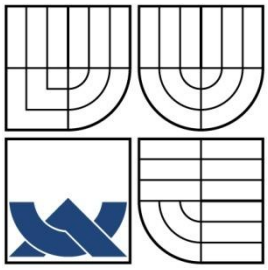
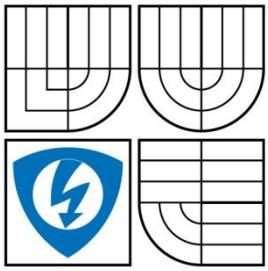


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

Implementace algoritmů slepé separace zdrojů v jazyce C/C++

IMPLEMENTATION OF BLIND SOURCE SEPARATION ALGORITHMS IN C/C++ LANGUAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — MARCEL FUNDERÁK
AUTHOR

VEDOUCÍ PRÁCE — ING. IVAN MÍČA
SUPERVISOR

BRNO 2007

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:

Bytem:

Narozen/a (datum a místo):

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - ☑ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako

   .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: .....

Vedoucí/ školitel VŠKP: .....

Ústav: .....

Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v*:

☑ tištěné formě – počet exemplářů ……..

☑ elektronické formě – počet exemplářů ……..

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: …….

…….. ……
Nabyvatel Autor

Anotace

Tato práce se zabývá jednou z metod slepé separace zdrojů (BSS), která se nazývá analýza nezávislých komponentů (z anglického Independent Component Analysis). Je uveden stručný teoretický podklad, ve kterém jsou vysvětleny základní poznatky důležité pro odvození jednotlivých algoritmů ICA. Tyto teoretické poznatky zahrnují zejména vysvětlení základních znalostí ze statistiky.

V další části jsou popsány metody vhodné k předzpracování vstupních signály – analýza hlavních komponent (PCA) a bělení signálů. Zejména bělení je důležitou součástí řešení algoritmů ICA.

Poté jsou již nastíněna různá řešení algoritmů ICA, jakož i úvod do této problematiky. Mezi uvedenými postupy lze vyzvednout popis algoritmů FastICA, které se jeví jako velice vhodné pro počítačové zpracování, jelikož jsou robustní a také nejsou výpočetně náročné oproti jiným metodám.

Dále je nastíněno zpracování jednoho algoritmu ICA v jazyce C++, konkrétně algoritmus FastICA pro komplexní signály.

Klíčová slova:

slepá separace zdrojů, analýza nezávislých komponentů, algoritmus, implementace, C++, FastICA, bělení

Abstract

This thesis is describing one of the methods of Blind Source Separation (BSS) which is Independent Component Analysis. There is shown some brief introduction to the theory behind in which there are explained some basic findings. These findings are important for understanding the theory behind algorithms of ICA. These theoretical findings include primarily explanations of basic knowledge of statistics science.

In next part there are described methods which are advisable for preprocessing of input signals – mainly Principal Component Analysis (PCA) and whitening of signals. Mainly whitening is very important part of solution of ICA algorithms.

Then there are described different ICA algorithm solutions and especially introduction in this problematic. FastICA algorithm description is mainly depicted because it is very good for computer processing since it is strong and it is less computer demanding than other algorithms.

After that follows implementation of one of the ICA algorithm in C++ programming language. FastICA algorithm for complex valued signal was chosen.

Keywords:

blind source separation, independent component analysis, algorithm, implementation, C++, FastICA, whitening

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Implementace algoritmů slepé separace zdrojů v jazyce C/C++ jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 4. 6. 2008

.....
(podpis autora)

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Ivanu Míčovi, doktorandovi Ústavu Telekomunikací, za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne 4. 6. 2008

.....
(podpis autora)

Obsah

Seznam použitých symbolů

Symbol	Popis
$\mathbf{A}$	směšující matice s prvky a_{ij}
$p_x(x)$	hustota pravděpodobnosti náhodné proměnné x
$p_x(\mathbf{x})$	hustota pravděpodobnosti n-rozměrného náhodného vektoru $\mathbf{x}$
$E\{x\}$	střední hodnota náhodné proměnné x
$\mathbf{C_x}$	kovariační matice $\mathbf{x}$
$\det(\mathbf{A})$	determinant matice $\mathbf{A}$
$\mathbf{m_x}$	vektor středních hodnot vektoru $\mathbf{x}$
m_x	střední hodnota náhodné veličiny x
σ^2	rozptyl
x_k	částečný součet řady $\{z_i\}$ nezávislých náhodných veličin
a_{ij}	koeficienty směšující matice
s_j	zdrojové signály
α_j	j-tý moment náhodné veličiny x
μ_j	j-tý centrální moment náhodné veličiny x
$\varphi(\omega)$	charakteristická funkce náhodné veličiny x
$\phi(\omega)$	druhá charakteristická funkce náhodné veličiny x
$\ln(x)$	přirozený logaritmus veličiny x
κ_k	k-tý kumulant
$kurt(x)$	koeficient špičatosti
$\tilde{\kappa}(x)$	normovaná špičatost (kumulant 4. řádu)
$\mathcal{J}(\mathbf{w})$	účelová funkce vektoru $\mathbf{w}$
$H(X)$	entropie diskrétní náhodné proměnné X
$P(X = a_i)$	hustota pravděpodobnosti náhodné proměnné X v bodě a_i
$\log(x)$	dekadický logaritmus proměnné x
$H(x)$	diferenciální entropie
$J(\mathbf{x})$	negentropie náhodného vektoru $\mathbf{x}$
$\mathbf{x}_{gauss}$	náhodný vektor s gaussovským rozdělením
F^i	zvolená lineárně nezávislá funkce
y_1	první hlavní komponent
$J_1^{PCA}(\mathbf{w}_1)$	kritérium prvního hlavního komponentu
$\mathbf{e}_n$	n-tý vlastní vektor
δ_{ij}	Kroneckerovo delta
$\mathbf{I}$	jednotková matice
$\mathbf{V}$	bělicí matice
$\mathbf{E}$	matice, jejíž sloupce jsou normované vlastní vektory kovariační matice
$\mathbf{D}$	diagonální matice vlastních hodnot kovariační matice
$x_i(t)$	směs zdrojových signálů
$s_1(t)$	zdrojový signál
a_{ij}	parametr smíchání
$\mathbf{x}$	náhodný vektor, jehož prvky jsou směsi signálů
$\mathbf{s}$	náhodný vektor, jehož prvky jsou zdrojové signály
$\tilde{\mathbf{A}}$	směšující matice $\mathbf{A}$, zprava vynásobená bělicí maticí $\mathbf{V}$
$\mathbf{A}^{-1}$	matice inverzní k směšující matici $\mathbf{A}$
$\mathbf{b}$	vektor k nalezení nalezení nezávislého komponentu
$\mathbf{q}$	vektor značící $\mathbf{b}^T\mathbf{A}$

w	vektor pro maximalizaci absolutní hodnoty špičatosti
sign(x)	znaménková funkce
z	vybělený vektor **x**
$\propto$	výraz na levé straně je v přímé úměrnosti výrazu na pravé straně
$\leftarrow$	symbol přiřazení
g	derivace vybrané aktivační funkce
γ	$\gamma = E\left\{G\left(\mathbf{w}^T\mathbf{z}\right)\right\} - E\left\{G(\nu)\right\}$
W	matice obsahující vektory **w**
B	matice inverzní k směšující matici **A**
$\tilde{p}_i^+$	hustota pravděpodobnosti se supergaussovským průběhem
$\tilde{p}_i^-$	hustota pravděpodobnosti se subgaussovským průběhem
n	vektor šumu
$\tilde{\mathbf{s}}$	vektor zašuměných zdrojových signálů

Seznam použitých zkratek

BSS	Blind source separation (Slepá separace zdrojů)
FastICA	Fast Fixed Point Algorithm for ICA (Algoritmus ICA s pevným bodem)
HOS	Higher Order Statistics (Statistika vyšších řádů)
ICA	Independent Component Analysis (Analýza nezávislých komponentů)
IDE	Integrated Development Environment (Integrované vývojové prostředí)
MMSE	Minimum Mean-Square Error (Minimalizace střední kvadratické hodnoty chyby)
PAST	Projection Approximation Subspace Tracking
PCA	Principal Component Analysis (Analýza hlavních komponentů)
VS	Microsoft Visual Studio

1 Úvod

Slepá separace zdrojů je souhrn metod, které jsou určeny pro oddělení směsí signálů. Tyto metody se nazývají slepé, protože o zdrojových signálech, které jsou součástí směsí, nemáme před jejich separací žádné nebo minimální poznatky. Mezi příklady těchto směsí je možné zahrnout směsi audiosignálů různých zdrojů (řečníků, hudebníků), měření EEG několika senzory, obrazová data, nebo také finanční data.

Tyto směsi signálů mohou být mixovány lineárně i nelineárně. Pro jednoduchost budeme uvažovat lineární model, protože i tento se dá využít pro reálná data. V obecné rovině jsou signály zaznamenávány s časovými prodlevami, které ale také nebudou brány v potaz. Kromě jednoduchého lineárního modelu se v práci zaměříme pouze na výskyt šumu ve směsích signálů, který nám stíží oddělení jednotlivých signálů. Dále je obtížné tento šum odstranit z jednotlivých oddělených signálů.

BSS je soubor několika metod, avšak jako nejvýznamnější se ukazují metody Independent Component Analysis (ICA, analýza nezávislých komponentů), a Principal Component Anylysis (PCA, analýza hlavních komponent). Proto budou brány na zřetel hlavně metody ICA, které jsou adaptovatelné na většinu reálných situací a jsou také rychlé. Metoda PCA bude jednou z hlavních částí předzpracování signálových směsí.

Jelikož jsou metody ICA založeny na statistice vyšších řádů, entropii a dalších principech, bude jedna část věnována matematickému aparátu, který poslouží jako základ pro další kapitoly. Ty budou věnovány samotným různým algoritmům ICA, kde budou v rámci možností této práce vysvětleny jejich hlavní podstaty, s důrazem na budoucí implementaci v programovacím jazyce.

Bude také uveden příklad implementace jednoho z algoritmů ICA s využitím pomocných knihoven, které usnadní tuto implementaci. Tato bude poté konfrontována s implementací v programovém prostředí Matlab.

2 Matematický aparát

V této části budou ukázány základní principy, které slouží pro zpracování signálů metodami ICA. Bude zaveden pojem náhodný vektor, hustota pravděpodobnosti, korelace, kovariance, šikmost, špičatost, entropie, negentropie a vzájemná informace. Dále bude naznačena metoda gradientní aproximace.

Tato kapitola si neklade za cíl vyložit pojmy do hloubky, nýbrž pouze seznámit s uvedenými pojmy a základními vztahy mezi nimi.

Na obsahu této kapitoly bude stavět zejména kapitola 4, která se na mnohé vztahy zde uvedené bude odkazovat a využívat je pro odvození algoritmů zejména analýzy nezávislých komponent.

2.1 Statistická nezávislost

Statistická nezávislost je klíčová vlastnost při analýze nezávislých komponent. Pro jednoduchost mějme dvě rozdílné náhodné proměnné *x* a *y*. Náhodná proměnna *x* je nezávislá na *y*, pokud nám znalost hodnoty *y* neříká nic o hodnotě *x*. Tyto dvě proměnné mohou být například výsledkem dvou různých fyzikálních jevů, které spolu vůbec nesouvisí. Příkladem může být házení kostkou a mincí, nebo řečový signál a šum okolí.

Matematicky je nezávislost vyjádřena pomocí hustot pravděpodobností a dvě náhodné proměnné jsou považovány za nezávislé, právě když platí

$$p_{x,y}(x,y) = p_x(x)\,p_y(y) \tag{2.1}$$

Dále nezávislé proměnné splňují základní vlastnost

$$E\{g(x)h(y)\} = E\{g(x)\}E\{h(y)\}, \tag{2.2}$$

kde $g(x)$ a $h(y)$ jsou dvě absolutně integrovatelné funkce x respektive y.

2.2 Vícerozměrné Gaussovo rozdělení pravděpodobnosti

Uvažujme n-rozměrný náhodný vektor $\mathbf{x}$. Tento bude považován za Gaussův, pokud jeho rozdělení pravděpodobnosti bude mít tvar

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\left(\det \mathbf{C}_{\mathbf{x}}\right)^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m}_{\mathbf{x}})^T \mathbf{C}_{\mathbf{x}}^{-1}(\mathbf{x}-\mathbf{m}_{\mathbf{x}})\right)}, \tag{2.3}$$

ve kterém n vyjadřuje rozměr $\mathbf{x}$, $\mathbf{m}_{\mathbf{x}}$ jeho střední vektor a $\mathbf{C}_{\mathbf{x}}$ kovariační matici $\mathbf{x}$. Pokud zvolíme $n = 1$, (2.3) se zjednoduší na

$$p_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x-m)^2}{2\sigma^2}\right)}, \tag{2.4}$$

což je známé vyjádření normálního rozdělení pravděpodobnosti pro jednu proměnnou, ve které m vyjadřuje střední hodnotu a σ^2 rozptyl.

