesma dimensão que X, $h^T C_X h \geq 0$, onde 0 é o vetor nulo de mesma dimensão que X. Aqui novamente o vetor X é considerado centralizado.

$$C_X = E\{XX^T\} \implies h^T E\{XX^T\}h = E\{h^T XX^T h\} \tag{2.1}$$

Seja $Y = h^T X$. Como $h^T X = X^T h$, temos:

$$E\{h^T XX^T h\} = E\{Y^2\} \geq 0 \tag{2.2}$$

Pois a esperança de qualquer função não negativa é não negativa. Além disso, Aapo Hyvärinen *et al.*, 2001 argumenta que na prática C_X é positiva definida na absoluta maioria

das observações (ou seja, a desigualdade em (2.2) é estritamente positiva), o que faz com que os autovalores de C_X sejam sempre positivos, fato este que será assumido como sendo sempre verdadeiro de agora em diante.

Sendo assim, sendo $E = (e_1, e_2, \dots, e_N)$ uma matriz que tem como colunas os autovetores de C_X e $D = diag(d_1, d_2, \dots, d_N)$ uma matriz diagonal com os autovalores desta, sabe-se que C_X pode ser decomposta da forma $C_X = EDE^T$. Além disso, a matriz $V = D^{-\frac{1}{2}}E^T$, aplicada sobre o vetor X, transforma-o em um vetor $Z = VX$ tal que Z é esferado. Note que $D^{-\frac{1}{2}}$ necessariamente existe, pois os autovalores de C_X são positivos. Outro fato importante é que E é uma matriz ortogonal, o que significa que $EE^T = I$. Com isso, Z é esferado pois:

$$C_Z = VE\{XX^T\}V = D^{-\frac{1}{2}}E^T EDE^T D^{-\frac{1}{2}}E^T \tag{2.3}$$

$$= D^{-\frac{1}{2}}E^T EDE^T ED^{-\frac{1}{2}} \tag{2.4}$$

$$= D^{-\frac{1}{2}}IDID^{-\frac{1}{2}} \tag{2.5}$$

$$= D^{-\frac{1}{2}}DD^{-\frac{1}{2}} = I \tag{2.6}$$

2.2 Aproximação da Negentropia

Para derivar um algoritmo capaz de aplicar o método ICA, é necessário primeiro desenvolver uma aproximação do cálculo da negentropia, definida no capítulo de desenvolvimento matemático, que seja eficiente de se computar, pois a definição original, $J(X) = H(X_G) - H(X)$, necessita dos valores da função densidade de probabilidade de X para a obtenção da entropia diferencial.

Uma aproximação, obtida em Aapo Hyvärinen, 1997, para uma variável aleatória y_i de média nula e variância unitária, é tal que:

$$J_K(y_i) = [E\{K(y_i)\} - E\{K(v)\}]^2 \tag{2.7}$$

Onde v é uma variável aleatória de distribuição Gaussiana padronizada e K é uma função, de agora em diante chamada função de contraste. A escolha da função K influenciará a performance do algoritmo para tipos diferentes de entradas, e mais a frente a sua escolha será o objeto das comparações a serem feitas na testagem da implementação. De acordo com Hyvarinen, 1999, as três principais funções de contraste são:

$$K_1(u) = \frac{1}{a_1} log(cosh(a_1 u)) \tag{2.8}$$

$$K_2(u) = -\frac{1}{a_2} \exp(-\frac{a_2 u^2}{2}) \tag{2.9}$$

$$K_3(u) = \frac{1}{4}u^4 \tag{2.10}$$

Para o algoritmo que será apresentado, as funções dadas pelas derivadas destas anteriormente apresentadas são de suma importância, e posteriormente será notado que elas influenciam diretamente na eficácia do algoritmo. Por isso, é importante apresentá-las:

$$k_1(u) = tanh(a_1 u) \tag{2.11}$$

$$k_2(u) = u \exp(-a_2 \frac{u^2}{2}) \tag{2.12}$$

$$k_3(u) = u^3 \tag{2.13}$$

Onde a_1 e a_2 são constantes tais que $1 \leq a_1 \leq 2$ e $a_2 \approx 1$.

A partir de (2.7) é possível definir uma função objetiva para estimar a transformação ICA de uma componente de S. Sendo w_i uma das linhas da matriz W, é possível estimar o valor de uma das componentes ao maximizar o valor da função, restringindo w_i de forma que $E\{(w_i^T X)^2\} = 1$:

$$J_K(w_i) = [E\{K(w_i^T X)\} - E\{K(v)\}]^2 \tag{2.14}$$

Seguindo esta equação, é possível estimar todas as componentes de S, tornando todos os w_i, w_j não correlacionados, criando o problema de otimização:

$$\text{maximize} \sum_{i=1}^{N} J_K(w_i) \tag{2.15}$$

$$\text{under constraint } E\{(w_j^T X)(w_k^T X)\} = \delta_{jk} \tag{2.16}$$

Onde $\delta_{jk} = 1$ se $j = k$, e 0 caso contrário.

