ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

FastICA

Semestrální práce z předmětu KIV/TKS

Vypracoval : **Václav Souhrada**
Os. č. : A08N0049P
Datum : 25. ledna 2009
E-mail : v.souhrada@students.zcu.cz

Obsah

1 Zadání....3
2 ICA....4
2.1 Předpoklad ICA....7
2.2 Nejednoznačnosti ICA....7
2.2.1 Pořadí nezávislých komponent....7
2.3 Předzpracování signálů pro ICA....7
2.3.1 Centrování....7
2.3.2 Bělení (Whitening)....8
3 FastICA....9
3.1 Vlastnosti FastICA algoritmu....9
3.2 Minimalizace odhadu negentropie....10
3.3 Parametry algoritmu FastICA....10
4 Projekt FastICA for Java....12
4.1 Struktura balíčků a jejich tříd....12
4.1.1 Balíček org.fastica....12
4.1.2 Balíček org.fastica.math....13
4.1.3 Balíček org.fastica.swing....13
4.1.4 Balíček org.fastica.util....14
5 Implementace FastICA....15
5.1 Problémy stávající implementace....15
5.2 Nová struktura FastICA implementace....16
6 Test algoritmu FastICA....17
6.1 Popis testovací úlohy....17
6.2 Test....17
6.3 Postřehy z provedeného testování....29
7 FASTICA při detekci a odstranění artefaktů z EEG....30
7.1 Proč algoritmus FastICA?....31
7.2 Předpoklady zpracování EEG prostřednictvím ICA/FastICA....31
7.3 Ukázky separací nezávislých komponent....31
8 Závěr....33
Seznam použité literatury....34

1 Zadání

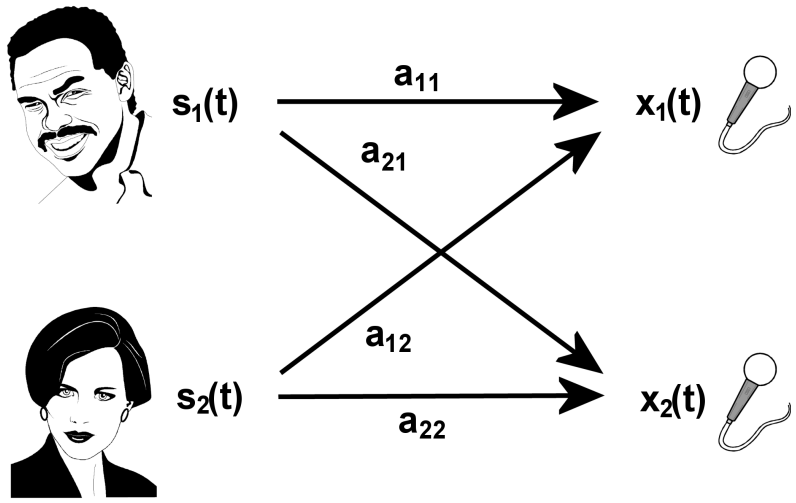
Independent Component Analysis při odstraňování artefaktů v EEG signálech při detekci ERP

- Nejdříve prostudujte princip metody ICA z dodaných podkladů.
- Vyberte vhodný způsob implementace metody ICA.
- Implementujte metodu v jazyce Java, jako lehkou komponentu (metoda bude pracovat pouze s primitivními datovými typy a poli)
- Bude kladen důraz na přehlednost a kvalitu kódu, dále bude požadována podrobná dokumentace v JavaDoc.
- Implementace metody bude začleněna do vznikající knihovny.
- Otestujte možnosti metody při detekci artefaktů a jejich odstraňování z EEG. Zvláště se zaměřte na problematiku nechtěného odstranění ERP z původního EEG.
- Výsledky shrňte do krátké 3-6 stránkové zprávy.

2 ICA

Analýza nezávislých komponent[1] (ICA[2]) je velmi známou a používanou metodou pro slepou separaci signálů (BSS[3]) [CICHOCKI AMARI] a dekonvoluci signálů. Hlavním principem je předpoklad nezávislosti původních signálů. Omezením ICA je, aby nezávislé komponenty neměly normální rozložení (Gaussovské) hustoty pravděpodobnosti jednotlivých vzorků.

Klasickým příkladem uváděných v literaturách zabývajících se algoritmem ICA je tzv. „***Coctail party problem***“. Ten si můžeme popsat jako situaci na nějakém větším večírku. Budeme samozřejmě předpokládat, že se na večírku lidé dobře baví, a tudíž probíhají mezi nimi různé konverzace (nudný večírek bez konverzace by nebyl příliš ideální pro popis ICA :-)). Naším úkolem je vnímat z davů hlasů pouze jeden hlas tzn. *separovat* jeden zdroj. Pro zjednodušení situace předpokládejme, že na večírku konverzují v danou chvíli, kdy budeme separovat zdroje, pouze dva mluvčí (s_1 a s_2), kteří vydávají zvuky $s_1(t)$ a $s_2(t)$ (viz. Obrázek 2.1).



Obrázek 2.1: Koktejl party – dva zdroje

Uši (na obrázku 2.1 mikrofony, snímače hlasů) zachycují signály $x_1(t)$ a $x_2(t)$ těchto hlasů. To lze vyjádřit následujícími rovnicemi:

(2a)

$$x_1(t) = a_{11}\, s_1 + a_{12} s_2$$

$$x_2(t) = a_{21}\, s_1 + a_{22} s_2$$

1 Nezávislé komponenty jsou proměnné, které jsou skryté a nemohou tak býti přímo měřeny nebo pozorovány.
2 Independent Component Analysis
3 Blind Signal Separation

Předpokladem úspěšné separace signálu s_1 a s_2 ze signálu x_1 a x_2 je nezávislost s_1 a s_2. Obdobným problémem jako je *Coctail party problem*, je měření EEG[4]. Nyní se podíváme na obecný popis ICA algoritmu podle obrázku 2.2.

Nezávislé komponenty jsou skryté proměnné, které nemohou být přímo měřeny nebo pozorovány. Například na obrázku 2.2 jsou to hlasy různých lidí (viz *Coctail party problem* uvedený v úvodu této kapitoly). V oblasti BSS se setkáme se třemi základními modely popisující transformaci:

- *lineární model*
- *šumový model*
- *konvolutorní model*

EEG data splňují lineární model, který můžeme popsat následujícím způsobem. Předpokládejme, že matice **A** je neznámá, a proto musíme z pozorované vektoru **X** odhadnout **A** i **S**. Z tohoto dostáváme vztah

$$\mathbf{X} = \mathbf{A} \,.\, \mathbf{S} \tag{2.1}$$

$$\mathbf{S} = (s_1, \ldots, s_n)^T \quad , \quad \mathbf{X} = (x_1, \ldots, x_n)^T$$

kde:

- **A** – mixážní matice rozměrů *n x m* (matice popisující prostředí měření - např. prostředí naší *Coctail party*).
- **S** – *m* rozměrný vektor vstupních signálů (nezávislé komponenty)
- **X** – *n* rozměrný vektor pozorovaných (měřených signálů)