Z některých vlastností normálního rozdělení můžeme uvést:

1. Pro popis normálního rozdělení nám stačí znalost kovariační matice $\mathbf{C}_{\mathbf{x}}$ a středního vektoru $\mathbf{m}_{\mathbf{x}}$.
2. Lineární transformace gaussových náhodných vektorů jsou zase Gaussovy náhodné vektory, což má za následek nemožnost oddělit signály s Gaussovým rozdělením ze směsí signálů pomocí klasických metod ICA, pokud o těchto signálech nemáme další informace.

Centrální limitní věta

Nechť

$$x_k = \sum_{i=1}^{k} z_i \tag{2.5}$$

je částečný součet řady $\{z_i\}$ nezávislých náhodných veličin z_i se stejným rozdělením pravděpodobnosti. Může být dokázáno (například [2]), že pokud $k \to \infty$, tak rozdělení x_k se blíží rozdělení normálnímu.

Centrální limitní věta je pro analýzu nezávislých komponent, potažmo pro slepou separaci zdrojů, velmi důležitá, protože typickým příkladem směsi vektoru **x** je prvek tvaru

$$x_i = \sum_{j=1}^{m} a_{ij} s_j \,, \tag{2.6}$$

kde a_{ij}, $j = 1, \ldots m$ jsou konstantní směšující koeficienty a s_j, $j = 1, \ldots m$ jsou neznámé zdrojové signály, a třebaže máme relativně malý počet těchto směsí (například $m = 15$), tak jejich suma se blíží normálnímu rozdělení. Toto platí i pro různé zdrojové signály, jejichž rozdělení pravděpodobnosti nejsou stejná.

2.3 Statistiky vyšších řádů

Pro popis náhodných proměnných s jiným rozdělením než je normální rozdělení, musíme zavést statistiku vyšších řádů (*HOS – Higher Order Statistics*). Tato je nutná pro řešení problémů analýzy nezávislých komponent. Dále pro jednoduchost budeme uvažovat pouze skalární náhodné veličiny.

Definujme j-tý moment α_j náhodné veličiny x jako

$$\alpha_j = E\left\{x^j\right\} = \int_{-\infty}^{\infty} x^j p_x(x) dx, \; pro \; j = 1, 2, \ldots \tag{2.7}$$

a j-tý centrální moment μ_j náhodné veličiny x jako

$$\mu_j = E\left\{\left(x - \alpha_1\right)^j\right\}, \; pro \; j = 1, 2, \ldots, \tag{2.8}$$

takže centrální momenty jsou vypočteny pomocí střední hodnoty m_x veličiny x, která je rovna prvnímu momentu α_1.

Existují distribuce, pro které nejsou momenty konečné a navíc znalost momentů nemusí znamenat určení rozdělení hustoty pravděpodobnosti. Ale většina běžně se vyskytujících distribucí má momenty konečné a jejich znalost odpovídá znalosti hustoty pravděpodobnosti.

Pokud definujeme *charakteristickou funkci* $\varphi(\omega)$ veličiny x jako spojitou Fourierovu transformaci hustoty pravděpodobnosti $p_x(x)$

$$\varphi(\omega) = E\left\{e^{(j\omega x)}\right\} = \int_{-\infty}^{\infty} e^{(j\omega x)} p_x(x) dx \,, \tag{2.9}$$

kde j je imaginární jednotka, můžeme jejím zlogaritmováním dostat druhou charakteristickou funkci $\phi(\omega)$

$$\phi(\omega) = \ln(\varphi(\omega)) = \ln\left(E\left\{e^{(j\omega x)}\right\}\right), \tag{2.10}$$

která generuje *kumulanty*. K-tý kumulant dostaneme jako derivaci

$$\kappa_k = \left(-j\right)^k \left.\frac{d^k \phi(\omega)}{d\omega^k}\right|_{\omega=0} . \tag{2.11}$$

Pro náhodnou veličinu x s nulovou střední hodnotou jsou tyto první 4 kumulanty:

$$\begin{aligned} \kappa_1 &= 0 \\ \kappa_2 &= E\left\{x^2\right\} \\ \kappa_3 &= E\left\{x^3\right\} \\ \kappa_4 &= E\left\{x^4\right\} - 3\left[E\left\{x^2\right\}\right]^2 \end{aligned} \quad , \tag{2.12}$$

takže první 3 kumulanty jsou rovny odpovídajícím obecným momentům.

Třetí centrální moment

$$\mu_3 = E\left\{\left(x - m_x\right)^3\right\} \tag{2.13}$$

se nazývá *koeficient šikmosti* a je důležitý pro vyjádření asymetrie hustoty pravděpodobnosti. Pro nulový μ_3 je hustota pravděpodobnosti symetrická.

Dále uvažujme momenty čtvrtého řádu. Momenty vyšších řádů se používají zřídka, takže se jimi nebudeme zabývat. Obecný moment čtvrtého řádu $\alpha_4 = E\left\{x^4\right\}$ se pro svou jednoduchost používá v některých algoritmech ICA. Na rozdíl od 4. centrálního momentu se používá koeficient *špičatosti* (z anglického kurtosis, značeno *kurt*(x)), který je jedním ze základních stavebních kamenů analýzy nezávislých komponent.

Špičatost je rovna 4. kumulantu z (2.12) a může být vyjádřena i jako normovaná

$$\tilde{\kappa}(x) = \frac{E\left\{x^4\right\}}{\left[E\left\{x^2\right\}\right]^2} - 3 \tag{2.14}$$

a pro vybělená data, kde $E\left\{x^2\right\} = 1$ se obě špičatosti zjednoduší na

$$kurt(x) = \tilde{\kappa}(x) = E\left\{x^4\right\} - 3, \tag{2.15}$$

takže pro vybělená data můžeme místo špičatosti použít 4. obecný moment $E\left\{x^4\right\}$.

Pro dvě nezávislé náhodné veličiny x a y platí, že špičatost jejich součtu je rovna součtu jejich špičatostí, zapsáno jako

$$kurt(x + y) = kurt(x) + kurt(y). \tag{2.16}$$

Koeficient špičatosti umožňuje posoudit, jak se liší průběh hustoty pravděpodobnosti náhodné veličiny od průběhu gaussovského. Pokud má x gaussovský průběh (nazýván mesokurtický), pak kurt(x) je rovna 0, pokud je průběh subgaussovský (platykurtický), je kurt(x) menší než 0 a nakonec pokud je průběh supergaussovský (leptokurtický), tak kurt(x) je větší než 0.

2.4 Gradientní metody

Hlavním úkolem ICA je nalézt separační matici **W**, která nám oddělí nezávislé zdroje. Tento úkol nejde vyřešit analyticky, ale pouze numericky zavedením *účelových funkcí*. Poté jsou nalezena řešení **W** v minimech nebo maximech těchto funkcí. Algoritmy řešení těchto funkcí jsou založeny na gradientech účelových funkcí, proto zde bude načrtnuta hrubá představa o gradientech vektorů a matic. Dále zde budou naznačeny typické metody řešení těchto problémů.

Mějme funkci g o m skalárních proměnných

$$g = g(w_1, ..., w_m) = g(\mathbf{w}), \tag{2.17}$$

kde jsme použili označení $\mathbf{w} = (w_1, ..., w_m)^T$ jako sloupcový vektor. Pokud je funkce g diferencovatelná, pak vektorový gradient **w** je m-rozměrný sloupcový vektor parciálních derivací

$$\frac{\partial g}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial g}{\partial w_1} \\ \vdots \\ \frac{\partial g}{\partial w_m} \end{pmatrix}. \tag{2.18}$$

Obdobně mějme funkci g prvků w_{ij} matice **W** s rozměry $m \times n$

$$g = g(\mathbf{W}) = g(w_{11}, ..., w_{ij}, ..., w_{mn}). \tag{2.19}$$

Pak můžeme analogicky zavést gradient matice **W** (můžeme si představit, že matici čteme po řádcích a počítáme vektorové gradienty) jako

$$\frac{\partial g}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial g}{\partial w_{11}} & \cdots & \frac{\partial g}{\partial w_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial g}{\partial w_{m1}} & \vdots & \frac{\partial g}{\partial w_{mn}} \end{pmatrix}. \tag{2.20}$$

Většina metod ICA využívá minimalizaci účelové funkce $\mathcal{J}(\mathbf{W})$, s ohledem na matici **W**. Pokud chceme nalézt minimum vícerozměrné funkce, pak použijeme klasickou metodu největší strmosti. Uvedeme příklad pro funkci vektoru **w**.

V této metodě minimalizujeme účelovou funkci $\mathcal{J}(\mathbf{w})$ po krocích s tím, že začínáme v některém počátečním bodě, vypočítáme gradient $\mathcal{J}(\mathbf{w})$ v tomto bodě a přesuneme řešení ve směru tohoto gradientu do vhodné vzdálenost. V tomto bodě opakujeme výpočet k novému bodu, až nalezneme minimum. Pro $t = 1,2\ldots$, dostáváme iteraci tvaru

$$\mathbf{w}(t) = \mathbf{w}(t-1) - \alpha(t) \frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}(t-1)}, \tag{2.21}$$

gradientu vypočítaného v bodě $\mathbf{w}(t\text{-}1)$. Hodnota $\alpha(t)$ je rovna délce kroku ve směru největší strmosti. V iteraci pokračujeme, dokud algoritmus nekonverguje, což znamená, že vzdálenost dvou po sobě jdoucích řešení je menší než předem určené číslo. Tato metoda je velmi obdobná pro matici **W**.

2.5 Entropie

Entropie je základním prvkem informační teorie. Značíme ji H a je definována pro diskrétní náhodnou proměnnou X jako

$$H(X) = -\sum_i P(X = a_i) \log P(X = a_i), \tag{2.22}$$

kde a_i jsou možné hodnoty X. Na základu logaritmu záleží jednotka entropie, většinou se používá základ 2, poté se jednotka nazývá bit.

Entropie vyjadřuje míru informace, kterou nám náhodná veličina poskytuje. Entropie je tím větší, čím víc je pozorovaná veličina náhodná.

Definice diskrétní entropie může být rozšířena pro spojité veličiny, nazývá se diferenciální entropie. Tato je definována pro náhodnou proměnnou x s hustotou pravděpodobnosti $p_x(x)$ jako

$$H(x) = -\int p_x(x) \log p_x dx\,, \tag{2.23}$$

která může být označena za míru náhodnosti stejně jako entropie. Diferenciální entropie může být také na rozdíl od entropie záporná a může nabývat velkých absolutních hodnot.

Dále je diferenciální entropie invariantní na měřítku

$$H(\alpha x) = H(x) + \log|\alpha|\,. \tag{2.24}$$

2.6 Negentropie

Jelikož gaussovské proměnné mají největší entropii ze všech náhodných proměnných stejného rozptylu, může být entropie měřítkem odchylky od normálního rozložení (*nongaussianity*). Toto může být rozšířeno i na vícerozměrné prostory, takže normální distribuce má největší entropii ze všech distribucí se stejnou kovariační maticí.

Měřítkem této odchylky od gaussovského rozdělení, které je nulové pro gaussovské rozdělení a pro jiná rozdělení je vždy nezáporné, může být negentropie. Tato je definována následovně

$$J(\mathbf{x}) = H(\mathbf{x}_{gauss}) - H(\mathbf{x})\,, \tag{2.25}$$

ve které je $\mathbf{x}_{\text{gauss}}$ náhodný vektor s gaussovským rozdělením a stejnou kovariační maticí jako **x.**

Jelikož je výpočet negentropie výpočetně náročný, zůstává použití negentropie jako měřítka odchylky od normálního rozdělení pouze v teoretické rovině. Snažíme se proto nalézt vhodnou aproximaci této vlastnosti, kterou může být provedena například pomocí kumulantů. V [1] je uveden postup této aproximace, založený na Gram-Charlierově rozvoji, jejímž výsledkem je

$$J(x) \approx \frac{1}{12} E\left\{x^3\right\}^2 + \frac{1}{48} kurt(x)^2\,. \tag{2.26}$$

Jelikož jsou však kumulanty vyšších řádů hodně náchylné na chybné hodnoty v dané množině pozorování a nezachycují dobře chování distribuce funkce kolem nulové hodnoty, nemusí být tato aproximace příliš vhodná.

Jako vhodnější metoda může být použita aproximace negentropie pomocí nepolynomických funkcí. Jak je uvedeno v [1], tato aproximace má tvar

$$J(x) \approx \frac{1}{2} \sum_{i=1}^{n} E\left\{F^i(x)\right\}^2\,, \tag{2.27}$$

ve kterém je *i* index namísto exponentu a F^i jsou vhodně zvolené lineárně nezávislé funkce. Přestože i tato aproximace není zcela přesná, pro gaussovské distribuce je rovna nule.