2.3 Cálculo de uma componente da fonte

De início o algoritmo desenvolvido será utilizado para se calcular apenas uma das componentes do vetor de fontes, tendo como base o vetor de observações esferado. A partir deste, é possível calcular todas as componentes aplicando-o para cada linha da matriz W e seguindo um método de decorrelação entre os resultados. Esta forma de calcular as fontes é a que será implementada e testada posteriormente.

De (2.14) note que $E\{K(v)\}$ assume um valor constante para uma mesma função K, pois v é sempre uma variável Gaussiana padrão. Como a motivação aqui é determinar apenas uma componente, os índices i em (2.14) deixarão de ser utilizados de forma a tornar mais fácil a compreensão. Assim, w_i, que representa a i-ésima linha da matriz W será representada nesta seção como w. Com isso os valores máximos de $J_K(w)$ são obtidos

em pontos específicos de $E\{K(w^T X)\}$. Em Hyvarinen, 1999 apresenta-se que tais pontos ótimos, sob a restrição $E\{(w^T X)^2\} = 1$ são tais que:

$$E\{Xk(w^T X)\} - E\{w_0^T X k(w_0^T X)\} w = 0 \tag{2.17}$$

Onde $k = \frac{dK}{du}$ é a derivada de K e w_0 é o valor de w que maximiza $J_K(w)$. É possível encontrar o valor ótimo de w nesta equação utilizando o método de Newton: Seja $F(w)$ a função definida pelo lado esquerdo da equação (2.17), tenta-se encontrar o valor de w tal que $F(w) = 0$, realizando sucessivamente as operações

$$w^+ = w - (J(F)(w))^{-1} F(w) \tag{2.18}$$

$$w = w^+ \tag{2.19}$$

até a convergência dos valores de w^+ e w, onde $J(F)(w)$ representa a matriz Jacobiana da função F no vetor w. No caso de F representada pela equação (2.17), a matriz Jacobiana é dada por:

$$J(F)(w) = E\{XX^T k'(w^T X)\} - E\{w_0^T X k(w_0^T X)\} \tag{2.20}$$

Tendo em vista que para realizar as operações dadas em (2.18) é necessário calcular a inversa de (2.20), se torna necessário aproximar alguns termos desta de modo a simplificar a inversão e de torná-la possível, pois o valor de w_0 é desconhecido durante o processo. Por isso considera-se que $E\{XX^T k'(w^T X)\} \approx E\{XX^T\} E\{k'(w^T X)\} = E\{k'(w^T X)\} I$, tornando $J(F)(w)$ diagonal e portanto necessariamente invertível, e $w_0 \approx w$, fazendo com que o seu cálculo seja possível. Assim obtem-se a seguinte iteração:

$$w^+ = w - [E\{Xk(w^T x)\} - E\{w^T X k(w^T X)\} w] / [E\{k'(w^T X)\} - E\{w^T X k(w^T X)\}] \tag{2.21}$$

$$w^* = w^+ / \|w^+\| \tag{2.22}$$

$$w = w^* \tag{2.23}$$

Note que é possível simplificar a multiplicação em (2.21) multiplicando os dois termos da equação por $[E\{w^T X k(w^T X)\} - E\{k'(w^T X)\}]$, obtendo:

$$w^+ = E\{Xk(w^T X)\} - E\{k'(w^T X)\} w \tag{2.24}$$

Obtendo um valor de w, é possível calcular o valor de uma das componentes do vetor de fontes, s, através do produto de w com o vetor de observações X, $s = wX$. Em Hyvarinen, 1999 é apresentado o argumento de que nem sempre essas iterações convergem a um resultado, porém na grande maioria dos casos tal método converge e apresenta resultados satisfatórios.

2.4 O algoritmo FastICA

Na seção anterior foi apresentada uma forma de se estimar uma das componentes do vetor de fonte S através do método de Newton. O algoritmo FastICA é uma extensão deste processo, aplicando-o para todos os w_i.