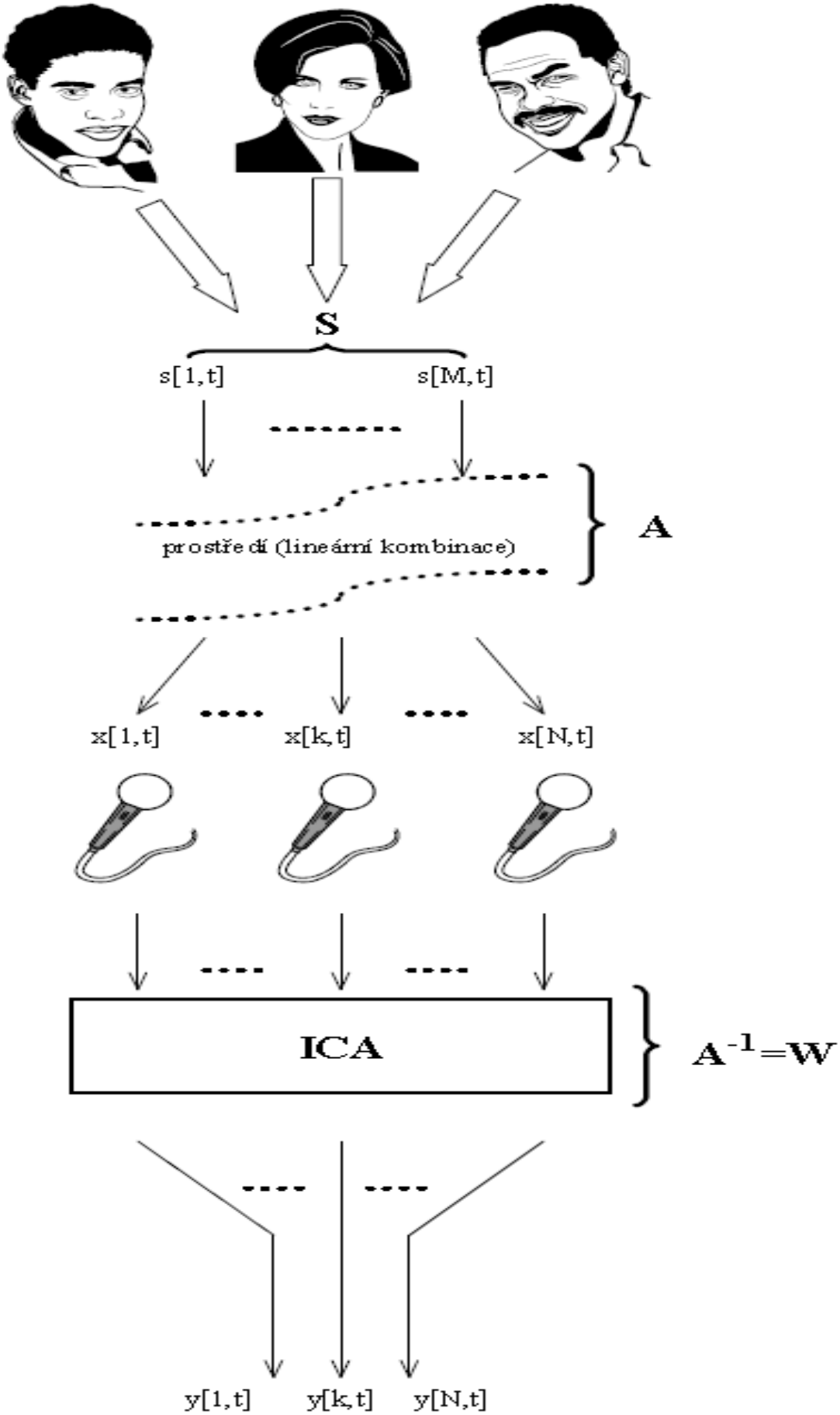
Počet měřených signálů *n* a počet nezávislých komponent *m* nemusí být shodný, ale pro správné provedení separace předpokládejme splnění nerovnosti $n \geq m$.

Zdroje (signály, nezávislé komponenty) mohou být odhadnuty následovně:

$$\mathbf{S} = \mathbf{A}^{-1} \,.\, \mathbf{X} = \mathbf{W} \,.\, \mathbf{X} \tag{2.2}$$

kde **W** je *inverzní* matice k *mixážní* matici **A**. Problém nalezení zdrojových signálů (nezávislých komponent) přechází na problém nalezení inverzní matice **W**.

4 Electroencephalography



Obrázek 2.2: Koktejl party – více zdrojů

Separace či rozklad signálů je prováděn tak, aby výsledné komponenty byli v rámci všech možností nezávislé a současně jejich lineární kombinace tvořila pozorované signály. Toho dosáhneme maximalizací funkce **F**, která měří nezávislost komponent.

2.1 Předpoklad ICA

Hlavním předpokladem pro použití ICA algoritmu je nezávislost jednotlivých komponent.

2.2 Nejednoznačnosti ICA

Analýza nezávislých komponent neposkytuje zcela jednoznačný výsledek. V praxi se můžeme setkat se dvěma nejednoznačnostmi:

- *Nemůžeme určit jejich energii*
- *Pořadí nezávislých komponent*

2.2.1 Pořadí nezávislých komponent

Můžeme volně měnit pořadí zdrojů v matici **S** a pořadí odpovídajících sloupců v matici **A**.

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{S} = \mathbf{A} \cdot \mathbf{P}^{-1} \cdot \mathbf{P} \cdot \mathbf{S} = \mathbf{A}'\mathbf{S}'$$
$$\mathbf{A}' = \mathbf{A} \cdot \mathbf{P}^{-1}$$
$$\mathbf{S}' = \mathbf{P} \cdot \mathbf{S}$$

Rovnice (2.3)

Část rovnice (2.3) **P**.**S** je rovna původním nezávislým proměnným, ale v jiném pořadí. Matice **A**.**P** je právě ta neznámá směšovací matice a **P** je permutační matice.

2.3 Předzpracování signálů pro ICA

Než začneme používat algoritmus ICA (nebo jeho další implementace) na data, je vhodné provést nejprve předzpracování signálu. Předzpracování je užitečnou (důležitou) částí, neboť nám umožňuje snížit složitost výpočtu (rychlejší konvergence). V této kapitole si popíšeme různé techniky předzpracování signálů, které využijeme i v algoritmu FastICA (kapitola 3).

2.3.1 Centrování

Základním a *nejdůležitějším* předzpracováním je centrování **x**. Provádí se odečtením střední hodnoty *m* od vzorků signálu **x** (od všech složek signálu **x**), tím pádem má nulovou střední hodnotu. Tento postup je samozřejmě pouze pro zjednodušení algoritmu ICA, který dokáže pracovat se signály se střední hodnotou, avšak operace se signály s nulovou střední hodnotou je mnohem jednodušší. Odstraněnou střední hodnotu je na konci výpočtu možné přidat zpět do signálu.

2.3.2 Bělení (Whitening)

Dalším užitečným nástrojem předzpracování signálů je vybělení pozorovaných proměnných. To znamená před použitím ICA, ale po provedení centrování, transformujeme pozorovaný vektor $\boldsymbol{x}$ lineárně tak, že získáme nový vektor $\mathbf{x}^{\sim}$, který je vybělený. Jeho komponenty jsou nekorelované a mají jednotkový rozptyl. Jinak řečeno, kovarianční matice $\mathbf{x}^{\sim}$ je rovna jednotkové matici.

3 FastICA

FastICA je implementace metody ICA (Independent Component Analysis, analýza nezávislých komponent). Jedná se o algoritmus navržený v roce 1997 **Hyvarinenem** [Hyvärinen Oja], který je založen na dvou základních principech:

- Jednoduchá aproximace *negentropie*[5]
- Rychlý optimalizační algoritmus založený na *Newtonově iterační metodě*

3.1 Vlastnosti FastICA algoritmu

Algoritmus FastICA má oproti ostatním ICA metodám několik *žádoucích* vlastností. Ty nejdůležitější z nich obsahuje následující přehled (překlad z [FASTICAHOME]):

1. Velmi rychlá konvergence, která na rozdíl od ostatních metod ICA (založených na gradientních spádových metodách) je kubická nebo nejméně kvadratická (záleží na typu modelu ICA).
2. Není zde potřeba parametrů nastavující velikost kroku (na rozdíl od gradientních algoritmů). Z této vlastnosti vyplývá jednoduché použití této metody.
3. Algoritmus hledá přímo nezávislé komponenty s *negaussovským* rozložením za pomoci nelinearity q. Tím se liší od mnoha algoritmů, které nejprve odhadnou vzájemnou hustotu pravděpodobnosti a nelinearita pak musí být určena podle ní.
4. Nezávislé komponenty mohou být odhadovány jedna po druhé. To je velice užitečné k informativnímu analyzování dat a snižuje tak výpočetní nároky metody v případech, kde potřebujeme odhadnout pouze některé komponenty.
5. Provedení metody může být optimalizováno výběrem vhodné nelinearity q. To znamená získání velice robustního algoritmu pomocí malé změny.
6. FastICA má mnoho výhod z neuronových algoritmů:
    - *je paralelní*
    - *distribuovatelný*
    - *výpočetně jednoduchý*

    Stochastické gradientní metody se zdají být výhodnější jen tehdy, je-li potřeba rychlá adaptace na měnící se prostředí.