Pokud jako speciální případ za aktivační funkce zvolíme jednu sudou a druhou lichou funkci, (2.27) přejde na tvar

$$J(x) \approx k_1(E\left\{G^1(x)\right\})^2 + k_2\left(E\left\{G^2\left(x\right)\right\}\right) - E\left\{G^2\left(\nu\right)\right\}^2\,, \tag{2.28}$$

kde zvolením $G^1(x) = x^3$ a $G^2(x) = x^4$ dostaneme stejnou aproximaci, jako v (2.26). Jako vhodné funkce, které dosahují vysoké robustnosti, mohou být použity tyto $G^1(x) = \frac{1}{a}\log\cosh(ax)$ a $G^2(x) = e^{\left(-\frac{x^2}{2}\right)}$, kde *a* leží v intervalu $\langle 1, 2\rangle$.

3 Metody předzpracování

V této části bude představena analýza hlavních komponent (PCA – Principal Component Analysis) jako metoda předzpracování před samotnou ICA. PCA snižuje nadbytečnost proměnných a umožňuje tak použít menší počet proměnných pro vyjádření co nejvíce informace o původní množině. Tato nadbytečnost je v modelu PCA vyjádřena jako vzájemná korelace mezi prvky dat a k jejímu výpočtu je potřeba pouze statistiky 2. řádu.

Jako další metoda bude uvedena metoda bělení (whitening), jež je velmi důležitá metoda předzpracování před ICA. Bělení, jak bude ukázáno, nám sníží nutný počet prvků směšující matice k odhadnutí, což má za následek rychlejší algoritmy ICA.

3.1 Základní principy PCA

Předpokladem pro provedení PCA je, že prvky zdrojového vektoru **x** jsou navzájem korelované, protože kdyby byly statisticky nezávislé, metody PCA by s těmito signály neprovedly nic. Dále musíme před provedením PCA tento náhodný vektor **x** zbavit střední hodnoty, takže $E\{\mathbf{x}\}=0$. Poté už můžeme provést lineární transformaci vektoru **x** na vektor **y**, ve kterém už není obsažena nadbytečnost způsobená korelací mezi prvky vektoru **x.**

Z těchto důvodů může být metod PCA využito například v komprimaci obrazových dat, kde je velmi mnoho obrazových prvků, které jsou spolu úzce korelované a proto je možné dosáhnout vysoké úspory datového toku.

Jelikož je metoda PCA založena na vzájemné korelaci/kovarianci prvků a jejím výsledkem jsou prvky s největšími rozptyly, využívá pouze metody statistiky 2. řádu, a proto nemůže být použita k oddělení jednotlivých zdrojových prvků.

Mezi možné metody PCA patří například metoda maximalizace rozptylu, minimalizace střední kvadratické hodnoty chyby (Minimum Mean-Square Error, MMSE), PAST (Projection Approximation Subspace Tracking) algoritmus, avšak pro příklad bude uvedena pouze metoda první, ve které se užívá vlastních hodnot kovariační matice.

3.2 Metoda maximalizace rozptylu

Uvažujme lineární kombinaci prvků $x_1,\ldots,x_n$ vektoru **x** jako

$$y_1 = \sum_{k=1}^{n} w_{k1} x_k = \mathbf{w}_1^T \mathbf{x},$$

kde $w_{11},\ldots w_{n1}$ jsou váhové koeficienty n-rozměrného vektoru $\mathbf{w}_1$. Potom můžeme proměnnou y_1 považovat za první hlavní komponent tehdy, pokud je její rozptyl maximální. Jelikož rozptyl závisí na velikosti (normě) a směru vektoru **w**, a může neomezeně růst, pak omezíme velikost **w** na konstantní délku (zpravidla rovnu 1). Poté již hledáme kritérium, které nám maximalizuje rozptyl

$$J_1^{PCA}(\mathbf{w}_1) = E\{y_1^2\} = E\left\{\left(\mathbf{w}_1^T\mathbf{x}\right)^2\right\} = \mathbf{w}_1^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w}_1 = \mathbf{w}_1^T\mathbf{C}_\mathbf{x}\mathbf{w}_1, \tag{3.1}$$

tak, že $\|\mathbf{w}_1\| = 1$.

Matice $\mathbf{C}_\mathbf{x}$ v rov. 3.1 je kovariační matice rozměrů $n \times n$ vektoru **x**, která je rovna korelační matici vektoru **x** s nulovou střední hodnotou

$$\mathbf{C}_\mathbf{x} = E\{\mathbf{x}\mathbf{x}^T\}. \tag{3.2}$$

Řešením tohoto problému jsou vlastní vektory $\mathbf{e}_1,\ldots,\mathbf{e}_n$ matice $\mathbf{C_x}$. Pořadí vlastních vektorů je takové, že příslušné vlastní hodnoty jsou seřazené sestupně podle odpovídajících vlastních čísel. Řešení maximalizující (3.1) je dáno

$$\mathbf{w}_1 = \mathbf{e}_1, \tag{3.3}$$

takže první hlavní komponent **x** je $y_1 = \mathbf{e}_1^T\mathbf{x}$.

Pak hledáme další hlavní komponenty tak, že je každý nekorelovaný s již nalezenými. Řešením je

$$\mathbf{w}_k = \mathbf{e}_k, \tag{3.4}$$

které nám určí *k*-tý hlavní komponent jako

$$y_k = \mathbf{e}_k^T\mathbf{x}. \tag{3.5}$$

3.3 Bělení

Bělení (z anglického whitening) nám značným způsobem usnadňuje řešení ICA. Proto, než přistoupíme k samotnému řešení ICA, provedeme bělení zdrojových signálů.

Náhodný vektor **z** s nulovou střední hodnotou je bílý, pokud jsou jeho složky z_i nekorelovány a mají jednotkový rozptyl $E\{z_i z_j\} = \delta_{ij}$ [1], což nám u kovariační matice dává $E\{\mathbf{z}\mathbf{z}^T\} = \mathbf{I}$, kde **I** je jednotková matice.

Nejvhodnějším příkladem by mohl být bílý šum, u kterého není žádná časová závislost jednotlivých složek šumu.

Jelikož je bělení v podstatě dekorelace s úpravami měřítek, můžou být s výhodou použity metody PCA. Z tohoto plyne, že bělení je vlastně lineární operace. Pokud máme náhodný vektor **x**, naším úkolem je najít takovou lineární transformaci **V**, že vektor $\mathbf{z} = \mathbf{V}\mathbf{x}$ je vektorem bílým.

Když budeme mít matici $\mathbf{E} = (\mathbf{e}_1 \ldots \mathbf{e}_n)$, kde jsou její sloupce normované vlastní vektory kovariační matice $\mathbf{C_x} = E\{\mathbf{x}\mathbf{x}^T\}$ a dále budeme mít diagonální matici $\mathbf{D} = diag(d_1 \ldots d_n)$ vlastních hodnot matice $\mathbf{C_x}$, pak transformace lineárního bělení má tvar

$$\mathbf{V} = \mathbf{D}^{-1/2}\mathbf{E}^T, \tag{3.6}$$

kde kovariační matici $\mathbf{C_x}$ můžeme přepsat ve tvaru $\mathbf{C_x} = \mathbf{E}\mathbf{D}\mathbf{E}^T$, s ortogonální maticí[2] **E**. Pak kovariance **z**

$$E\{\mathbf{z}\mathbf{z}^T\} = \mathbf{V}E\{\mathbf{x}\mathbf{x}^T\}\mathbf{V}^T = \mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{E}\mathbf{D}\mathbf{E}^T\mathbf{E}\mathbf{D}^{-1/2} = \mathbf{I} \tag{3.7}$$

je jednotkovou maticí, což znamená, že vektor **z** je vektorem bílým.

Vektor **z** bude také bílý, pokud matici **V** zprava vynásobíme libovolnou ortogonální maticí.

---

[1] δ_{ij} je tzv. Kroneckerovo delta, které udává, jestliže jsou si dvě proměnné (většinou celá čísla) rovny.

$$\delta_{ij}\begin{cases}1, & když\ i = j\\ 0, & když\ i \neq j\end{cases}$$

[2] Ortogonální matice je taková matice, jejíž transpozice je rovna její inverzní matici, takže

$$\mathbf{A}^T\mathbf{A} = \mathbf{A}\mathbf{A}^T = \mathbf{I}$$

4 Analýza nezávislých komponentů

V této části se budeme zabývat samotným problémem analýzy nezávislých komponent (ICA). Využijeme poznatky předešlých kapitol k odvození základních algoritmů pro nalezení zdrojových signálů ze směsí signálů a ukážeme význam metody bělení.

V poslední části bude naznačen postup pro metody ICA pracující se šumem v signálech.

4.1 Základní poznatky k ICA

Představme si, že jsme v místnosti, kde jsou 3 (může být i jiný počet) zdroje zvuků (např. řečník, kytarista, zpěvačka) a tyto vydávají zároveň různé zvuky. Dále máme k dispozici 3 mikrofony na různých místech v místnosti, které nám zaznamenávají 3 směsi zvuků od těchto zvukových zdrojů. Tyto směsi označme $x_1(t)$, $x_2(t)$, $x_3(t)$, kde každá tato směs je lineární kombinací (uvedeno jako zjednodušený případ – v reálné situaci jsou směsi zaznamenány s časovým zpožděním, s odrazy od okolí a se šumem) zdrojů zvuků $s_1(t)$, $s_2(t)$, $s_3(t)$. Toto můžeme zapsat jako lineární rovnice

$$\begin{aligned} x_1(t) &= a_{11}s_1(t) + a_{12}s_2(t) + a_{13}s_3(t) \\ x_2(t) &= a_{21}s_1(t) + a_{22}s_2(t) + a_{23}s_3(t), \\ x_3(t) &= a_{31}s_1(t) + a_{32}s_2(t) + a_{33}s_3(t) \end{aligned} \tag{4.1}$$

kde a_{ij} jsou parametry závisící na vzdálenosti zdrojů zvuků od mikrofonů.

Naším úkolem tedy bude oddělit zdrojové signály $s_i(t)$ pouze ze znalosti směsí $x_i(t)$. Nebudeme brát v úvahu časové prodlevy vzniklé putováním signálů k mikrofonům.

Pokud bychom znali směšující parametry a_{ij}, pak bychom jednoduše vyřešili lineární soustavu rovnic, ale jelikož tyto parametry neznáme, jakožto neznáme ani zdrojové signály, pak je problém mnohem složitější.

Řešení problému ICA vychází ze statistiky zdrojových signálů a toto řešení předpokládá statistickou nezávislost zdrojů, ze které dokážeme určit směšující parametry a_{ij}, které nám poté řeší celý problém ICA.

4.2 Definice ICA

Nyní se pokusíme model ICA popsat obecně. Uvažujme n náhodných proměnných $x_1, \ldots, x_n$, které jsou lineárními kombinacemi n náhodných proměnných $s_1, \ldots, s_n$

$$x_i = a_{i1}s_1 + a_{i2}s_2 + a_{i1}s_1 + \ldots + a_{in}s_n, \tag{4.2}$$

pro všechna $i = 1, \ldots, n$ a kde a_{ij}, pro $i, j = 1, \ldots, n$, jsou reálné koeficienty. Z definice jsou statisticky navzájem nezávislé.

Toto je základní model ICA, kde nezávislé komponenty jsou skryté proměnné, což znamená, že je nemůžeme přímo pozorovat. Navíc koeficienty a_{ij} jsou rovněž neznámé. Vše, co dokážeme pozorovat, jsou náhodné proměnné x_i a pomocí nich musíme určit jak koeficienty a_{ij}, tak zdrojové signály s_i.

Pro zjednodušení zápisu budeme používat maticový a vektorový zápis, kde $\mathbf{x}$ je náhodný vektor, jehož prvky jsou směsi $x_1, \ldots, x_n$ a podobně $\mathbf{s}$ je náhodný vektor s prvky $s_1, \ldots, s_n$. Poté označíme jako $\mathbf{A}$ matici koeficientů a_{ij}. Vektory tu považujeme za sloupcové. Takže můžeme (4.2) přepsat jako

$$\mathbf{x} = \mathbf{A}\mathbf{s}. \tag{4.3}$$

Abychom mohli tento model úspěšně řešit, musíme zavést některá omezení:

1. Nezávislé komponenty považujeme za statisticky nezávislé, toto znamená, že pokud máme množinu náhodných veličin $x_1, x_2, \ldots, x_n$, tak znalost hodnoty jedné nám neříká, jakou hodnotu mají ostatní veličiny. Více v části 2.1.
2. Nezávislé komponenty *nemají* Gaussovu distribuci, nanejvýš jeden komponent má Gaussovu distribuci.
3. Směšující matice je čtvercová (a regulární), což znamená, že máme stejný počet zdrojových signálů, jako senzorů.