Aplicando repetidas vezes as operações apresentadas em 2.21 para cada linha da matriz W, obtem-se todos os vetores iguais. Por isso o FastICA pode ser definido calculando cada vetor por vez, e após a obtenção do m-ésimo vetor, torna-o não correlacionado com todos os $m - 1$ já calculados, para isso utilizando o método de ortogonalização de Gram-Schmidt:

$$w_m = w_m - \sum_{i=1}^{m-1} w_m^T w_i w_i \tag{2.25}$$

$$w_m = w_m / \sqrt{w_m^T w_m} \tag{2.26}$$

Com isso, para calcular todos os N vetores de pesos, ou seja, para calcular todas as componentes do vetor de fontes, o algoritmo FastICA calcula cada linha da matriz W por vez e a ortogonaliza com as anteriormente obtidas. Assim, este algoritmo pode ser definido da forma:

Programa 2.1 FastICA

```
1   FUNCAO FastICA(x, N)   ▷ x é um vetor esferado, N é a sua dimensao
2       W ⟵ matriz NxN inicializada aleatoriamente
3       para i=1:N
4           w_i ⟵ i–ésima linha da matriz W   ▷ interpretada como vetor coluna
5           enquanto w_i não convergir:
6               w+ = E{x k(w_i^T x)} − E{k'(w_i^T x)} w_i
7               w_i = w+/||w+||
8           para j=1:i
9               w_i = w_i − w_i^T w_j w_j
10          w_i = w_i / sqrt(w_i w_i^T)
11          W[i] = w_i
12
13      devolva W
```

Capítulo 3

Implementação e Testagem

Os capítulos anteriores apresentaram o contexto em que o método da Análise de Componentes Independentes é desenvolvido, bem como um algoritmo utilizado para aplicá-lo no problema da Separação Cega de Fontes, em que existem observações de misturas de fontes sonoras e deseja-se estimá-las. Neste capítulo será apresentada a implementação do FastICA anteriormente discutido.

Além disso, em (2.7) foi mostrada a possibilidade de variação das chamadas funções de contraste que são utilizadas na estimação da negentropia, de suma importância na execução do algoritmo, como em (2.1). Por isso a implementação realizada concede espaço para alternância destas funções, entre as três apresentadas em (2), tornando possível a comparação entre os resultados obtidos para cada uma destas, o que leva à possibilidade da realização de uma testagem entre elas. Para isso este trabalho inspirou-se em GIANNAKOPOULOS *et al.*, 1999, que apresenta maneiras de avaliar cada uma das funções.

3.1 Implementação

A implementação do algoritmo, realizada na linguagem Python 3, é dividida em três classes: A chamada FastICA, que realiza as operações descritas pelo algoritmo (2.1), utilizando como função de contraste alguma das apresentadas pela classe ContrastFunction, que é uma interface para a criação dessas funções, e a classe Preprocessing, que aplica o pré-processamento necessário para utilizar o algoritmo, a saber, o processo de esferamento descrito no capítulo (2). Nesta monografia apenas uma parte específica do código será apresentada, o jupyter notebook onde todo o programa foi escrito e os testes foram realizados está disponível no repositório onde este trabalho se encontra.

3.1.1 Pré-processamento

O Pré-processamento, realizado pela classe Preprocessing, resume-se à aplicação do processo de esferamento. Para isso, é necessário primeiro notar que no contexto do problema não se possui o vetor aleatório das misturas, por isso também não é possível ter acesso a sua matriz de covariância. Dessa forma, estima-se esta matriz a partir das observações

deste vetor, dadas por uma matriz de agora em diante tratada como $\boldsymbol{X}$, como mostrado a seguir, onde a função "center" centraliza as observações.

Programa 3.1 Estimação da matriz de covariância de $\boldsymbol{X}$.

```
1   def covMatrix(self):
2
3       num_samples = np.shape(self.mat)[1]
4
5       self.center()
6
7       cov_mat = np.dot(self.mat, self.mat.T)
8       cov_mat = cov_mat / num_samples
9
10      return cov_mat
```

Com isso, calculam-se as matrizes de autovetores e autovalores, como apresentadas em (2), a partir da função própria da biblioteca Numpy "numpy.evd()". Assim, o esferamento termina ao calcular a matriz $\boldsymbol{D}^{-\frac{1}{2}}$ e com isso obter $\boldsymbol{V} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{E}^{T}$.