5 Konstrastní funkce sloužící k porovnání statické nezávislosti

3.2 Minimalizace odhadu negentropie

Optimalizační proces je silnou stránkou FastICA. Minimalizaci vzájemné informace získaných dat můžeme díky tvaru provést Newtonovou iterační metodou. Lze totiž efektivně počítat jeho derivaci. Výhodou je také variabilita celého algoritmu.

Volit můžeme:

1. Vhodnou nelinearitu G, pro daný druh dat.
2. Způsob optimalizace:
   a. *Výpočtem komponenty s nejmenším odhadem za podmínky nekorelovanosti s již dříve nalezenými komponentami (algoritmus **deflation**).*
   b. *Výpočtem všech komponent na jednou za podmínky nekorelovanosti (algoritmus **symmetric**).*

3.3 Parametry algoritmu FastICA

V této kapitole si popíšeme základní parametry algoritmu FastICA., které využijeme při testování implementace *FastICA for Matlab* a *FastICA for Java*. Více informací o daném testovaní naleznete v kapitole 6. Především se zaměříme na základní možnosti nastavení v *toolboxu FastICA for Matlab*.

1. **Approach** tento parametr slouží k nastavení separace komponent. K dispozici máme následující volby:
   a) *Deflation* – komponenty budou separovány jedna po druhé
   b) *Symmetric* – separace komponent proběhne najednou
2. **NumOfIC** tento parametr udává počet komponent, které budou separovány.
3. **g** tento parametr umožňuje vybrat nelinearitu, která se použije při výpočtu. K dispozici máme čtyři volby:
   a) *pow3:* $g(u) = u^3$
   b) *tanh:* $g(u) = \tanh(a_1 . u)$
   c) *gauss:* $g(u) = u . e^{\frac{-a_2 u^2}{2}}$
   d) *skew:* $g(u) = u^2$
4. **Stabilization** parametr udává, zda bude použita při výpočtu stabilizovaná verze výpočetního programu nebo použita nebude. K dispozici máme tedy dvě volby:
   a) *on* – pokud zvolíme tuto možnost, pak *FastICA* může být ovlivněna *velikostí kroku*. Když uvázne mezi dvěma body, dojde k rozpůlení iteračního kroku. Nedojde-li však ke

konvergenci do poloviny konvergenčních kroků, tak se znovu rozpůlí iterační krok na půlku pro zbytek výpočtu.

b) *off* – vypnutí změny konvergenčního kroku

5. **Only** parametrem *only* udáváme způsob provedení výpočtu. K dispozici máme dvě volby:
   a) *pca* – vrací ortogonální matici **E** a diagonální matici **D**
   b) *white* – FastICA vrátí pouze vybělené signály, *vybělenou* matici **W** a *rekonstruovanou* matici **W**.

4 Projekt FastICA for Java

Autorem projektu FastICA for Java [FASTICA4JAV] je p. **Michael Lambertz**. Projekt vznikl v rámci jeho semestrální práce v *Institutu komunikačního inženýrství na univerzitě Aachen* (RWTH). Projekt je založený na algoritmu FastICA (kapitola 3) a je distribuován pod *Apache Software License*.

4.1 Struktura balíčků a jejich tříd

Výše uvedená implementace algoritmu FastICA obsahuje následující balíčky:

- **org.fastica** - hlavní balíček implementace. Obsahuje všechny hlavní třídy vytvářející FastICA algoritmus.
- **org.fastica.math** – obsahuje různé matematické funkce pro výpočet algoritmu.
- **org.fastica.swing** – tento balíček obsahuje GUI dané implementace.
- **org.fastica.util** – třídy užitečné pro snazší práci s audio signály.

4.1.1 Balíček org.fastica

Balíček org.fastica můžeme označit za hlavní balíček celé této implementace. Obsahuje stěžejní třídy celého algoritmu.

Rozhraní

- **ContrastFunction - v**yužití kontrastních funkcí k odhadu *negentropie.*
- **EigenValueFilter** – filtr vlastních čísel může být použit pro vyloučení nevhodných údajů získaných z výsledku činnosti algoritmu PCA. Zejména u těch dat, jejichž rozptyl je velmi malý.
- **ProgressListener** – sledování pokroku (postupu) FastICA algoritmu.

Třídy

- **BelowEVFilter** – filtrování všech vlastních čísel, jejichž hodnota je menší, než požadovaná.
- **CompositeEVFilter** – tato třída může být použita k vytvoření řetězců filtrů vlastních čísel.
- **FastICA** - třída obsahující hlavní část algoritmu FastICA.
- **FastICAConfig** – konfigurace vlastních čísel.
- **FirstEVFilter** – filtruje vlastní čísla po přiřazení jejich hodnoty.

- **GaussCFunction** - tato třída je užitečná, pokud jsou nezávislé komponenty vysoce super-Gaussovské, nebo je-li jejich robustnost velmi důležitá.

- **PCA** – algoritmus Principal Component Analysis
- **PercentageEVFilter** - řadí vlastní čísla a vrací nejvyšší vlastní čísla, jejichž součet je vyšší než stanovené procento z celkového množství všech vlastních čísel.
- **Power3CFunction** – tato třída reprezentuje funkci $\mathbf{x^3}$, která je užitečná pro odhad *sub-Gaussovských* nezávislých komponent, když nejsou k dispozici žádné odlehlé komponenty.
- **SortingEVFilter** – řazení vlastních čísel a vektorů.
- **TanhCFunction** – reprezentace funkce **tanh (a * x)** , která je užitečná pro všeobecné účely jako kontrastní funkce.

Enums

- FastICAConfig.Approach
- FastICAException.Reason
- ProgressListener.ComputationState

Exceptions

- **FastICAException** – tato třída vytváří výjimku, která může nastat při běhu algoritmu FastICA.

4.1.2 Balíček org.fastica.math

- **EigenValueDecompositionSymm** - tato třída počítá vlastní číslo dekompozicí reálné symetrické matice.
- **Matrix** – třída *Matrix* nabízí řadu užitečných statických funkcí pro výpočet matic.
- **Vector** - nabízí řadu užitečných statických funkcí pro výpočet vektorů.

4.1.3 Balíček org.fastica.swing

Tento balíček obsahuje *grafické uživatelské prostředí* aplikace *FastICA for Java*. Jedná se především o hlavní Frame aplikace a různé další okna, zobrazující jednotlivé dialogy aplikace. Popis těchto tříd je k dispozici v přiloženém JavoDocu, avšak v této práci se dané GUI nevyužívá, a tak nemá cenu se o něm podrobněji zmiňovat. Uveďme si pouze výčet jednotlivých tříd:

- DialogEVFilter

- EigenValueDialog
- FastICAApp
- FastICAFrame
- FastICAThread
- MatrixDialog
- NumberDialog

4.1.4 Balíček org.fastica.util

- **AudioBuffer** – pohodlnější zacházení s audio signály.
- **AudioVector** – třída obsahuje užitečné statické metody pro výpočty s audio signály.
- **WaveMerger** – tato třída spojí dva audio signály do jednoho signálu.

5 Implementace FastICA

V předchozí kapitole jsme si popsali strukturu implementace algoritmu **FastICA** v jazyku **Java** (*FastICA for Java*). Nyní se budeme zabývat úpravou výše uvedené implementace pro náš univerzitní výzkum *v oblasti* **EEG**. Abychom mohli projekt *FastICA for Java* otestovat na EEG data, potřebujeme především pozměnit proces načítání dat. *FastICA for Java* totiž umožňuje pouze vstup *audio* signálů, kterým se ovšem v našem projektu nezabýváme. Proto bylo zapotřebí implementovat možnost načítání i ostatních souborů (především textových).