Metody ICA nám nepomohou vyřešit následující aspekty:

1. Energie zdrojových signálů.

Jelikož jsou vektor **s** i matice **A** neznámé, pak vynásobení jednoho komponentu číslem můžeme eliminovat vydělením příslušného sloupce matice **A** tímto číslem

$$\mathbf{x} = \sum_i \left(\frac{1}{\alpha_i} \mathbf{a}_i \right) \left(s_i \alpha_i \right). \tag{4.4}$$

Stále ovšem je zachována neurčitelnost znaménka, která ale ve většině případů není podstatná.

2. Nedokážeme rekonstruovat pořadí zdrojových signálů.

Před provedením ICA musíme ještě provést odečtení střední hodnoty z náhodných vektorů směsí, což ale neovlivní výpočet směšující matice, která je stejná jak pro vystředěné, tak pro nevystředěné vzorky. Proto tuto matici vypočítáme pro vystředěné hodnoty a použijeme ji pro nalezení původních zdrojových signálů.

Nyní se pokusíme ukázat, proč je bělení důležitou součástí předzpracování k ICA. Pokud označíme **z** jako výsledek po bělení vektoru **x**, pak můžeme (4.3) přepsat jako

$$\mathbf{z} = \mathbf{VAs} = \tilde{\mathbf{A}}\mathbf{s}, \tag{4.5}$$

kde **V** je bělicí matice a $\tilde{\mathbf{A}}$ je bělením změněná směšující matice. Tato je maticí ortogonální. Z toho vyplývá, že místo hledání n^2 prvků původní směšující matice **A** hledáme pouze $n(n-1)/2$ prvků matice $\tilde{\mathbf{A}}$, což znamená, že bělení nám usnadní řešení ICA o (přibližně) polovinu, protože operace bělení je mnohem jednodušší než ICA.

Pokud jde o zdrojové signály s gaussovskou distribucí, pak pouze tyto nedokážeme ze směsí oddělit, zbudou nám proto jejich lineární kombinace. Pokud máme pouze jeden gaussovský zdroj, pak tento není s žádným jiným kombinovaný, což vede k vyřešení celé ICA.

4.3 ICA pomocí maximalizace odlišnosti od gaussovského rozdělení

Jako měřítko odlišnosti od gaussovského rozložení nám v této kapitole poslouží kumulant 4.řádu – špičatost a dále také negentropie, oba pojmy byly vysvětleny v kap. (2.3) resp. (2.5).

Tento soubor algoritmů využívá poznatků centrální limitní věty (viz kap. 2.2), která říká, že rozložení pravděpodobnosti součtu nezávislých náhodných veličin se blíží ke gaussovskému rozdělení s rostoucím počtem sčítaných veličin.

Uvažujme vektor splňující pravidla ICA

$$\mathbf{x} = \mathbf{As},$$

pak odhad nezávislých komponent je dán nalezením inverzní matice k matici **A**

$$\mathbf{s} = \mathbf{A}^{-1}\mathbf{x}. \tag{4.6}$$

Označíme jeden odhad nezávislého komponentu jako $y = \mathbf{b}^T\mathbf{x} = \sum_i b_i x_i$, kde **b** je vektor, který máme nalézt. Můžeme také psát $y = \mathbf{b}^T\mathbf{A}\mathbf{s}$, pak *y* je lineární kombinací s_i, dána koeficienty $\mathbf{b}^T\mathbf{A}$. Tyto označíme jako **q**. Pak dostáváme

$$y = \mathbf{b}^T\mathbf{x} = \mathbf{q}^T\mathbf{s} = \sum_i q_i s_i \,. \tag{4.7}$$

Pokud je **b** rovno jednomu z řádků inverzní matice k **A**, pak je lineární kombinace $\mathbf{b}^T\mathbf{x}$ rovna jednomu z nezávislých komponent. Poté je **q** takové, že všechny jeho prvky jsou nulové, pouze jediný je roven 1.

Hledáme tedy vektor **b** takový, který maximalizuje odlišnost rozdělení $\mathbf{b}^T\mathbf{x}$ od rozdělení gaussovského, čímž dostaneme jeden nezávislý komponent.

V *n*-rozměrném prostoru vektoru **b** máme 2*n* lokálních maxim odlišnosti od gaussovského rozdělení – 2 pro každý nezávislý komponent (s_i a $-s_i$, protože nedokážeme určit znaménko rekonstruovaného komponentu).

Jako první měřítko odlišnosti od gaussovského rozdělení použijeme špičatost z kap. (2.4). Pro připomenutí, špičatost náhodné proměnné y je definována jako

$$kurt(y) = E\left\{y^4\right\} - 3\left(E\left\{y^2\right\}\right)^2, \tag{4.8}$$

kde dále předpokládáme, že y je normalizovaná, takže má jednotkový rozptyl, takže se nám špičatost zjednoduší na $kurt(y) = E\left\{y^4\right\} - 3$. Špičatost je nulová pro gaussovské rozdělení a pro téměř všechna jiná rozdělení je různá od nuly. Proto se budeme snažit maximalizovat absolutní hodnotu špičatosti. Tohoto dosáhneme pomocí vektoru **w**, pro který bude mít lineární kombinace $\mathbf{w}^T\mathbf{x}$ největší absolutní hodnotu špičatosti.

V kap. (2.4) je uveden náčrt gradientní metody, kterou budeme používat. Zvolíme některý vektor **w**, spočítáme gradient, ve kterém špičatost roste nejvíce, a v tomto směru tento vektor posuneme.

Pro vybělený vektor **z** můžeme absolutní špičatost $\mathbf{w}^T\mathbf{z}$ vypočítat jako

$$\frac{\partial\left|kurt\left(\mathbf{w}^T\mathbf{z}\right)\right|}{\partial\mathbf{w}} = 4sign\left(kurt\left(\mathbf{w}^T\mathbf{z}\right)\right)\left[E\left\{\mathbf{z}\left(\mathbf{w}^T\mathbf{z}\right)^3\right\} - 3\mathbf{w}\left\|\mathbf{w}\right\|^2\right], \tag{4.9}$$

a protože vektor **w** normujeme a zajímá nás hlavně jeho směr, pak můžeme algoritmus zjednodušit na

$$\begin{aligned} \Delta\mathbf{w} &\propto sign\left(kurt\left(\mathbf{w}^T\mathbf{z}\right)\right)E\left\{\mathbf{z}\left(\mathbf{w}^T\mathbf{z}\right)^3\right\} \\ \mathbf{w} &\leftarrow \frac{\mathbf{w}}{\left\|\mathbf{w}\right\|} \end{aligned}, \tag{4.10}$$

kde pokud známe dopředu povahu rozdělení nezávislých komponentů, nemusíme počítat znaménko špičatosti.

Tato gradientní metoda je vhodná do učících se algoritmů (také on-line algoritmy), kdy je schopná se adaptovat na změny povah příchozích dat. Nevýhodou je pomalá konvergence.

Pokud vektoru **w** namísto posouvání měníme hodnotu podle

$$\mathbf{w} \leftarrow E\left\{\mathbf{z}\left(\mathbf{w}^T\mathbf{z}\right)^3\right\} - 3\mathbf{w}\,, \tag{4.11}$$

pak dostáváme algoritmus s pevným bodem (FastICA, fixed-point) pro maximalizaci špičatosti. Po každém kroku musí být w normován a konvergence je dosaženo tehdy, když vektorový součin posledních dvou kroků je roven (téměř) 1. Konvergence u tohoto

typu algoritmu je velmi rychlá (kubická). Tento algoritmus je také mnohem jednodušší, než gradientní metoda.

Pro měření odlišnosti od gaussovského rozložení je také určena negentropie, která je i vhodnější, protože špičatost je náchylná na hodnoty, které se vymykají průměrných hodnotám ve veličině. Negentropie je tedy robustnější, zato výpočetně náročnější. Proto se budeme nakonec snažit obě metody zkombinovat v jednu.

Teorie k negentropii je obsažena v kap. (2.6). Negentropii můžeme aproximovat pomocí

$$J(y) \approx \frac{1}{12} E\left\{y^3\right\}^2 + \frac{1}{48} kurt(y)^2, \tag{4.12}$$

avšak tato aproximace pro symetrická rozdělení trpí stejnými vlastnostmi, jako algoritmy využívající pouze špičatost.

Proto zvolíme aproximaci podle 2.28, v upraveném tvaru pro jednu aktivační funkci

$$J(y) \propto \left[E\left\{G(y)\right\} - E\left\{G(\nu)\right\}\right]^2, \tag{4.13}$$

ve které je ν standardizovaná gaussovská proměnná a *G(x)* je aktivační funkce.

Poté už můžeme dostat gradientní algoritmus využívající negentropii ve tvaru

$$\begin{aligned} \Delta \mathbf{w} &\propto \gamma E\left\{\mathbf{z} g\left(\mathbf{w}^T \mathbf{z}\right)\right\} \\ \mathbf{w} &\leftarrow \frac{\mathbf{w}}{\|\mathbf{w}\|} \end{aligned}, \tag{4.14}$$

kde $\gamma = E\left\{G\left(\mathbf{w}^T \mathbf{z}\right)\right\} - E\left\{G(\nu)\right\}$ a g je derivace vybrané aktivační funkce. Příklady vhodných funkcí g jsou

$$\begin{aligned} g_1(y) &= \tanh(ay) \\ g_2(y) &= y e^{-\frac{y^2}{2}} \\ g_3(y) &= y^3 \end{aligned} \tag{4.15}$$

pro $a \in \langle 1, 2 \rangle$, často $a = 1$.

FastICA algoritmus využívající aproximace negentropie má tvar

$$\mathbf{w} \leftarrow E\left\{\mathbf{z} g\left(\mathbf{w}^T \mathbf{z}\right)\right\} - E\left\{g'\left(\mathbf{w}^T \mathbf{z}\right)\right\} \mathbf{w}. \tag{4.16}$$

Nejprve zvolíme funkci g, například z rov. 4.15 a provedeme krok pomocí rov. 4.16, po kterém normujeme vektor **w**. Funkce g' jsou derivace funkcí g

$$\begin{aligned} g_1'(y) &= a(1 - \tanh^2(ay)) \\ g_2'(y) &= (1 - y^2) e^{-\frac{y^2}{2}} \\ g_3'(y) &= 3y^2 \end{aligned} \tag{4.17}$$

.

Konvergence algoritmu FastICA je znovu mnohem rychlejší, než lineární konvergence gradientní metody.

Všechny doposud uvedené algoritmy nám naleznou pouze jeden nezávislý komponent. Pro nalezení dalších musíme tyto algoritmy opakovat, ale abychom zabránili konvergenci do již nalezených maxim, využijeme vlastnosti nezávislých komponent ve vybělením prostoru – ortogonalitu.

Proto hledaný soubor vektorů $\mathbf{w}_1, \ldots, \mathbf{w}_n$ po každém kroku iterace algoritmů ortogonalizujeme.

Tuto ortogonalizaci lze provést např. pomocí dvou metod. První je Gram-Schmidtova ortogonalizace a druhá je symetrická ortogonalizace.

Gram-Schmidtova ortogonalizace spočívá v tom, že po naleznutí p vektorů $\mathbf{w}_1, \ldots, \mathbf{w}_p$, hledáme vektor $\mathbf{w}_{p+1}$ a v každém kroku iterace algoritmu od tohoto vektoru odečítáme průměty $\left(\mathbf{w}_{p+1}^T \mathbf{w}_j\right)\mathbf{w}_j$, *pro* $j = 1, \ldots, p$ dříve nalezených p vektorů a normalizujeme vektor $\mathbf{w}_{p+1}$. Proto jsou nezávislé komponenty určovány po jednom a může dojít ke kumulaci chyby ve výpočtu prvního vektoru do následujících vektorů.

Symetrická ortogonalizace je odlišná v tom, že nezávislé komponenty počítá paralelně a ortogonalizaci provádí až po výpočtu první iterace algoritmu pro všechny komponenty. Využívá ortogonalizační matici $\mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_m)^T$. Tato se určí jako

$$\mathbf{W} \leftarrow \left(\mathbf{W}\mathbf{W}^T\right)^{-\frac{1}{2}} \mathbf{W}, \tag{4.18}$$

kde $(\mathbf{W}\mathbf{W}^T)^{-\frac{1}{2}} = \mathbf{E}\,diag\left(d_1^{-1/2}, \ldots, d_m^{-1/2}\right)\mathbf{E}^T$ je určeno pomocí vlastních hodnot matice $\mathbf{W}$.

Popis kompletního algoritmu FastICA s aproximací negentropie a Gram-Schmidtovou ortogonalizací je uveden v tab. 4.1, stejný algoritmus se symetrickou ortogonalizací v tab. 4.2.

Tab. 4.1 – Kompletní algoritmus FastICA s aproximací negentropie pro určení m nezávislých komponent pomocí Gram-Schmidtovy ortogonalizace.