Programa 3.2 Obtenção da matriz de esferamento $\boldsymbol{V}$.

```
1   def sphering(self):
2
3       self.center()
4       cov = self.covMatrix()
5
6       D, E = self.evd(cov)
7
8       diag = np.zeros((len(D), len(D)))
9
10      for i in range(len(D)):
11          diag[i][i] = 1.0 / np.sqrt(np.absolute(D[i]))
12
13      V = np.dot(diag, E.T)
14
15      return V
```

3.1.2 Implementação do FastICA

Após calcular a matriz de esferamento $\boldsymbol{V}$ e aplicá-la às observações $\boldsymbol{X}$, obtêm-se uma matriz de observações cuja covariância equivale à identidade, descrita como $\boldsymbol{Z}$ no capítulo (2). Utilizando $\boldsymbol{Z}$ como observações a implementação do algoritmo passa a ser uma aplicação direta do descrito em (2.1), como apresentado a seguir. Assim como no pré-processamento, este algoritmo foi implementado em uma classe para facilitar a escrita do código. As funções de contraste são atribuídas no construtor desta classe. É importante notar que "convergence_parameter" foi adicionado para tratar de uma função de contraste, a saber, (2.10), que ao ser utilizada pode fazer com que os valores atualizados de $\boldsymbol{w}$ não atinjam a convergência. Além disso, "kPrime" denota a função dada pela derivação da função de contraste utilizada.

Programa 3.3 Implementação do algoritmo FastICA descrito em (2.1).

```
1    def fastICA(self):
2       z = self.z
3
4       # Quantidade de fontes que queremos estimar
5       # e quantidade de observações que cada mistura tem
6       m, n = np.shape(z)
7
8       p = 1
9
10      W = np.array([np.zeros(m)])
11
12      while p <= m:
13
14         # w tem que ter dimensao m
15
16         wp = np.random.rand(m)
17
18         wp = self.normalize(wp)
19
20         wp_prev = np.zeros(m)
21
22         count = 0
23
24         while 1 - np.dot(wp, wp_prev.T) >= 1e-8:
25
26            wp_prev = wp
27
28            k_value = self.func.k(np.dot(wp, z))
29            kPrime_value = self.func.kPrime(np.dot(wp, z))
30
31            if self.convergence_parameter == True:
32               wp = (np.mean(z * k_value, axis=1) - 0.01 * np.mean(kPrime_value)
                       * wp)
33            else:
34               wp = (np.mean(z * k_value, axis=1) - np.mean(kPrime_value) * wp)
35
36            summation = 0.0
37            for j in range(1, p):
38
39               summation = summation + np.dot(np.dot(wp.T, W[j, :]), W[j, :])
40
41            wp = wp - summation
42
43            wp = self.normalize(wp)
44
45            count = count + 1
46
47         W = np.concatenate((W, [wp]), axis=0)
48
49         p = p + 1
50
51      return W[1:m+1, :]
```

3.2 Testagem

Como discutido no capítulo (1), a negentropia é invariante sobre mudanças de posições entre as componentes dos vetores e multiplicações por constantes. Assim, o vetor de componentes obtido ao término da execução do algoritmo pode não ser idêntico ao original, pois suas componentes podem estar em ordens diferentes ou multiplicadas por algum outro valor, o que dificulta a comparação direta entre eles. Dessa forma, preferiu-se realizar uma testagem comparando a acurácia e a performance do algoritmo, variando as funções de contraste entre as três apresentadas em (2). Além disso, também foram feitas comparações com a implementação do FastICA disponibilizada pela biblioteca Scikit-Learn, que é muito usada para projetos de Aprendizado de Máquina, de forma a medir as qualidades estatísticas e computacionais de implementações de fora deste trabalho.

O vetor de componentes original é criado artificialmente de modo a garantir a falta de ruídos e o controle da quantidade de fontes e da dimensão das observações, que como tratado em todo este trabalho, devem ser iguais. Posteriormente será discutida a situação onde há a existência de ruído e as dimensões entre as observações e as fontes não são iguais. No geral, apesar das restrições impostas no desenvolvimento deste algoritmo, Aapo Hyvärinen *et al.*, 2001 argumenta que o FastICA na forma tratada aqui possui muitas aplicações reais.