Výsledná implementace této semestrální práce bude součástí EEG knihovny, která bude implementována do softwaru WEKA-ERP. Z tohoto důvodu není nutné implementovat GUI v rámci této semestrální práce, neboť bude později obsaženo v softwaru WEKA-ERP.

Změnou prošla i samotná struktura implementace. Byly přejmenovány jednotlivé balíčky podle identifikace místního univerzitního výzkumu (viz kapitola 5.2).

5.1 Problémy stávající implementace

Pro nás největším problémem implementace *FastICA for Java* je její záměr vzniku. Jak víme z úvodu čtvrté kapitoly, projekt *FastICA for Java* vznikl pro potřeby výzkumu v *Institutu komunikačního inženýrství na univerzitě Aachen.* Na tamní univerzitě nejspíše provádí výzkumy v oblasti audio signálů, a zřejmě z tohoto důvodu jsou v dané implementaci vstupem a výstupem audio signály. Bohužel tento záměr je jen domněnkou, neboť se mi nepodařilo vypátrat potřebu použití projektu na dané univerzitě.

Další velký problém je samotná dokumentace implementace. Obdivuji p. **Michaela Lambertze** za to, že napsal na pochopení velmi složitý FastICA algoritmus, avšak můj obdiv ztratil v nedostatečné dokumentaci. Nevím, zda-li někde na tamní univerzitě mají danou dokumentaci, ale na internetu bohužel nic k dispozici kromě *JavaDocu* není. On i samotný *JavaDoc* je nedostatečně okomentován. Pokud však nějaká lepší dokumentace někde v šuplíku nebo na odstaveném médiu existuje, pak se srdečně autorovi omlouvám. Pokud však navštívíte internetové stránky [FASTICA4JAV] projektu *FastICA for Java*, pak mi jistě dáte za pravdu. Ano, jsou tam i externí odkazy na algoritmus FastICA, avšak co se týče dokumentace k samotnému projektu, pak naleznete pouze jen zmiňovaný *holý* JavaDoc.

5.2 Nová struktura FastICA implementace

Mnou modifikovaná implementace *FastICA for java* je součástí nové interní EEG knihovny algoritmů vznikající v rámci výzkumu EEG a ERP na *Fakultě aplikovaných věd, Západočeské univerzitě v Plzní*.

Struktura jednotlivých balíčků zůstala *téměř* zachována. Byl pouze přejmenován balíček *org* na náš nový interní kořenový balíček nazvaný ***pilsner***. Záměrně byl zachován i balíček obsahující GUI implementace, který jak již z předešlých kapitol víme, není prozatím potřebný.

6 Test algoritmu FastICA

V této části semestrální práce se budeme zabývat jednoduchým testováním algoritmu FastICA. Test provedeme v softwaru *Matlab*. Pro *Matlab* existuje FastICA jako *toolbox*[6], který je volně ke stažení na adrese [FASTICAHOME]. V současné době je *toolbox* ve verzi 2.5.

6.1 Popis testovací úlohy

Algoritmus FastICA otestujeme na jednoduchých, uměle vytvořených signálech. Dva signály reprezentují signál artefaktu. Zbylé dva signály mají sinusový průběh. Především půjde o to přesvědčit se, zda-li jsou signály po smíchání dobře algoritmem odděleny.

6.2 Test

Pro test byly vybrány následující signály[7]:

- s_1 = sin(t)
- s_2 = sin(3*t)
- s_3 = 0*t kde

  s3 = 0,6 pro t = 468 a 472

  s3 = 0,8 pro t = 469 a 471

  s3 = 1,0 pro t = 470
- s_4 = 0*t kde

  s4 = -0,6 pro t = 50 a 54

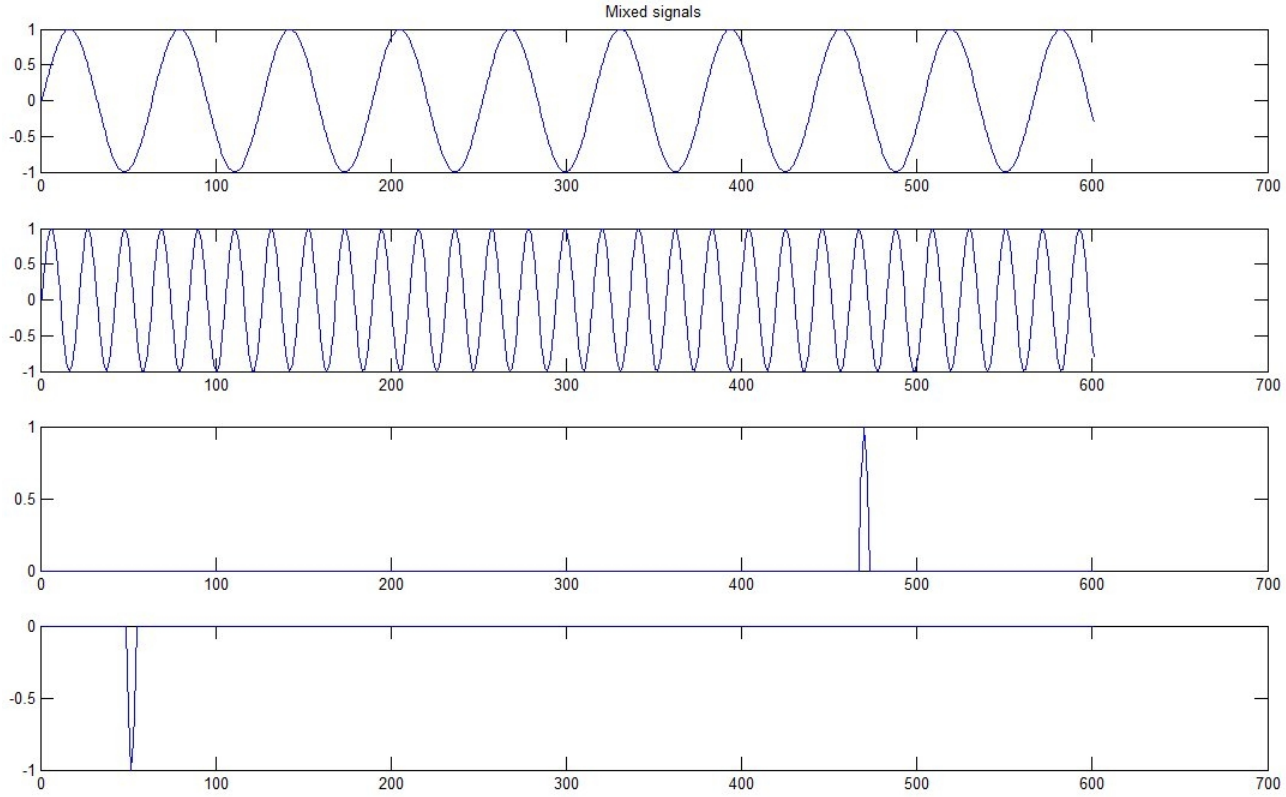
  s4 = -0,8 pro t = 51 a 53

  s4 = -1,0 pro t = 52

Průběh jednotlivých vstupních signálů můžete vidět na obrázku 6.1.