1. Z dat odečteme střední hodnotu.
2. Vybělíme data, abychom získali vektor **z**.
3. Zvolíme počet nezávislých komponent k určení, označíme m. Nastavíme čítač $p \leftarrow 1$.
4. Zvolíme (třeba náhodně) počáteční hodnotu normovaného vektoru $\mathbf{w}_p$.
5. Provedeme $\mathbf{w} \leftarrow E\left\{\mathbf{z}g\left(\mathbf{w}^T\mathbf{z}\right)\right\} - E\left\{g'\left(\mathbf{w}^T\mathbf{z}\right)\right\}\mathbf{w}$, kde g je např. z rov. 4.15.
6. Provedeme ortogonalizaci $\mathbf{w}_p \leftarrow \mathbf{w}_p - \sum_{j=p}^{p-1}\left(\mathbf{w}_p^T\mathbf{w}_j\right)\mathbf{w}_j$.
7. Znormujeme $\mathbf{w}_p \leftarrow \mathbf{w}_p / \left\|\mathbf{w}_p\right\|$.
8. Pokud $\mathbf{w}_p$ nezkonvergoval, vrátíme se ke kroku 5.
9. Nastavíme $p \leftarrow p+1$, pokud je $p \le m$, vrátíme se ke kroku 4.

Tab. 4.2 – Kompletní algoritmus FastICA s aproximací negentropie pro určení *m* nezávislých komponent pomocí symetrické ortogonalizace.

1. Z dat odečteme střední hodnotu.
2. Vybělíme data, abychom získali vektor **z**.
3. Zvolíme počet nezávislých komponent k určení, označíme *m*.
4. Zvolíme počáteční hodnoty $\mathbf{w}_i$, *pro* $i = 1,...,m$. Znormujeme $\mathbf{w}_i$. Provedeme ortogonalizaci matice **W** z kroku 6.
5. Pro $\forall i$ provedeme $\mathbf{w} \leftarrow E\left\{\mathbf{z}g\left(\mathbf{w}^T\mathbf{z}\right)\right\} - E\left\{g'\left(\mathbf{w}^T\mathbf{z}\right)\right\}\mathbf{w}$, kde *g* je např. z rov. 4.15.
6. Provedeme ortogonalizaci matice $\mathbf{W} = (\mathbf{w}_1,...,\mathbf{w}_m)^T$ pomocí $\mathbf{W} \leftarrow \left(\mathbf{W}\mathbf{W}^T\right)^{-\frac{1}{2}}\mathbf{W}$.
7. Pokud algoritmus nezkonvergoval, vrátíme se ke kroku 5.

4.4 ICA pomocí metody maximální věrohodnosti (Maximum Likelihood)

Tato metoda je založena na určování parametrů, které nám dávají maximální pravděpodobnosti pozorování.

Nejprve určíme hustotu pravděpodobnosti základního modelu ICA jako

$$p_x(\mathbf{x}) = |\det \mathbf{B}|\, p_s(\mathbf{s}) = |\det \mathbf{B}| \prod_i p_i(s_i) = |\det \mathbf{B}| \prod_i p_i(\mathbf{b}_i^T\mathbf{x}), \tag{4.19}$$

kde $\mathbf{B} = (\mathbf{b}_1,...,\mathbf{b}_n)^T$ je maticí inverzní k matici **A** a p_i jsou hustoty pravděpodobností nezávislých komponent.

Poté předpokládáme *T* pozorování veličiny **x** a věrohodnost je rovna součinu hustot pravděpodobností v bodech *T*, což značíme *L*, které je funkcí **B**

$$L(\mathbf{B}) = \prod_{t=1}^{T}\prod_{i=1}^{n} p_i\left(\mathbf{b}_i^T\mathbf{x}(t)\right)|\det \mathbf{B}|. \tag{4.20}$$

Budeme používat logaritmu z rov. 4.20, který bude navíc vyjádřen pomocí střední hodnoty ke zjednodušení zápisu

$$\frac{1}{T}\log L(\mathbf{B}) = E\left\{\sum_{i=1}^{n}\log p_i\left(\mathbf{b}_i^T\mathbf{x}\right)\right\} + \log|\det \mathbf{B}|. \tag{4.21}$$

Bohužel toto vyjádření je velmi obtížné na výpočet, protože musíme určovat hustoty pravděpodobností nezávislých komponentů, což je problém. Naštěstí máme dvě možnosti řešení. První předpokládá určitou předchozí znalost hustot pravděpodobností nezávislých komponent p_i, které jednoduše dosadíme do rov. 4.21. Druhou možností je aproximovat možné hustoty pomocí množiny vhodně zvolených hustot. Tato množina může (a také je) být velmi malá – stačí nám dokonce pouze dvě hustoty pravděpodobností.

Toto řešení je velice vhodné, protože nepřesnosti v aproximaci hustot pravděpodobností nám nevadí, dokud má tato hustota stejný charakter jako hustota pravděpodobnosti nezávislého komponentu.

Můžeme například uvést logaritmické hustoty pravděpodobností

$$\begin{aligned} \log \tilde{p}_i^{+}(s) &= \alpha_1 - 2\log(\cosh(s)) \\ \log \tilde{p}_i^{-}(s) &= \alpha_2 - \left[s^2/2 - \log(\cosh(s))\right], \end{aligned} \tag{4.22}$$

kde α_1, α_2 jsou kladné parametry, které můžeme zanedbat. Hustota $\tilde{p}_i^{+}$ má supergaussovský průběh a naopak $\tilde{p}_i^{-}$ má průběh subgaussovský.

Prvním algoritmem využívající metod maximalizace věrohodnosti je gradientní Bell-Sejnowskeho algoritmus. Tento obdržíme derivováním rov. 4.21 jako

$$\frac{1}{T}\frac{\partial \log L}{\partial \mathbf{B}} = \left[\mathbf{B}^T\right]^{-1} + E\left\{\mathbf{g}\left(\mathbf{Bx}\right)\mathbf{x}^T\right\}, \tag{4.23}$$

ve kterém $\mathbf{g}(\mathbf{y}) = \left(g_i(y_i),...,g_n(y_n)\right)$ je vektorová funkce sestávající z funkcí g_i distribucí s_i

$$g_i = (\log p_i)' = \frac{p_i'}{p_i}. \tag{4.24}$$

Avšak tento algoritmus konverguje velmi pomalu a je výpočetně náročný, protože se musí v každém kroku počítat inverzní matice **B**.

Předchozí algoritmus může být zjednodušen vynásobením pravé strany (4.23) $\mathbf{B}^T\mathbf{B}$. Tímto dostaneme algoritmus přirozeného gradientu

$$\Delta\mathbf{B} \propto \left(\mathbf{I} + E\left\{\mathbf{g}(\mathbf{y})\mathbf{y}^T\right\}\right)\mathbf{B}. \tag{4.25}$$

Tento algoritmus je speciálním případem algoritmů nelineární dekorelace.

Jako nelinearitu g z rov. 4.24 můžeme použít

$$g^+(y) = -2\tanh(y) \tag{4.26}$$

pro supergaussovské nezávislé komponenty, naopak pro subgaussovské komponenty použijeme nelinearitu

$$g^-(y) = \tanh(y) - y. \tag{4.27}$$

Dále uvedeme algoritmus FastICA pro metodu Maximum Likelihood. Tento je vpodstatě úpravou algoritmu FastICA z předešlé kapitoly. Navíc je tento algoritmus výpočetní optimalizací algoritmu přirozeného gradientu a jako nelinearitu g můžeme pro všechny nezávislé komponenty použít hyperbolický tangens.

Základní krok tohoto algoritmu má tvar

$$\mathbf{B} \leftarrow \mathbf{B} + diag(\alpha_i)\left[diag(\beta_i) + E\left\{\mathbf{g}(\mathbf{y})\mathbf{y}^T\right\}\right]\mathbf{B}, \tag{4.28}$$

ve kterém je **y = Bx**, $\beta_i = -E\left\{y_i g(y_i)\right\}$ *a* $\alpha_i = -1/\left(\beta_i + E\left\{g'(y_i)\right\}\right)$.

Po každém kroku je nutné matici **B** dekorelovat a normalizovat

$$\mathbf{B} \leftarrow \left(\mathbf{BCB}^T\right)^{-\frac{1}{2}}\mathbf{B}, \tag{4.29}$$

kde $\mathbf{C} = E\left\{\mathbf{xx}^T\right\}$ je kovariační matice směsí signálů.

Kompletní algoritmus FastICA pro metodu maximalizace věrohodnosti je uveden v tab. 4.3.

Tab. 4.3 – Kompletní algoritmus FastICA pro metodu maximalizace věrohodnosti s počátečním vybělením dat. Jako nelinearita *g* se běžně volí hyperbolický tangens.

1. Z dat odečteme střední hodnotu a vypočteme kovariační matici $\mathbf{C} = E\left\{\mathbf{x}\mathbf{x}^T\right\}$.
2. Vybělíme data, abychom získali vektor **z**.
3. Zvolíme počáteční separační matici **B**.
4. Vypočteme

$$\mathbf{y} = \mathbf{B}\mathbf{z}$$
$$\beta_i = -E\left\{y_i g\left(y_i\right)\right\},\ pro\ i = 1,...,n \quad .$$
$$\alpha_i = -1/\left(\beta_i + E\left\{g'\left(y_i\right)\right\}\right),\ pro\ i = 1,...,n$$

5. Provedeme krok iterace

$$\mathbf{B} \leftarrow \mathbf{B} + diag(\alpha_i)\left[diag(\beta_i) + E\left\{\mathbf{g}(\mathbf{y})\mathbf{y}^T\right\}\right]\mathbf{B}.$$

6. Dekorelujeme a normalizujeme matici B

$$\mathbf{B} \leftarrow \left(\mathbf{B}\mathbf{C}\mathbf{B}^T\right)^{-\frac{1}{2}}\mathbf{B}.$$

7. Pokud algoritmus nezkonvergoval, vrátíme se ke kroku 4.

4.5 Metody ICA pro odstranění šumu ze směsí

V této poslední kapitole se budeme zabývat možnostmi odstranění šumu ze směsí signálů, určením separačních matic pro zašuměné směsi a nakonec také odstraněním šumu z rekonstruovaných nezávislých komponentů.

Nejprve upravíme základní model ICA z rov. 4.3 na

$$\mathbf{x} = \mathbf{A}\mathbf{s} + \mathbf{n}, \tag{4.30}$$

ve kterém je $\mathbf{n} = \left(n_1,...,n_n\right)^T$ vektor šumu. Tento šum je nezávislý na nezávislých komponentech a považujeme jej za šum s Gaussovým rozložením. Kovariační matici šumu označíme $\mathbf{\Sigma}$ a většinou bude mít tvar $\sigma^2\mathbf{I}$ (diagonální matice), což v některých případech nemusí platit, ale většinou předpokládáme znalost kovariační matice.

Pokud je přidaný šum ve tvaru (4.30), pak tento nazýváme šumem senzorovým, protože šum je připočten ke každému senzoru, tzn. ke každému signálu směsi x_i. Na druhou stranu však může být šum přidán k nezávislým komponentům (zdrojům signálů). Takovýto šum poté nazýváme šumem zdrojovým a model ICA můžeme přepsat do podoby

$$\mathbf{x} = \mathbf{A}\left(\mathbf{s} + \mathbf{n}\right). \tag{4.31}$$

Poté považujeme zdroje za zašuměné, což vyjádříme pomocí $\tilde{s}_i = s_i + n_i$ a přepíšeme model ICA na

$$\mathbf{x} = \mathbf{A}\tilde{\mathbf{s}}, \tag{4.32}$$

což je velmi podobné základnímu modelu ICA a poznatky pro bezšumnou ICA zde můžeme normálně použít. Proto není v případě zdrojových šumů obtížné oddělit jednotlivé komponenty – separační matici $\mathbf{A}^{-1}$ nalezneme normálními postupy. Co je však obtížné, je rekonstrukce původních nezašuměných komponentů.

Jiný přístup pro získání originálních nezávislých komponent spočívá v předpokladu relativně malého počtu nezávislých komponent a malého počtu zdrojů šumu. Protože pokud označíme $\tilde{\mathbf{s}} = \left(s_1,...,s_k,n_1,...,n_l\right)^T$, kde s_i pro $i = 1,...,k$ jsou námi

hledané nezávislé komponenty a n_i pro $i = 1,...,l$ jsou šumy, tak pokud je součet $k+l$ roven počtu směsí, které zaznamenáváme, potom tento model znovu vyjádříme pomocí základního modelu ICA. Poté použijeme algoritmus pro získání jednotlivých komponent (nemůžeme například použít metod maximalizace věrohodnosti, protože ty určují všechny komponenty najednou). Tím jsme dospěli k vyřešení problému, pouze se nám nepodaří od sebe oddělit gaussovské šumy, ale to není naším cílem. Bohužel tento postup má omezené použití.