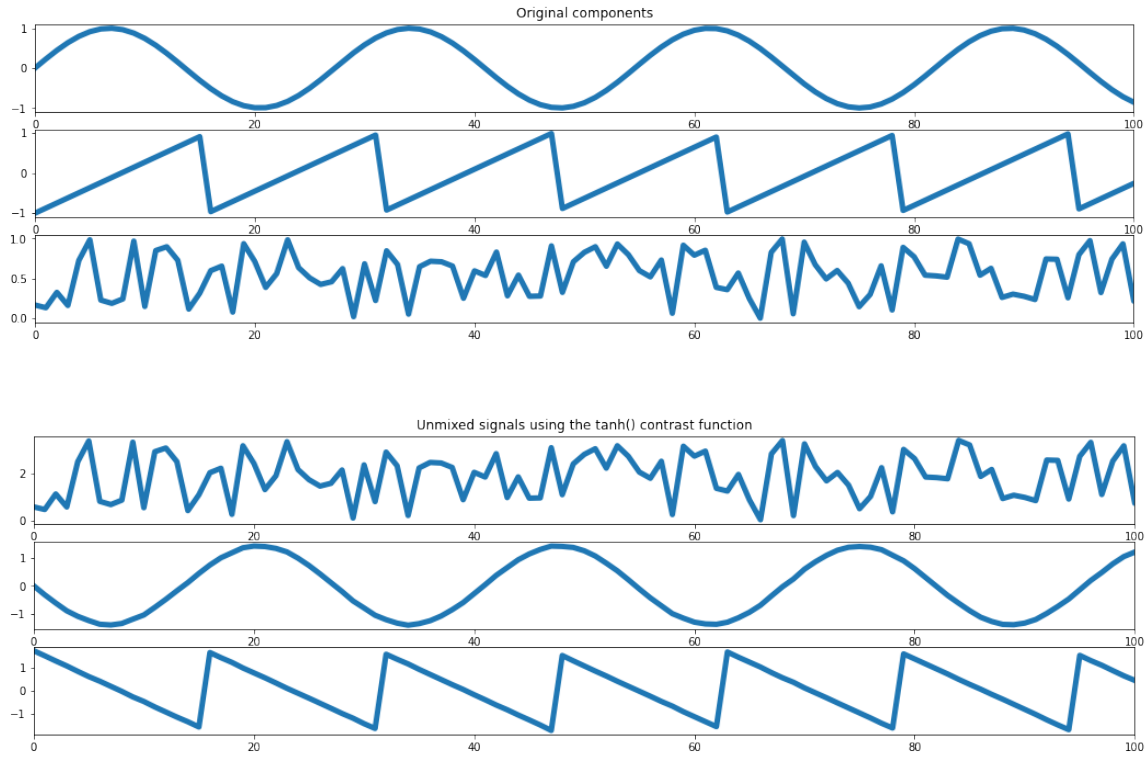

Figura 3.1: *Comparação dos resultados obtidos utilizando o FastICA. Aqui é possível notar o que foi anteriormente dito sobre a dificuldade de se comparar os vetores obtidos e originais. Como o valor da negentropia de um vetor aleatório não varia quando este tem a ordem de suas componentes invertidas ou se as suas componentes estão multiplicadas por constantes, existem infinitos vetores diferentes que apresentam o mesmo valor que o original. O algoritmo é capaz de estimar um destes possíveis vetores.*

3.2.1 Índice de Performance Estatística

Para realizar as comparações entre as acurácias das diferentes funções foi utilizado um índice, apresentado em Aapo Hyvärinen *et al.*, 2001 e em GIANNAKOPOULOS *et al.*, 1999, para avaliar a proximidade entre a inversa da matriz de misturas A^{-1} gerada artificialmente e a matriz estimada para transformar as observações nos dados originais, no caso da implementação deste trabalho trata-se da matriz WV e para a função da biblioteca ScikitLearn é o atributo components_ de um objeto da classe FastICA. Este índice indica o quão próximo o produto destas matrizes é de uma matriz de permutação, que para toda linha e coluna possui um elemento unitário e os outros nulos. É descrito pela equação, onde m corresponde a dimensão destas matrizes:

$$E_1 := \sum_{i=1}^{m}\left(\sum_{j=1}^{m}\frac{|p_{ij}|}{\max_k |p_{ik}|} - 1\right) + \sum_{j=1}^{m}\left(\sum_{i=1}^{m}\frac{|p_{ij}|}{\max_k |p_{jk}|} - 1\right) \tag{3.1}$$

Onde p_{ij} é o ij-ésimo elemento da matriz P, dada pelo produto das matrizes anteriormente explicado. A partir deste valor, foram realizados testes para medir estes valores em cada uma das funções de contraste e na função pronta para cinco diferentes vetores de fontes.