6 Přesný název tohoto toolboxu je : The FastICA package for Matlab.

7 Tyto signály byly vytvořeny pouze pro jednoduchou ilustraci příkladu.

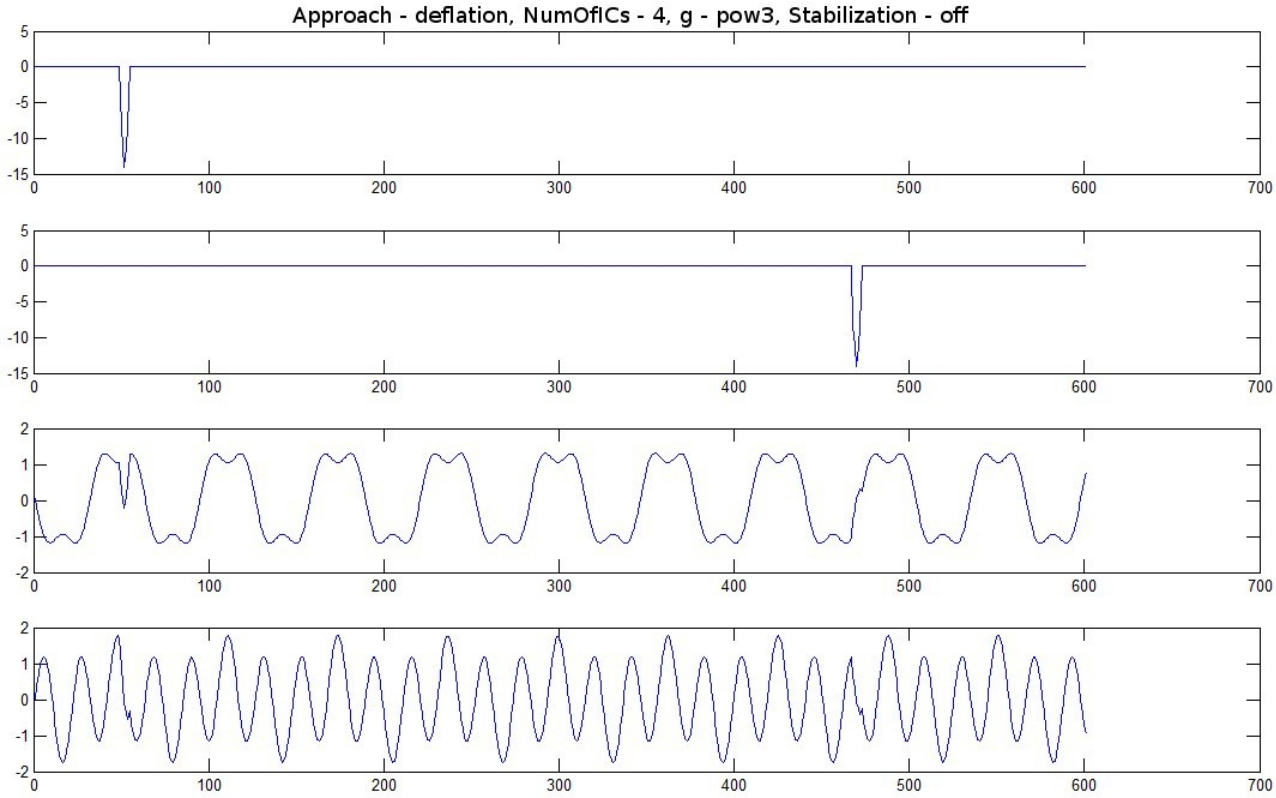


Obrázek 6.1: Vstupní signály (s_1, s_2, s_3 a s_4)

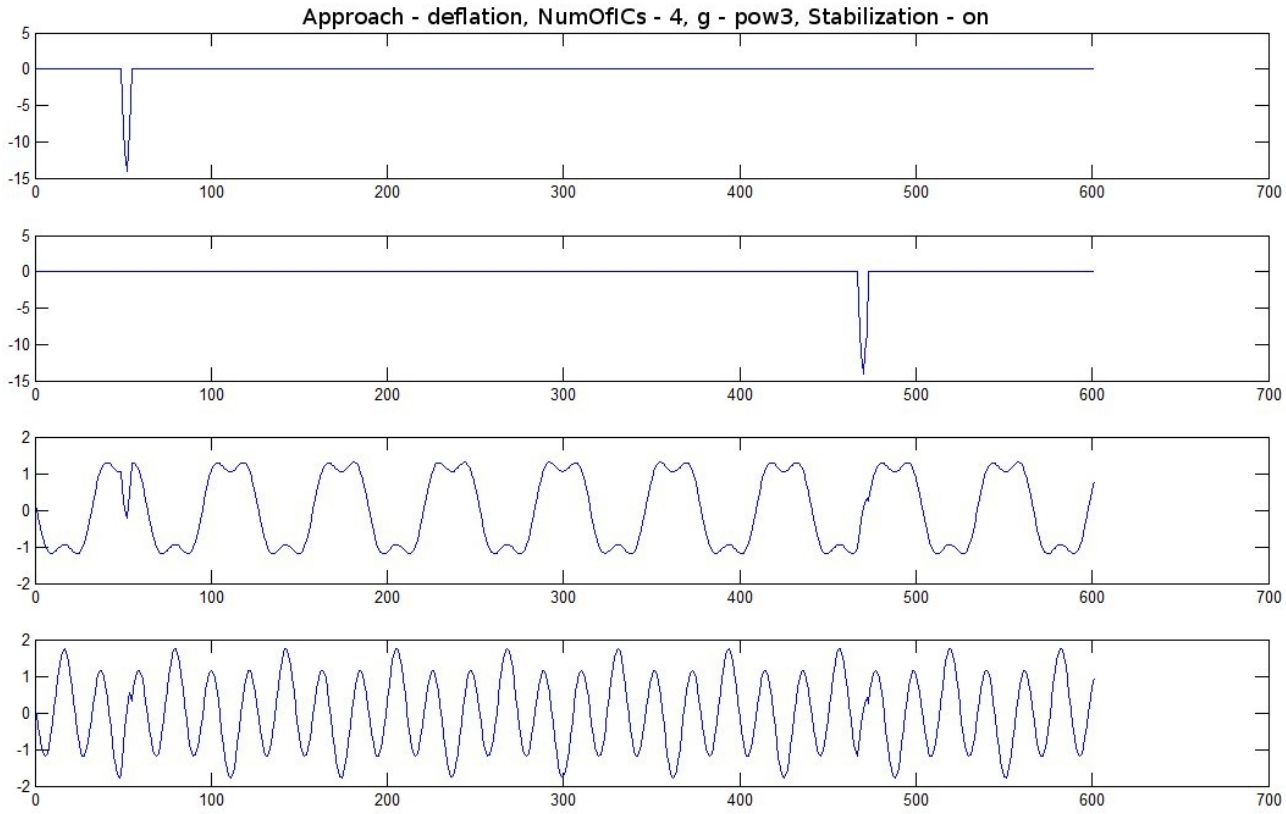
Předtím, než provedeme samotné spuštění algoritmu FastICA, jsou signály automaticky z mixovány a je také provedeno jejich předzpracování (centrování, bělení), kterému jsme se věnovali v kapitole 2.3. Samozřejmě máme také k dispozici parametry algoritmu, jejichž pomocí ho můžeme přizpůsobit podle našich požadavků. My však dopředu nevíme, jaké parametry jsou pro nás nejvhodnější, a tak provedeme test všech možných kombinací. Nyní si ukážeme jednotlivé výsledky experimentu. V tabulce 6.1 jsou v jednotlivých řádcích uvedeny obrázky a k nim v příslušných sloupcích jsou parametry algoritmu FastICA, které byli pro daný výsledný obrázek nastaveny.