Určení směšující matice

Pokud chceme nalézt směšující matici pro prostředí se šumem, musíme najít vhodné metody, na které nemá šum vliv. Mezi tyto metody patří třeba odstranění vlivu šumu pomocí špičatosti, protože gaussovský šum nemá na průběh špičatosti (vyplývá z definice) vliv. Dále můžeme použít gaussovské momenty jako aktivační funkce v aproximaci negentropie. Mezi další metody patří různé modifikace metody maximalizace věrohodnosti, avšak některé jsou výpočetně velmi náročné.

Pokud bychom měli vhodná data (neobsahují hodnoty vymykající se průměrným hodnotám v daném souboru), lze použít momenty až 6. řádu, avšak díky omezení není tato metoda příliš vhodná.

Odstranění šumu z oddělených signálů

Prosté odhadnutí směšující matice nám k úplnému vyřešení problému nestačí, protože invertováním modelu ICA dostaneme nezávislé komponenty, které obsahují šum. Proto se snažíme nalézt metody, které tento odstraní.

Mezi tyto metody patří hlavně metoda Maximum aposteriori estimation. Dále se dá použít časové struktury signálů.

4.6 Algoritmus FastICA pro komplexní signály

V případě, že potřebujeme oddělit signály s komplexními hodnotami, musíme použít variantu algoritmu FastICA pro komplexní signály. Jelikož jde o algoritmus FastICA, jsou zachovány jeho výhody, jako je robustnost a relativně malá výpočetní náročnost.

Signály s komplexními hodnotami dostaneme například při použití Fourierovy transformace. Toto řešení je vhodné zejména při oddělování konvolučních směsí signálů.

Tento algoritmus vychází z obecného modelu ICA, jež je vyjádřen pomocí (4.3). V tomto modelu budeme předpokládat proměnné **s** a **x** jako komplexní proměnné a dále také rovnost pozorovaných směsí a zdrojových signálů. Směšující matice **A** může být také komplexní.

Stejně jako u klasické ICA je nutné před samotným zpracováním provést bělení, např. pomocí PCA.

Dále zde zůstává problém s neurčitostí, protože nelze nalézt fázi hledaných zdrojových signálů (u lineárních směsí nelze nalézt znaménko zdrojových signálů).

Poté je nutné zavést aktivační funkci, stejně jako u předchozích algoritmů. Tato funkce bude mít tvar

$$J_G(\mathbf{w}) = E\left\{G\left|\mathbf{w}^{\mathbf{H}}\mathbf{x}\right|^2\right\}, \tag{4.33}$$

kde G je reálná hladká funkce, **w** je n-rozměrný komplexní vektor, **H** značí hermitovskou transpozici a střední hodnota $\mathbf{w}^{\mathbf{H}}\mathbf{x}$ je rovna 1. Je vhodné, aby funkce G nebyla

strmá, protože pak bude algoritmus odolný proti výchylkám. Uvedeme 3 funkce i s jejich derivacemi, které jsou vhodné pro algoritmus:

$$G_1(y) = \sqrt{a_1 + y}, g_1(y) = \frac{1}{2\sqrt{a_1 + y}}$$

$$G_2(y) = \log(a_2 + y), g_2(y) = \frac{1}{a_2 + y}, \tag{4.34}$$

$$G_3(y) = \frac{1}{2}y^2, g_3(y) = y$$

kde a_1 a a_2 jsou zvolené konstanty. G_1 a G_2 jsou méně strmé než G_3 a proto také vhodnější. Pokud si zvolíme funkci G_3, pak algoritmus měří aproximaci špičatosti.

Nakonec již hledáme extrémy $E = \left\{ G\left(\left| \mathbf{w}^{\mathbf{H}} \mathbf{x} \right|^2 \right) \right\}$. Ukázka derivace je v [8]. Je nutné pozorované směsi vybělit tak, aby $E\left\{ \mathbf{x}\mathbf{x}^{\mathbf{H}} \right\} = \mathbf{I}$. Výsledný algoritmus má poté tvar:

$$\mathbf{w}^+ = E\left\{ \mathbf{x}\left(\mathbf{w}^{\mathbf{H}} \mathbf{x} \right) * g\left(\left| \mathbf{w}^{\mathbf{H}} \mathbf{x} \right|^2 \right) \right\} - E\left\{ g\left(\left| \mathbf{w}^{\mathbf{H}} \mathbf{x} \right|^2 \right) + \left| \mathbf{w}^{\mathbf{H}} \mathbf{x} \right|^2 g'\left(\left| \mathbf{w}^{\mathbf{H}} \mathbf{x} \right|^2 \right) \right\} \mathbf{w}$$

$$\mathbf{w}_{iterace} = \frac{\mathbf{w}^+}{\left\| \mathbf{w}^+ \right\|}. \tag{4.35}$$

Tímto se určí jeden nezávislý komponent. Aby algoritmus nekonvergoval ke stejným extrémům, je nutno po každé iteraci využít Gram-Schmidtovu ortogonalizaci (viz Tab. 4.1, pouze se namísto obyčejné transformace využije transformace hermitovské). Pokud chceme určit všechny nezávislé komponenty najednou, lze využít ortogonalizace symetrické:

$$\mathbf{W} = \mathbf{W}\left(\mathbf{W}^{\mathbf{H}} \mathbf{W} \right)^{-1/2} \tag{4.36}$$

5 Implementace algoritmu komplexní FastICA v C++

V této práci jsem se pokusil vytvořit implementaci jednoho z algoritmů ICA v jazyce C++. Vybral jsem si algoritmus FastICA pro komplexní signály a tento jsem naprogramoval v prostředí MS Visual Studio 2005.

5.1 Použité nástroje a knihovny

Na úvod je třeba říct, že implementace některého z algoritmů ICA nebo BSS v jazyce C/C++ je na rozdíl od implementace v programu Matlab [13] značně náročnější, jelikož je třeba nejdříve vytvořit určitý rámec, který nám dovolí pracovat se složitějšími datovými typy, jako jsou matice a vektory. Tyto datové typy jsou již v Matlabu vestavěné a k nim je přidáno mnoho funkcí, které programátorovi značně ulehčí vývoj algoritmu. Proto je vhodné využít některou z existujících rozšiřujících knihoven pro vývoj v C++.

Využití takovéto knihovny sice usnadní práci, ale dosáhnout jednoduchosti programování v Matlabu nelze. Dále je nutné mít na paměti, že Matlab provádí konverze jednotlivých datových typů automaticky podle potřeby, kdežto v C++ je nutné být obezřetný k těmto konverzím, jelikož při nesprávném využití dochází k runtime chybám, které je nutno zdlouhavě debuggovat.

Jak již bylo naznačeno, algoritmus jsem implementoval v prostředí MS Visual Studio 2005, které nabízí programátorovi mnoho možností a funkcí jak při vývoji, tak při ladění programů.

Dále jsem využil knihovnu IT++ [12], která je komunitním řešením, a je proto zdarma. Tato knihovna nabízí velmi rozsáhlé rozhraní pro usnadnění vývoje. Je zde obsaženo mnoho tříd pro vývoj programů pro zpracování řeči, signálů, třídy pro datovou komunikaci a také matematické třídy. Tato knihovna pro některé své funkce potřebuje další knihovny, které jsou rovněž dostupné zdarma. Jsou to zejména knihovny: BLAS, LAPACK, FFTW. Je nutno dodat, že knihovny BLAS a LAPACK lze nahradit komerčními knihovnami firem Intel a AMD.

Kompilace knihovny IT++

Je nutné uvést, že vývojáři knihovny IT++ nepodporují instalaci této knihovny v prostředí Visual Studio, takže mnou uváděný postup je do jisté míry nestandardní, ale je funkční a jeho výhodou je, že lze nakonec využít všech výhod spojených s programováním v IDE (Integrated Development Environment). Standardní postup instalace knihovny IT++ je napsán na stránkách projektu. Tato instalace je především pro operační systémy Unixového typu, avšak lze ji rovněž provést v systému MS Windows, avšak je zapotřebí program Cygwin (vytváří Unixové prostředí na architektuře Windows). Vývojáři projektu zde rovněž popisují postup pro kompilaci ve Visual Studiu, ale pro tento postup je nutné mít nainstalovanou některou z komerčních knihoven MKL, nebo ACML.

V případě, že nemáme přístup k těmto knihovnám, je třeba postupovat poněkud odlišně a využít postup navržený Hervé Boeglenem [11]. Postup je uveden na jeho osobních stránkách a je doplněn obrázky s postupem. Jeho postup předpokládá využití již zkompilovaných pomocných knihoven BLAS, LAPACK a FFTW a jejich vložení do systému. Dále je potřeba správně upravit zdrojové kódy knihovny IT++, aby využívala možností podpůrných knihoven. Výsledkem jsou pak dvě sady knihoven IT++, které lze využít pro psaní programů ve Visual Studiu. Jedna sada má v sobě ponechány informace pro debugger a je proto vhodná při vývoji programu, naopak druhá tyto informace

neobsahuje a je navíc kompilována s ohledem na rychlost kódu. Pokud máte navíc MS Visual Studio v plné verzi (ne verze Express), lze navíc vynechat instalaci Microsoft Platform SDK.

Na internetových stránkách [11] je dále uveden postup, jak začít využívat knihovnu IT++ pro nový projekt ve VS.

5.2 Popis funkcí

Program je postaven na hlavní třídě `cfastica`, která je součástí příloh. Tato třída je založena na využití některých z funkcí obsažených v knihovně IT++. Tento postup byl zvolen hlavně z toho důvodu, že samotná implementace maticového a vektorového počtu v C++ by byla časově náročná, jelikož by bylo nutné se zabývat mnoho elementárními funkcemi spojenými se samotnou úpravou, vytvářením a početními úkony s maticemi a vektory. V IT++ je již zabudován rámec pro maticový a vektorový počet, který je jak robustní, tak flexibilní. Tato robustnost spočívá hlavně ve využití dynamicky alokovaných vícerozměrných polí, jež jsou základním kamenem při využití maticového počtu v C++, a jejich kvalitní správě v paměti počítače. Toto je zejména důležité, protože při pokusu vytvořit vlastní dynamická pole se otevírá prostor k chybám, které jsou závažné a projevují se většinou až při běhu programu a jsou těžko odhalitelné. Proto je výhodné stavět již na odzkoušeném postupu a zbytečně se nezabývat základními věcmi, které jsou jednak časově náročné a dále také náchylné k chybám.

Druhou výhodou je flexibilnost těchto funkcí, jelikož většina datových typů z IT++ je vytvořena pomocí šablon tříd a proto je možné vytvářet složitější datové typy IT++ s pomocí většiny vestavěných datových typů.

Při samotném využívání funkcí a datových typů z IT++ je téměř vždy nutné nahlížet do dokumentace na stránkách projektu, protože použití některých funkcí není jednoznačné a oproti Matlabu není vždy zcela zřejmé jak danou funkci správně využít. Avšak při použití dokumentace se lze mnoha chyb vyvarovat. Dále jsou u většiny funkcí obsažená metadata, která se zobrazují při využívání těchto funkcí, a proto jsou pomůckou při vývoji.

Samotný program vychází z volně dostupné implementace v Matlabu [xx]. Oproti tomuto kódu není naprogramována postupná aproximace jednotlivých zdrojových signálů, ale je zde pouze aproximace všech zdrojů najednou (pomocí symetrické ortogonalizace).

Nyní se pokusíme ukázat princip fungování této implementace. Nejdříve je nutné zavolat konstruktor třídy `cfastica::cfastica(cmat nesmichane, double eps, bool smichat = true)`, který má v prvním parametru matici komplexních pozorovaných směsí (program umožňuje i umělé smíchání nesmíchaných signálů pomocí náhodné komplexní matice **A**), v dalším parametru nastavujeme parametr a_2 kontrastní funkce g_2 z (4.34) a nakonec boolovskou proměnnou `smichat`, která určuje, zda se mají vstupní signály smíchat (při jejím neuvedení je standardně nastavena na true). Tento konstruktor inicializuje hlavní proměnné třídy `cfastica`, v případě potřeby smíchá signály a nakonec volá členskou funkci (ang. member function) `void cfastica::beleni()`.

Tuto funkci si zde pro srovnání ukážeme i s její alternativou v Matlabu, protože je jasně vidět, jak i při použití knihovny IT++ roste délka kódu a je třeba pečlivě dodržovat konverze jednotlivých datových typů, aby nedocházelo k chybám.