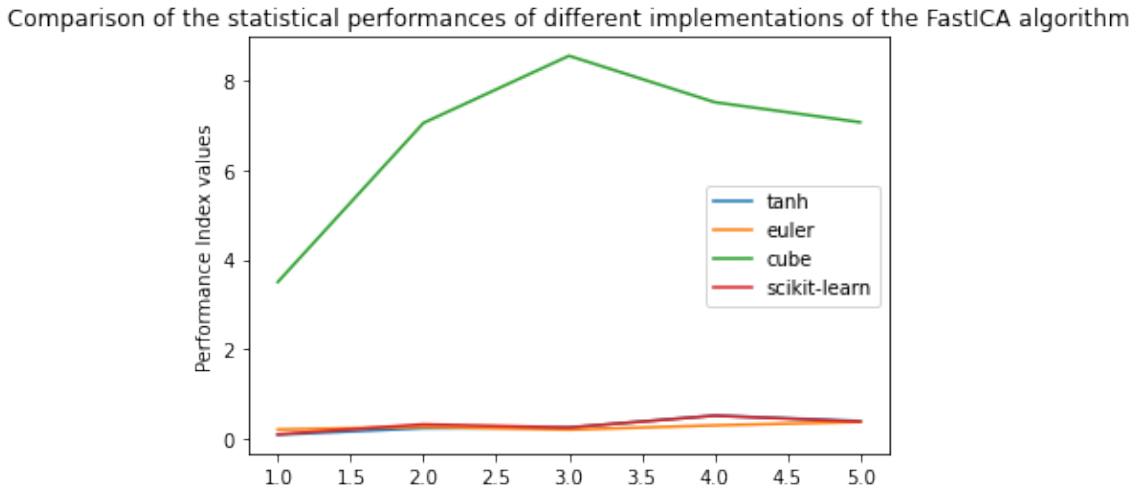

Figura 3.2: *Comparação entre os valores determinados pelos índices de acurácia para cada uma das quatro situações possíveis: Utilizando as funções (2.8) ou (2.9) ou (2.10) ou utilizando a implementação externa dada pela biblioteca ScikitLearn.*

Deste gráfico é possível notar que a implementação utilizando a função de contraste denotada por "cube", descrita em (2.10), apresenta um valor de E_1 muito maior que o das outras implementações. Assim tornou-se interessante comparar as acurácias apenas entre as outras implementações. Analisando esta parte do gráfico, gerado em um novo espaço a seguir, pode-se notar que os valores de E_1 para as funções (2.8) e (2.9) são muito próximos aos valores obtidos pela implementação externa, o que indica que o FastICA desenvolvido neste trabalho tem uma acurácia satisfatória. A partir destes dados também foi possível perceber que estes valores devem variar para diferentes vetores de fontes, como o esperado em Aapo Hyvärinen *et al.*, 2001.

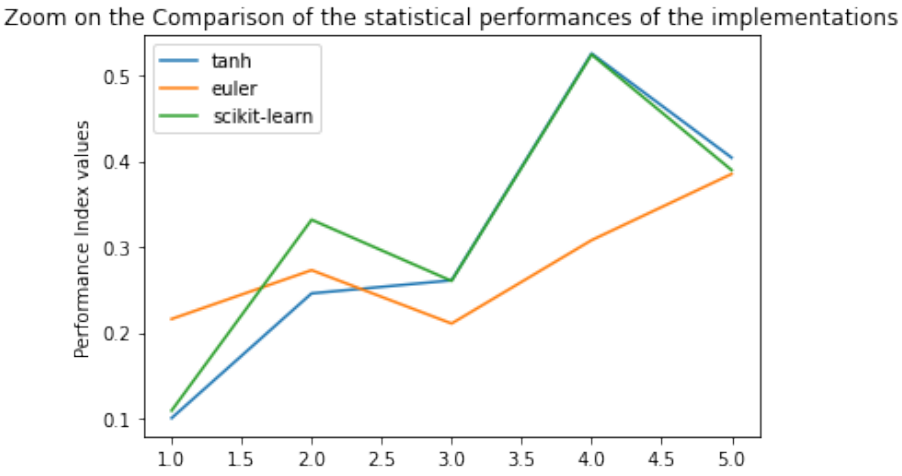

Figura 3.3: *Gráfico dos valores de índices de performance estatística para as implementações com as funções (2.9) e (2.8) e para a disponibilizada pela biblioteca ScikitLearn.*

3.2.2 Performance Computacional

Para medir a performance computacional de cada uma das implementações foi utilizada uma biblioteca disponível no Python 3 chamada papi, que trata de medir a quantidade de operações de ponto flutuante realizadas em um determinado trecho de código. Com isso, como a implementação externa do algoritmo realiza o processo de esferamento internamente, a contagem de operações levou em conta as operações de esferamento para cada versão do FastICA. Dessa forma, foi possível gerar um gráfico para comparar os valores medidos para cada ocasião:

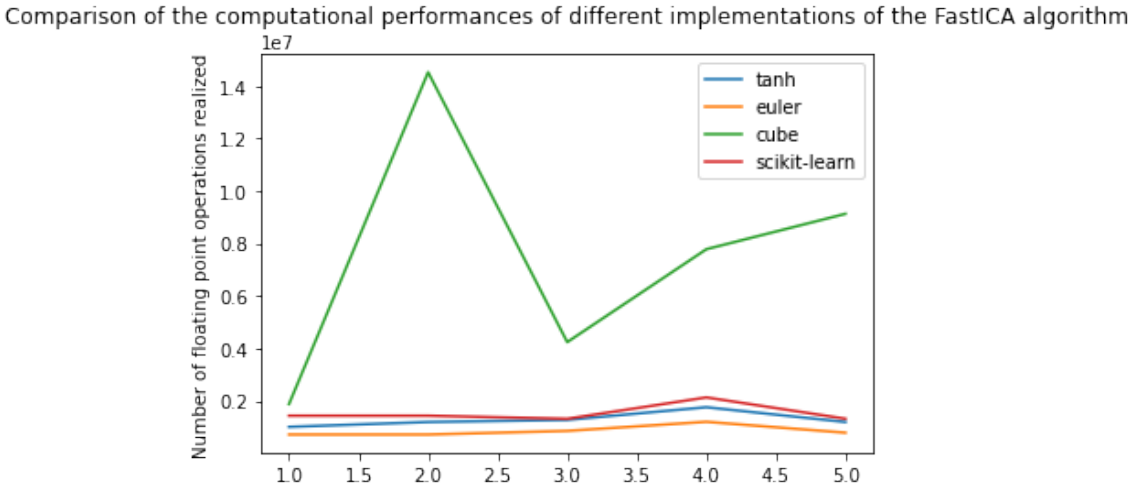

Figura 3.4: *Comparação da quantidade de operações de ponto flutuante realizadas para cada variação da implementação do algoritmo FastICA deste trabalho e da biblioteca ScikitLearn.*

Neste gráfico é possível notar que novamente a implementação que utiliza a função dada por (2.10) apresenta performance muito inferior se comparada às demais. Por isso, assim como no caso da performance estatística, é interessante analisar o gráfico concentrado apenas nas demais implementações.

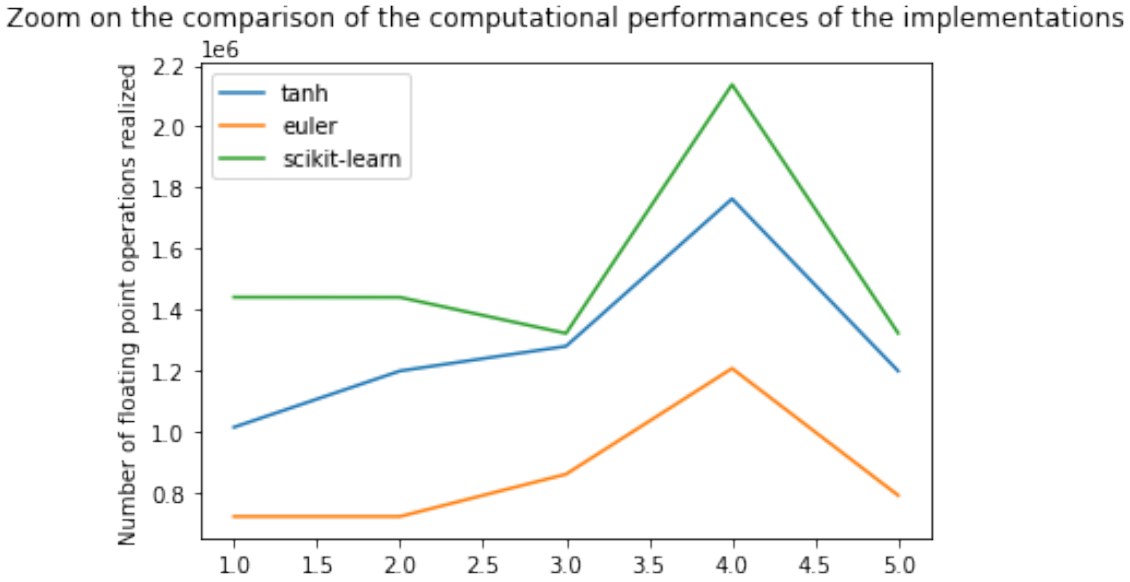

Figura 3.5: *Gráfico dos valores de operações de ponto flutuante realizadas pelo FastICA utilizando a função de contraste (2.8) e (2.9) e pelo provido externamente pela biblioteca descrita. Note que aqui são apresentados valores maiores para a implementação do ScikitLearn do que para as deste trabalho, isso se deve a algumas operações de checagem que a função externa realiza de modo a garantir que será possível realizar os cálculos, o que natural para uma classe de uma biblioteca disponível publicamente para se utilizar.*

Concentrando a análise apenas nas implementações mais eficientes pela medição desta biblioteca, é possível perceber que a performance computacional das variantes da implementação deste trabalho para as funções (2.8) e (2.9) é muito próxima da performance dada pelo ScikitLearn, o que indica que a eficiência destas pode ser considerada satisfatória.