	Approach	*Num. Of ICs*	*Nonlinearity*	*Stabilization*
Obrázek 6.2	deflation	4	pow3	off
Obrázek 6.3	deflation	4	pow3	on
Obrázek 6.4	deflation	4	tanh	off
Obrázek 6.5	deflation	4	tanh	on
Obrázek 6.6	deflation	4	skew	off
Obrázek 6.7	deflation	4	skew	on
Obrázek 6.8	deflation	4	gauss	off
Obrázek 6.9	deflation	4	gauss	on
Obrázek 6.10	symmetric	4	pow3	off
Obrázek 6.11	symmetric	4	pow3	on
Obrázek 6.12	symmetric	4	tanh	off
Obrázek 6.13	symmetric	4	tanh	on
Obrázek 6.14	symmetric	4	skew	off
Obrázek 6.15	symmetric	4	skew	on
Obrázek 6.16	symmetric	4	gauss	off
Obrázek 6.17	symmetric	4	gauss	on

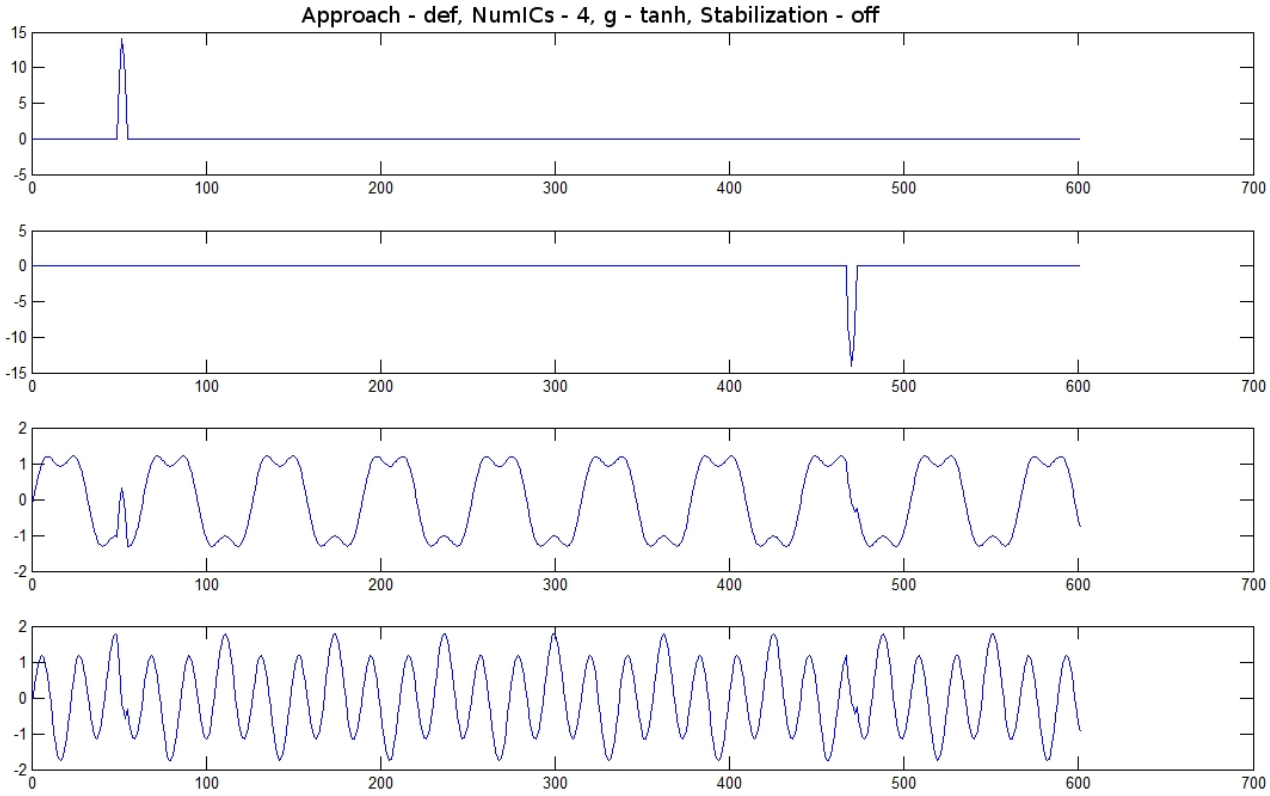
Tabulka 6.1: Kombinace jednotlivých nastavení algoritmu FastICA.



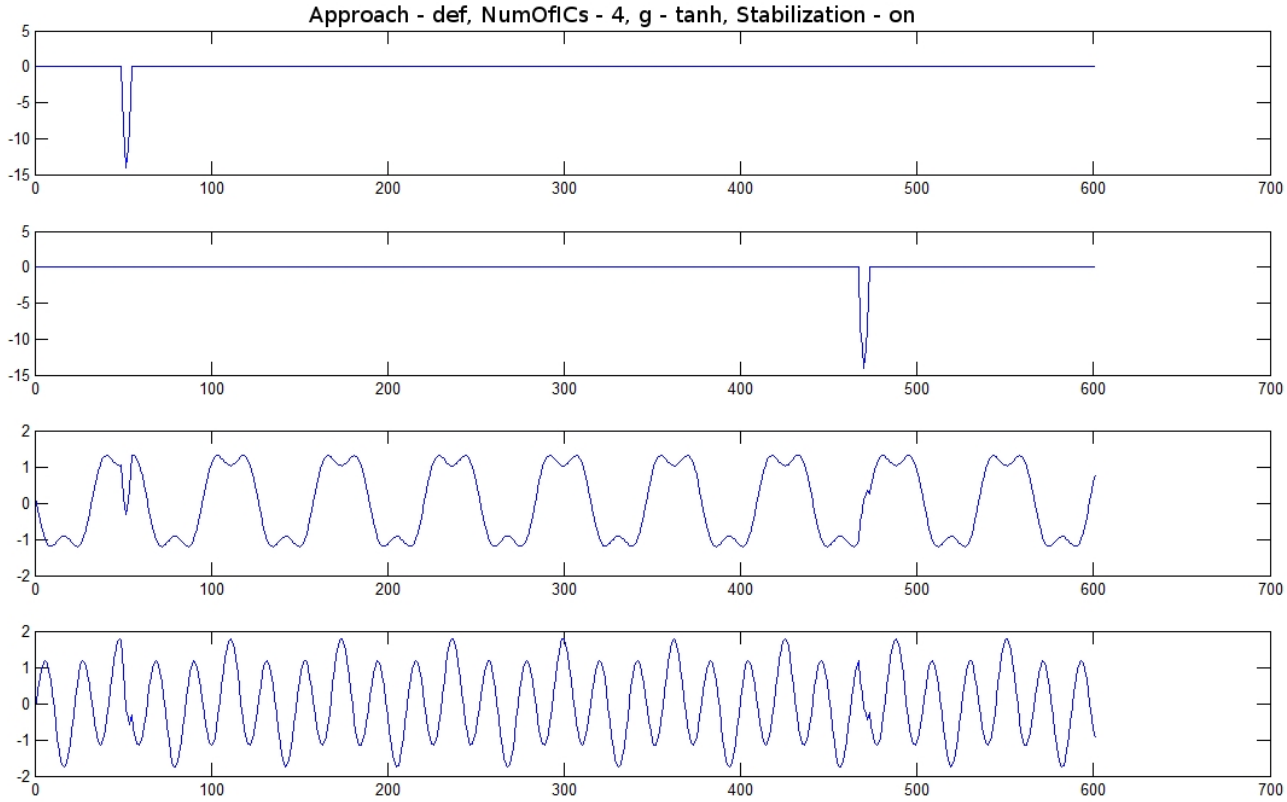
Obrázek 6.2: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