Kód z C++ pro funkci `void cfastica::beleni()`:

```
void cfastica::beleni()
{
	cvec dx;			//eigenvalue kdyz je nehermitovska
	vec dx_hermit;		//eigenvalue kdyz je hermitovska
	cmat Vx;			//eigenvector
	cmat dxdiag;		//eigenvalues diagonalni matice
	cmat sqrtdxdiag;	//sqrt(inv(dx))
	cmat temp;
	cmat C_pomocna;

	C_pomocna = komplexni_kovariance(this->xold);

	if(C_pomocna.hermitian_transpose() ==
C_pomocna.transpose())
		//neni hermitovska
	{
		eig(C_pomocna,dx,Vx);
	}
	else
		//je hermitovska
	{
		eig_sym(C_pomocna,dx_hermit,Vx);
		dx = to_cvec(dx_hermit);
	}
		//podminka jestli je matice hermitovska

	dxdiag = diag(dx);

	temp = itpp::inv(dxdiag);

	sqrtdxdiag = zeros_c(dxdiag.rows(),dxdiag.cols());
	for(int i=0; i<sqrtdxdiag.rows(); i++)
		for(int j=0; j<sqrtdxdiag.cols(); j++)
			if(i==j)
				sqrtdxdiag(i,j) = std::pow(temp(i,j),0.5);

	this->whiteningMatrix = sqrtdxdiag * Vx.H();

	this->x = whiteningMatrix * this->xold;
}
```

A kód z Matlabu pro stejnou funkčnost:

```
[Ex, Dx] = eig(cov(xold'));
Q = sqrt(inv(Dx)) * Ex';
x = Q * xold;
```

V této funkci je použita pomocná funkce `cmat cfastica::komplexni_kovariance(cmat X)` pro výpočet kovariance komplexní proměnné, která musela být vytvořena, jelikož knihovna IT++ tuto nemá implementovánu. Dále je vidět, že oproti Matlabu je nutné brát v potaz tvar vstupní matice pro výpočet vlastních hodnot a vlastních vektorů, jelikož pro hermitovskou matici je nutno využít jinou funkci než pro matici nehermitovskou. Je zde také vidět, jak je řešena odmocnina diagonální matice `dxdiag` pomocí kódu `sqrtdxdiag(i,j) = std::pow(temp(i,j),0.5);`.

Dále bylo zapotřebí odprogramovat již zmíněnou funkci pro kovarianci komplexní proměnné, která využívá pomocnou funkci `std::complex<double> cfastica::ccov(cvec x1, cvec x2)`, jež vrací komplexní číslo a jejími vstupními hodnotami jsou vždy dva řádky ze vstupní matice, pro niž tuto kovarianci počítáme. Čísla řádků záleží na prvku kovarianční matice, který počítáme. Následuje kód pro funkci `ccov`.

```
std::complex<double> cfastica::ccov(cvec x1, cvec x2)
{
	int N = x1.size() - 1;
	//int N = x1.size();

	cvec x1c,x2c;

	double c_u12, c_v12, c_vu12, c_uv12;

	std::complex<double> c12;

	x1c = x1 - mean(x1);  //centrovane nahodne veliciny
	x2c = x2 - mean(x2);

	c_u12 = (elem_mult_sum(real(x1c),real(x2c)))/N;
	//kovariance realnych casti
	c_v12 = (elem_mult_sum(imag(x1c),imag(x2c)))/N;
	//kovariance imag. casti
	c_vu12 = (elem_mult_sum(imag(x1c),real(x2c)))/N;
	//kriz. kovariance imag:real-1:2
	c_uv12 = (elem_mult_sum(real(x1c),imag(x2c)))/N;
	//kriz. kovariance real:imag-1:2

	std::complex<double> temp(c_u12 + c_v12,c_vu12 - c_uv12);
	c12 = temp;

	return c12;
}
```

Následuje popis funkčnosti členské funkce `void cfastica::symetricka_separace()`, která je hlavní funkcí třídy a ve které je implementován samotný algoritmus FastICA pro komplexní signály. Nejprve je definováno a inicializováno několik pomocných proměnných a je zde také nastaven počet iterací hlavního algoritmu pomocí proměnné `int maxcounter = 10;`. Dále je také náhodně určena separační matice **W**, která je poté v každém běhu algoritmu upravována. Jednotlivé iterace algoritmu probíhají v cyklu `while ( counter < maxcounter)`.

V tomto je poté pro jednotlivé řádky matice **W** proveden cyklus `for (int i=0; i<n;i++)`, ve kterém se provádí výpočty shodné s následujícím kódem pro Matlab a odpovídající (4.35):

```
gWx(j,:) = 1./(eps + abs(W(:,j)'*x).^2);
dgWx(j,:) = -1./(eps + abs(W(:,j)'*x).^2).^2;
W(:,j) = mean(x .* (ones(n,1)*conj(W(:,j)'*x)) .*
(ones(n,1)*gWx(j,:)),2) - mean(gWx(j,:) + abs(W(:,j)'*x).^2 .*
dgWx(j,:)) * W(:,j);
```

Další část kódu již odpovídá symetrické ortogonalizaci z (4.36). Po proběhnutí všech iterací již následuje samotná separace pomocí vynásobení matice smíchaných signálů **x** separační maticí **W** zleva.

Poslední členskou funkcí třídy je `cmat cfastica::fourier(cvec X, int velikostOkna)`, která segmentuje vstupní signál na segmenty o délce `velikostOkna` s polovičním překryvem. Tyto segmenty jsou následně váhovány pomocí Hanningova okna a podrobeny Fourierově transformaci po sloupcích.

5.3 Využití knihovny

V samotném programu se mnou vytvořená knihovna použije tak, že se nejdříve zavolá konstruktor se vstupními parametry. Dále se již zavolá pouze členská funkce `void cfastica::symetricka_separace()` na objekt vytvořený konstruktorem.

Po provedení výpočtů je již v objektu matice separovaných signálů, se kterou se může dále pracovat.

Je nutno poznamenat, že program obsahuje pouze třídu `cfastica`, a další rozšiřující funkce nejsou dostupné. Proto je čtení a zápis do souborů řešen s pomocí knihovny IT++, která umožňuje tyto operace provádět do vlastního typu souboru s příponou „it“. Tyto soubory je poté možno číst a zapisovat i v programu Matlab s pomocí přiložených souborů. Výhodou je, že datové typy a konkrétní proměnné se ukládají i se jmény a s příslušným typem, což usnadňuje následné čtení.

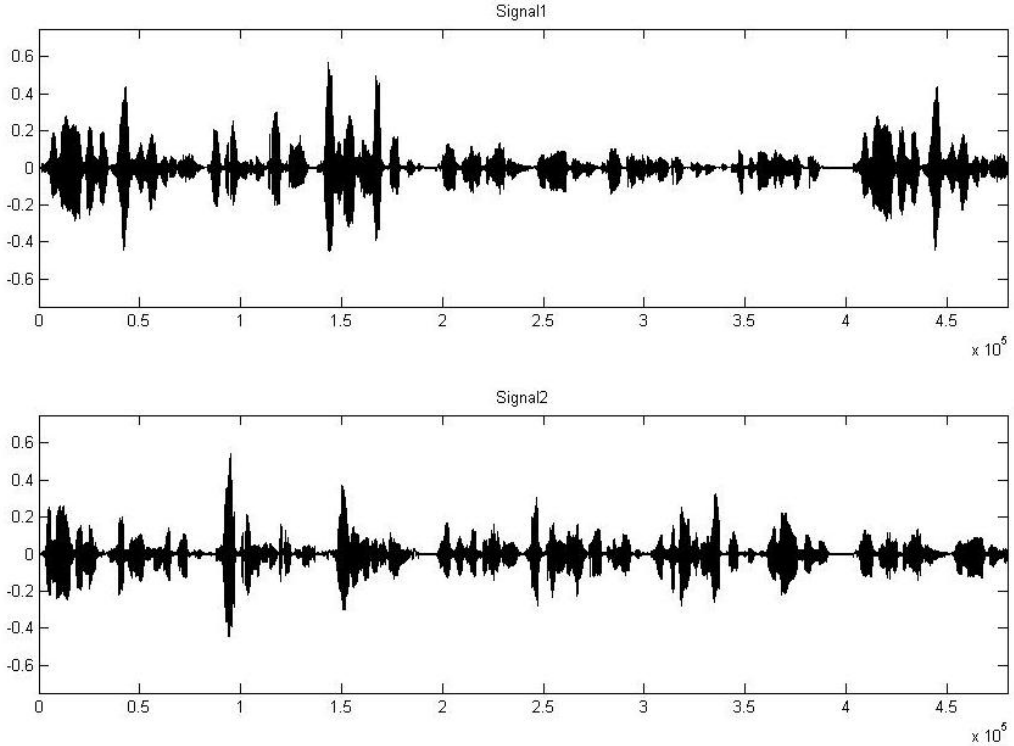
Z těchto důvodů jsou v následující kapitole také uvedena grafická znázornění z Matlabu, avšak samotné výpočty probíhaly za pomocí mého programu.

6 Příklad použití algoritmu FastICA pro komplexní signály

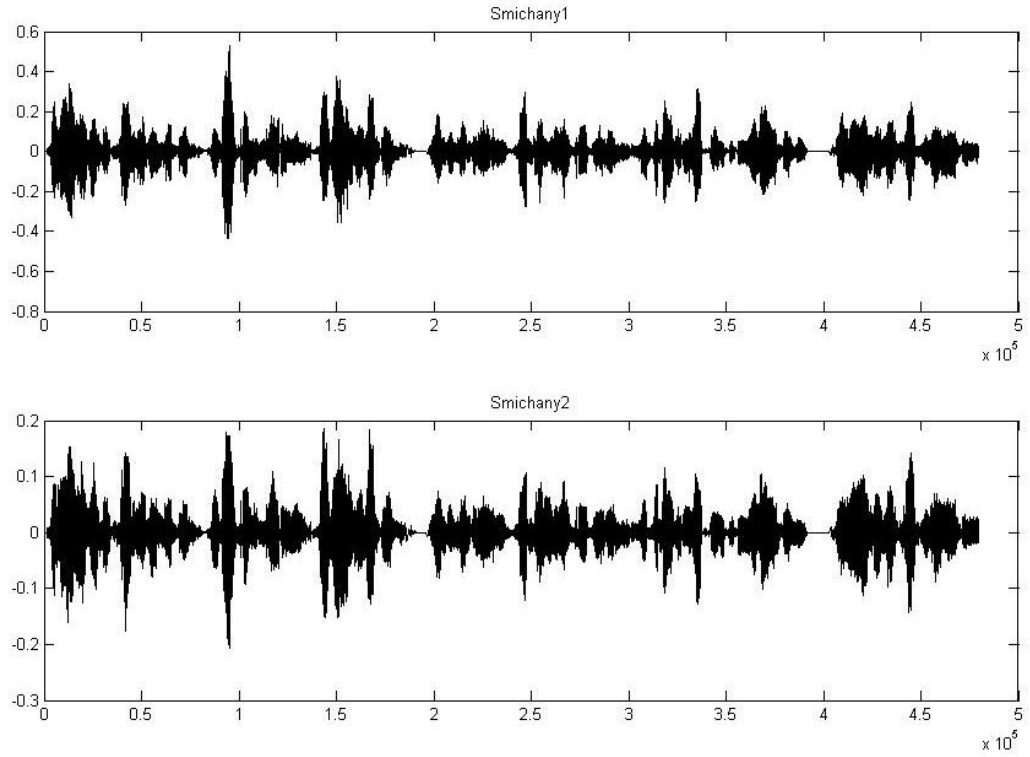
V této kapitole budou uvedeny příklady separace zvukových signálů. Dále budou uvedeny některé závislosti pro výpočetní náročnost a také srovnání implementace v C++ a v Matlabu.

Nejprve bude uveden příklad umělého smíchání reálných zvukových nahrávek pomocí náhodně generované komplexní matice **A**. Tento příklad sice nesimuluje reálné smíchání zvuků v odrazovém prostředí, ale je přiblížením lineárního smíchání v bezodrazové komoře.

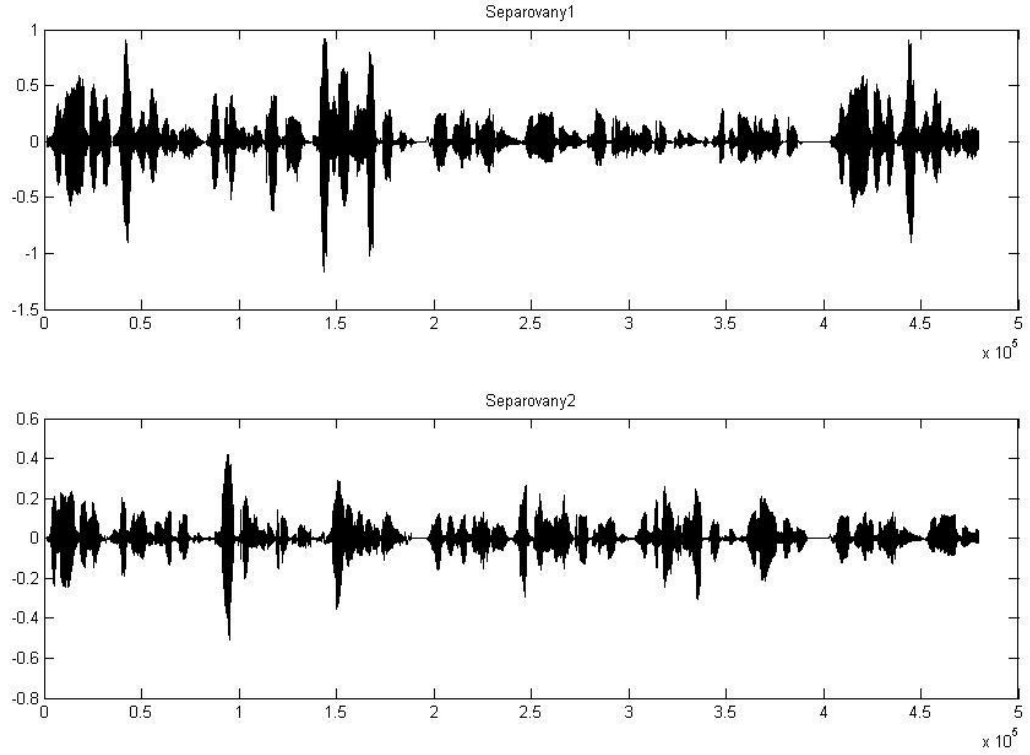
Na obr. 6.1 jsou uvedeny dva vstupní signály o stejné délce, které jsou poté uměle smíchány. Tyto smíchané signály jsou vyobrazeny na obr. 6.2. Na obr. 6.3 jsou uvedeny již separované signály (pouze jejich reálná složka).