Capítulo 4

Conclusão

Este trabalho tratou de apresentar como o método da Análise de Componentes Independentes é aplicado ao problema da Separação Cega de Fontes. Iniciando com o capítulo de Desenvolvimento Matemático, foram apresentados os conceitos necessários para se chegar ao método da Análise de Componentes Independetes a partir da minimização da Informação Mútua. Além disso, foram apresentadas restrições no contexto do problema necessárias para se chegar ao algoritmo que este trabalho expôs.

Prosseguindo, o capítulo sobre o algoritmo FastICA apresentou detalhes do método que são importantes ao tratá-lo computacionalmente, como a necessidade de estimar a negentropia e o processo de esferamento do vetor de observações de forma a facilitar o processamento dos cálculos que vieram a ser apresentados na descrição da formação do algoritmo. Por fim, este foi descrito a partir de um pseudocódigo, que facilita o entendimento da implementação que foi fornecida no capítulo seguinte.

Finalmente, o capítulo de implementação e testagem discutiu a forma como escrever o código do algoritmo apresentado anteriormente visando esclarecer alguns detalhes de cálculo que não foram apresentados nos capítulos anteriores, como o vetores aleatórios a partir de matrizes com suas observações, mostrando a necessidade de fazer certas estimativas que não tinham sido levadas em conta. Além disso, foi apresentada a possibilidade de comparar a acurácia e a performance computacional do algoritmo variando suas implementações a partir de diferentes funções que são utilizadas para o cálculo dos resultados. Essas implementações também foram comparadas com uma disponível na biblioteca da linguagem Python ScikitLearn, de modo a saber se o código produzido neste trabalho possui uma performance satisfatória com os popularmente usados. Essa necessidade nasceu da dificuldade de se comparar os resultados obtidos com os originalmente criados. Com isso foi possível perceber que uma variante da implementação apresenta resultados piores através de um número maior de operações de ponto flutuante, mas que as outras podem ser comparáveis à fornecida pela biblioteca, concluindo que tais implementações podem ser consideradas aceitáveis.

Concluíndo, apesar das restrições impostas aqui para determinar o algoritmo FastICA, este apresenta resultados interessantes para o problema da Separação Cega de Fontes. Além disso, Aapo Hyvärinen *et al.*, 2001 indica que as melhores formas de analisar o problema

sem tais restrições, a saber, nos casos em que as observações apresentam ruídos e em que as dimensões do vetor de observações e do vetor de fontes são distintas, são realizando tentativas de aproximar tais condições para as originalmente propostas neste trabalho, utilizando para isso outras técnicas que não serão apresentadas aqui e que constituem uma área ativa de pesquisa.

References

[Comon 1994] Pierre Comon. "Independent component analysis, a new concept?" *Signal Processing* 36.3 (1994). Higher Order Statistics, pp. 287–314. ISSN: 0165-1684. DOI: https://doi.org/10.1016/0165-1684(94)90029-9. URL: https://www.sciencedirect.com/science/article/pii/0165168494900299 (cit. on pp. 2, 5, 6).

[Giannakopoulos *et al.* 1999] Xavier Giannakopoulos, Juha Karhunen, and Erkki Oja. "An experimental comparison of neural algorithms for independent component analysis and blind separation". *International Journal of Neural Systems* 09.02 (Apr. 1999), pp. 99–114. ISSN: 1793-6462. DOI: 10.1142/s0129065799000101. URL: http://dx.doi.org/10.1142/s0129065799000101 (cit. on pp. 15, 19).

[Hyvarinen 1999] A. Hyvarinen. "Fast and robust fixed-point algorithms for independent component analysis". *IEEE Transactions on Neural Networks* 10.3 (1999), pp. 626–634. DOI: 10.1109/72.761722 (cit. on pp. 5, 9, 10, 12).

[A. Hyvärinen and Oja 2000] A. Hyvärinen and E. Oja. "Independent component analysis: algorithms and applications". *Neural Networks* 13.4 (2000), pp. 411–430. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(00)00026-5. URL: https://www.sciencedirect.com/science/article/pii/S0893608000000265 (cit. on p. 2).

[Aapo Hyvärinen 1997] Aapo Hyvärinen. "New approximations of differential entropy for independent component analysis and projection pursuit". In: *Advances in Neural Information Processing Systems*. Ed. by M. Jordan, M. Kearns, and S. Solla. Vol. 10. MIT Press, 1997. URL: https://proceedings.neurips.cc/paper_files/paper/1997/file/6d9c547cf146054a5a720606a7694467-Paper.pdf (cit. on p. 10).

[Aapo Hyvärinen *et al.* 2001] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. Wiley, May 2001. DOI: 10.1002/0471221317. URL: https://doi.org/10.1002%2F0471221317 (cit. on pp. 2, 5, 6, 9, 18, 19, 23).