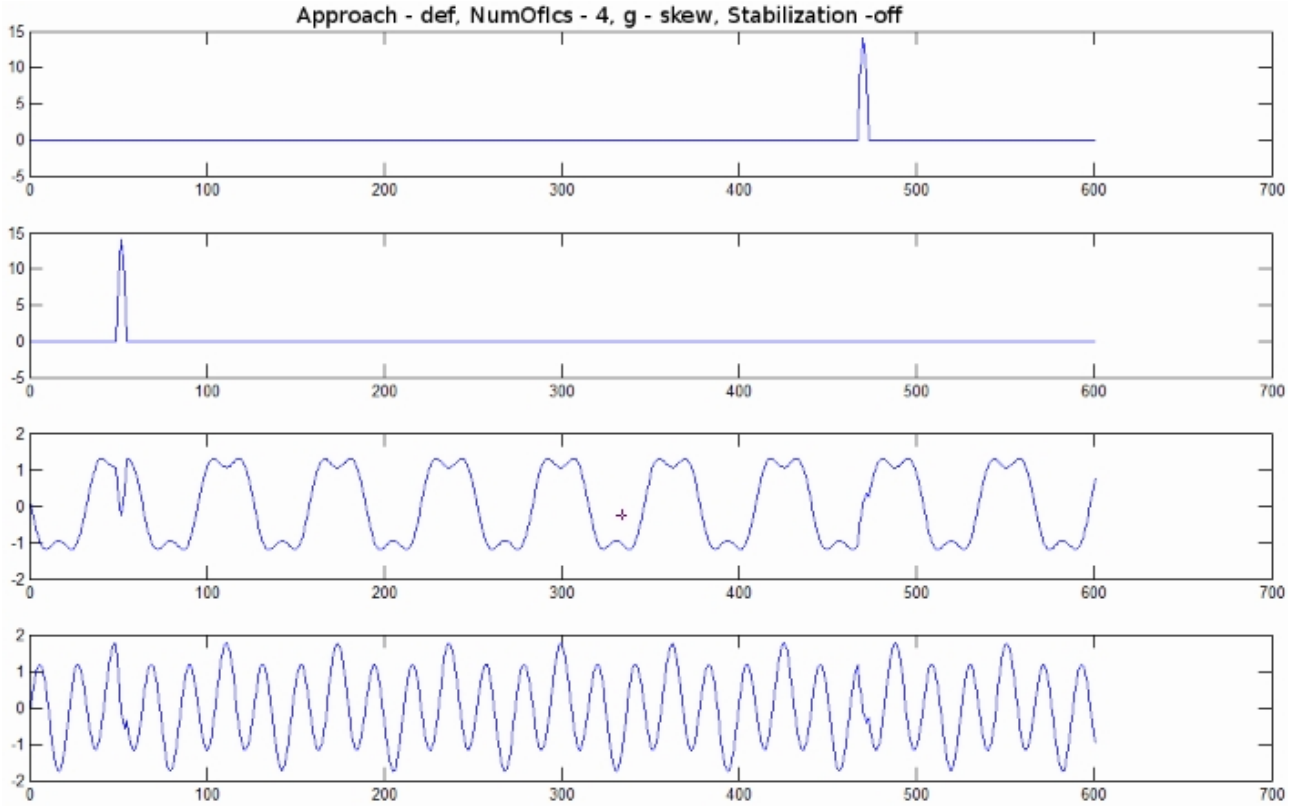
Obrázek 6.3: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



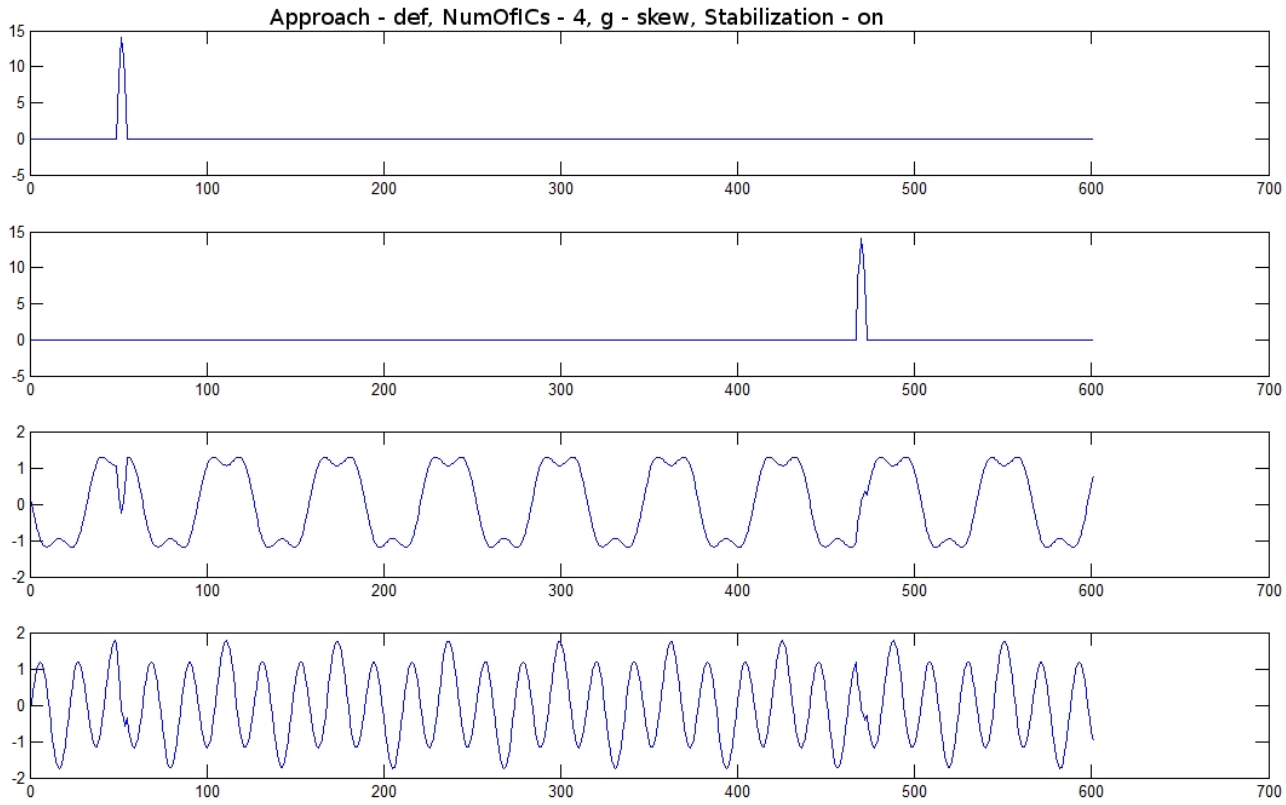
Obrázek 6.4: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



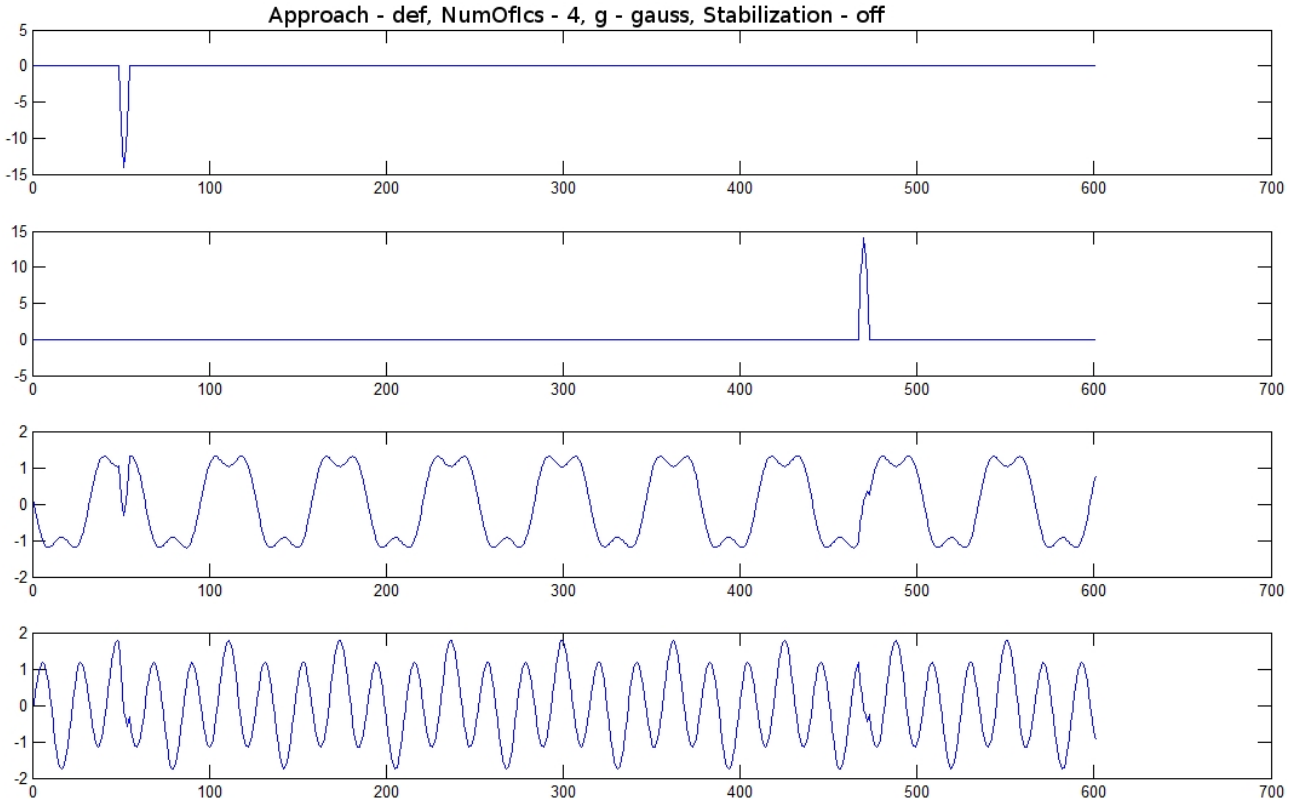
Obrázek 6.5: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