Obr. 6.1 – Dva zdrojové signály, určené k umělému smíchání. Oba signály trvají přibližně 10s a mají stejný počet vzorků (vzorkováno 48000 vz/s).



Obr. 6.2 – Signály (jejich reálné složky) po smíchání komplexní maticí **A**. Z obrázku je patrné, že průběhy mají již podobný tvar, avšak jejich energie je rozdílná.



Obr. 6.3 – Reálné složky separovaných signálů. Z obrázku není příliš patrná inverze signálů (neurčitost znaménka – vyplývá z teorie) oproti původním signálům.

Z obrázků lze poznat, že došlo k separaci smíchaných zdrojových signálů, avšak tyto byly smíchány pouze lineárně. Pokud bychom zdrojové signály smíchali v reálném

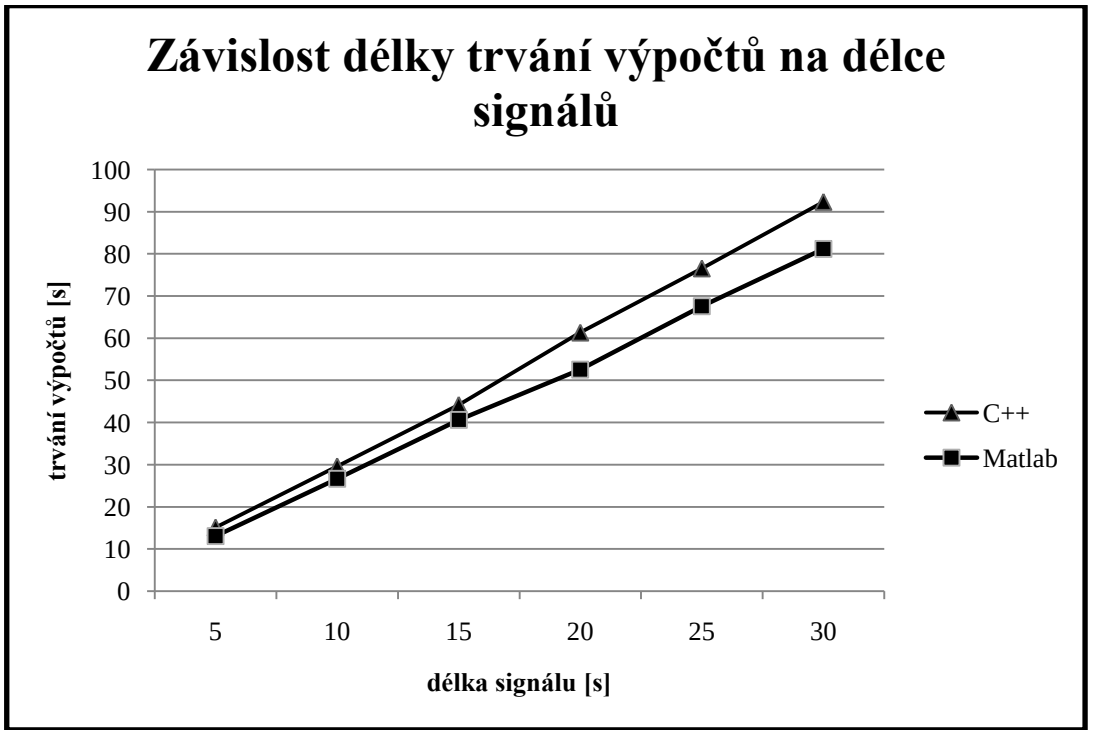
prostředí (vznikly by konvoluční směsi), pak by nabízený algoritmus selhal. Pokud máme konvoluční směsi, pak lze využít například algoritmus FDICA (frequency-domain ICA). Bohužel se zpětným přechodem k časové závislosti vyvstává problém neurčitosti znaménka a pořadí signálů (ve standardní ICA nám nevadí).

6.1 Naměřená pozorování

V této části si uvedeme příklad výpočetní náročnosti algoritmu FastICA pro komplexní signály vyjádřenou pomocí časové závislosti doby výpočtu na délce trvání zdrojových signálů. Tato závislost bude uvedena pro implementaci v C++ i v programovém prostředí Matlab. Jednotlivé naměřené časy jsou uvedeny v tab. 6.1 a jejich grafické znázornění v obr. 6.4.

Tab. 6.1 – Závislost doby výpočtu algoritmu FastICA pro komplexní signály na době trvání signálů. Signály byly vzorkovány 48000 vz/s.

Délka signálů	C++	Matlab
t[s]	t[s]	t[s]
5	15,1406	13,0794
10	29,6406	26,6777
15	44,2031	40,6367
20	61,3281	52,5568
25	76,5469	67,5701
30	92,3281	81,1827



Obr. 6.4 – Závislost délky trvání výpočtů na délce signálů. Z grafu je patrná lineární závislost délky výpočtů. Drobné odchylky od linearity lze přičíst nepřesnosti měření, která je závislá od aktuálního vytížení PC.

6.2 Diskuze výsledků a možná vylepšení

Z naměřených dat je zřejmé, že implementace v programu Matlab je rychlejší. Jelikož pro výpočty z oblasti lineární algebry využívá Matlab stejné knihovny jako knihovna IT++ (tyto jsou BLAS a LAPACK), pak lze důvody pomalejšího kódu pro C++ hledat hlavně v optimalizaci, dále pak ve verzi použitých knihoven a jejich optimalizaci pro dané sestavení.

Tato optimalizace implementace v C++ by hlavně spočívala v redukci počtu použitých proměnných zejména v členské funkci knihovny `cfastica void cfastica::symetricka_separace()`, která nese hlavní výpočetní úkony daného algoritmu. Dále by se mohl zmenšit počet prováděných operací tak, že by se některé příkazy sloučily. Avšak nastává problém, protože je nutné brát velkou pozornost na konverze jednotlivých datových typů, jelikož je použití knihovny IT++ v tomto ohledu složitější oproti Matlabu, kde jsou tyto konverze prováděny automaticky a programátor na ně nemusí brát zřetel.

Pokud se vrátíme k problému velkého počtu použitých proměnných, pak tento souvisí s výše uvedeným problémem velkého počtu malých operací – návratové hodnoty těchto operací jsou ukládány do dočasných proměnných, které je nutno stále vytvářet. To vytváří problém, jelikož zde pracujeme s mnohdy obrovskými datovými poli (např. signál délky 30s se vzorkováním 48000 vz/s obsahuje 1440000 vzorků), která je nutno dynamicky alokovat v paměti, což zabírá čas. Navíc vlastností C++ je zahazování proměnných po každé ukončené iteraci určitého cyklu, což vede opět k dealokaci těchto polí a další ztrátě času.

Z výše uvedených důvodů by byla tato optimalizace velice náročná na čas a to hlavně ve fázi ladění a odstraňování chyb za běhu programu, jelikož tyto chyby se projeví až díky konverzi do určitého datového typu a nejsou zřejmé při kompilaci programu.

7 Závěr

V této práci jsem se pokusil nastínit řešení jedné z metod slepé separace zdrojů (BSS). Touto metodou byla nezávislá analýza komponent (ICA), která je v dnešní době metodou nejrozšířenější.

Jelikož jsou metody ICA založeny na principech statistik vyšších řádů a dalších složitých principů, byla v této práci zahrnuta kapitola 2, která se snažila podat zevrubný náhled do používané tématiky. Jinak by v následujících kapitolách nemohly být jasně vysvětleny používané algoritmy.

V kapitole 3 byly uvedeny postupy vhodné k předzpracování dat, které nám následnou ICA mohou usnadnit. Byly uvedeny postupy PCA – analýza hlavních komponentů a metoda bělení. Metody PCA jsou důležité zejména v případech, kdy máme k dispozici více pozorování, než zdrojových signálů (tzn. směšující matice není čtvercová) a mohou nám dokonce v některých případech pomoci odstranit šum ze směsí. Pokud bychom použili PCA nelineární, tak ta se potom dá použít k nalezení vlastních nezávislých komponentů [1, s. 239]. Metody bělení nám poté mohou usnadnit určení nezávislých komponent.

Vlastní odvození používaných algoritmů je uvedeno v kapitole 4. Byly zde uvedeny algoritmy založené na měření odlišnosti od Gaussova rozdělení – špičatost a negentropie, a také metoda maximalizace věrohodnosti.

V předposlední části kapitoly 4 byly navíc uvedeny postupy při odstraňování šumů ze směsí signálů. Tyto postupy záležely hlavně na povaze šumu – zdrojový nebo senzorový. Některé z těchto metod jsou založeny na metodách bezšumových, ale jiné jsou mnohdy velmi složité a výpočetně náročné, proto jim nebyl věnován velký prostor. Navíc se ukazuje, že základní metody ICA lze použít i na zašuměné signály (odstup šumu by měl být alespoň 20 dB) – a tyto pak oddělit. Z již naznačené složitosti jiných postupů vyplývá, že jejich použití by se mělo předejít a to hlavně použitím klasických metod zpracování signálů – filtrů.

Závěr 4. kapitoly byl věnován popisu algoritmu FastICA pro komplexní signály, který byl implementován v jazyce C++. Tato implementace byla provedena s využitím knihovny IT++ a byla popsána v kapitole 5.

V 6. kapitole bylo uvedeno reálné použití implementovaného algoritmu spolu s ukázkou výpočetní náročnosti tohoto řešení. Dále bylo uvedeno srovnání s implementací v programu Matlab. Jelikož byl program v C++ pomalejší, než jeho ekvivalent v Matlabu, bylo naznačeno, ve kterých částech kódu by mohlo dojít k optimalizaci.

Použitá literatura a internetové zdroje

Literatura

[1] HYVÄRINEN, A., KARHUNEN, J., OJA, E. Independent component analysis. John Wiley & Sons, Toronto, 2001. 481s. ISBN 0-471-40540-X.

[2] FAJMON, B., RŮŽIČKOVÁ, I. Matematika 3. Skripta FEKT VUT v Brně. Brno, Ústav matematiky, 254s.

[3] HYVÄRINEN, A. Survey on Independent Component Analysis. Paper. Helsinky University of Technology, Laboratory of Computer and Information Science. 35s.

[4] EKSLER, V. Prostorová lokalizace a separace naslepo zdrojů akustických signálů polem mikrofonů. Doktorská práce. Brno, 2006. 98s.

[5] HYVÄRINEN, A. Fast and Robust Fixed-Point Algorithms for Independent Component Analysis. Paper. Helsinky University of Technology, Laboratory of Computer and Information Science. 16s.

[6] CHOI, S., CICHOCKI, A., PARK, H. M., LEE, S.Y. Blind Source Separation and Independent Component Analysis: A Review. Neural Information Processing - Letters and Reviews, Vol. 6, No. 1, January 2005. 57s.

[7] KOVÁR, M. Maticový a tenzorový počet. Skripta FEKT VUT v Brně. Brno, Ústav matematiky, 179s.

[8] BINGHAM, E., HYVÄRINEN, A. A fast fixed-point algorithm for independent component analysis of complex valued signals. International Journal of Neural Systems, Vol. 10, No. 1, February, 2000. 8s.

Internetové zdroje

[9] The Freesound Project. Dostupné z: <http://freesound.iua.upf.edu/>.

[10] FastICA. Dostupné z: <http://www.cis.hut.fi/projects/ica/fastica/>.

[11] Hervé Boeglen. Kompilace knihovny IT++ ve VS. Dostupné z: <http://herve.boeglen.free.fr/itpp_windows/>.

[12] Knihovna IT++. Hlavní stránka projektu. Dostupné z: <http://itpp.sourceforge.net/>.

[13] Implementace algoritmu FastICA pro komplexní veličiny v programu Matlab. Dostupné z: <http://www.cis.hut.fi/ella/publications/cfastica_public.m>.