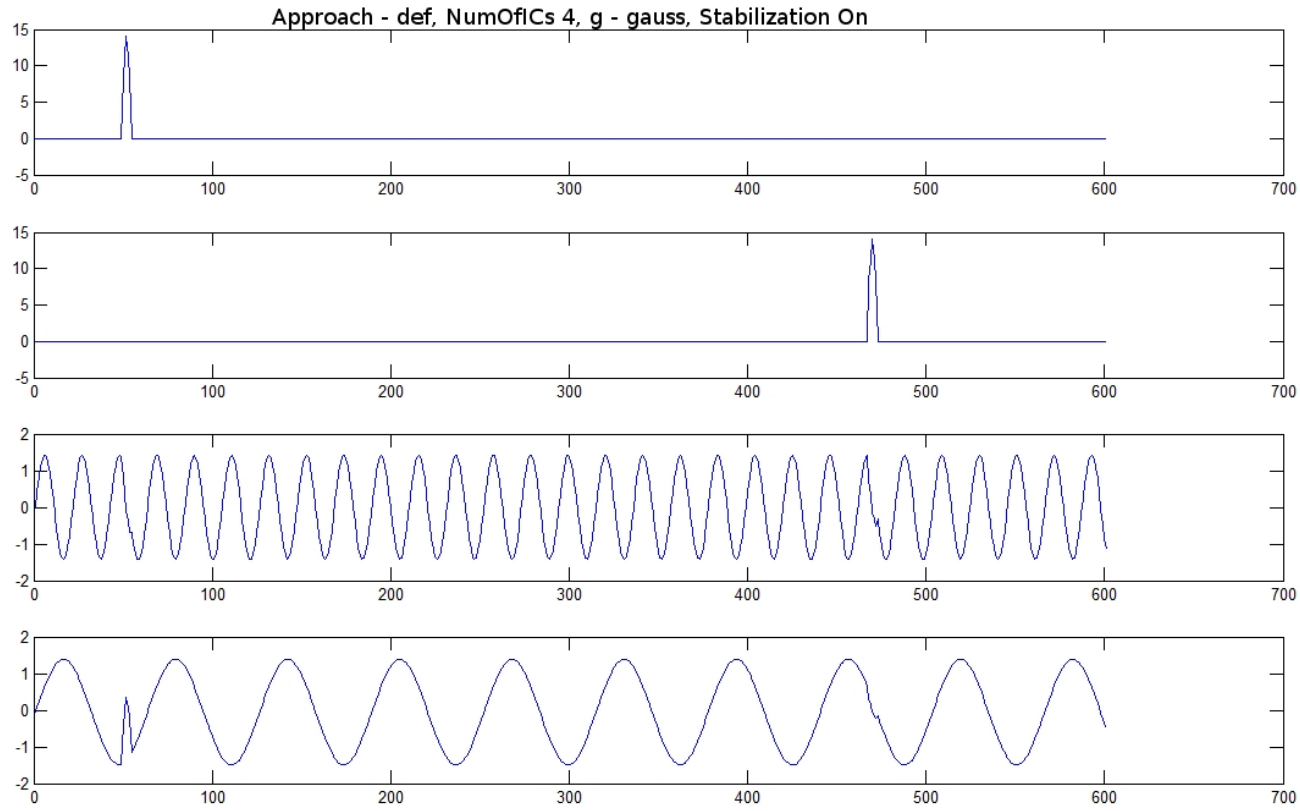
Obrázek 6.6: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



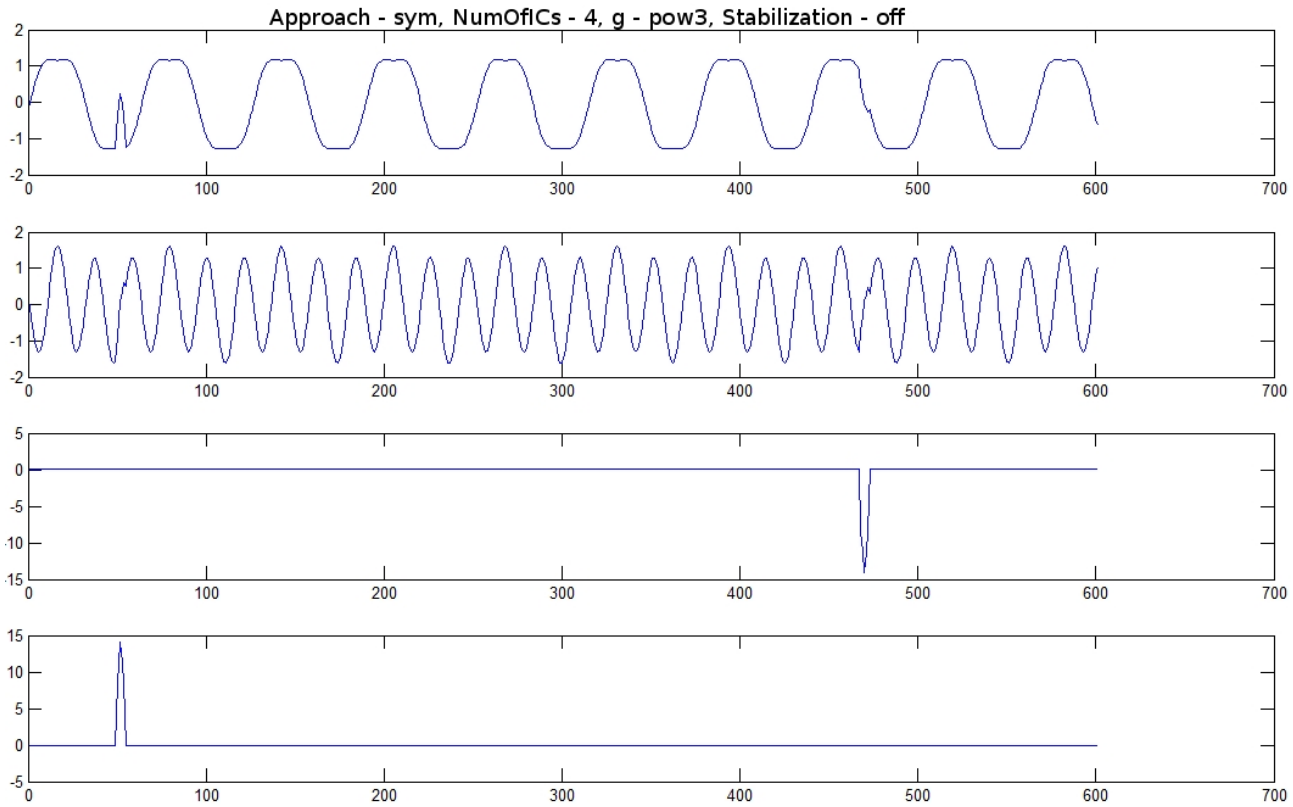
Obrázek 6.7: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



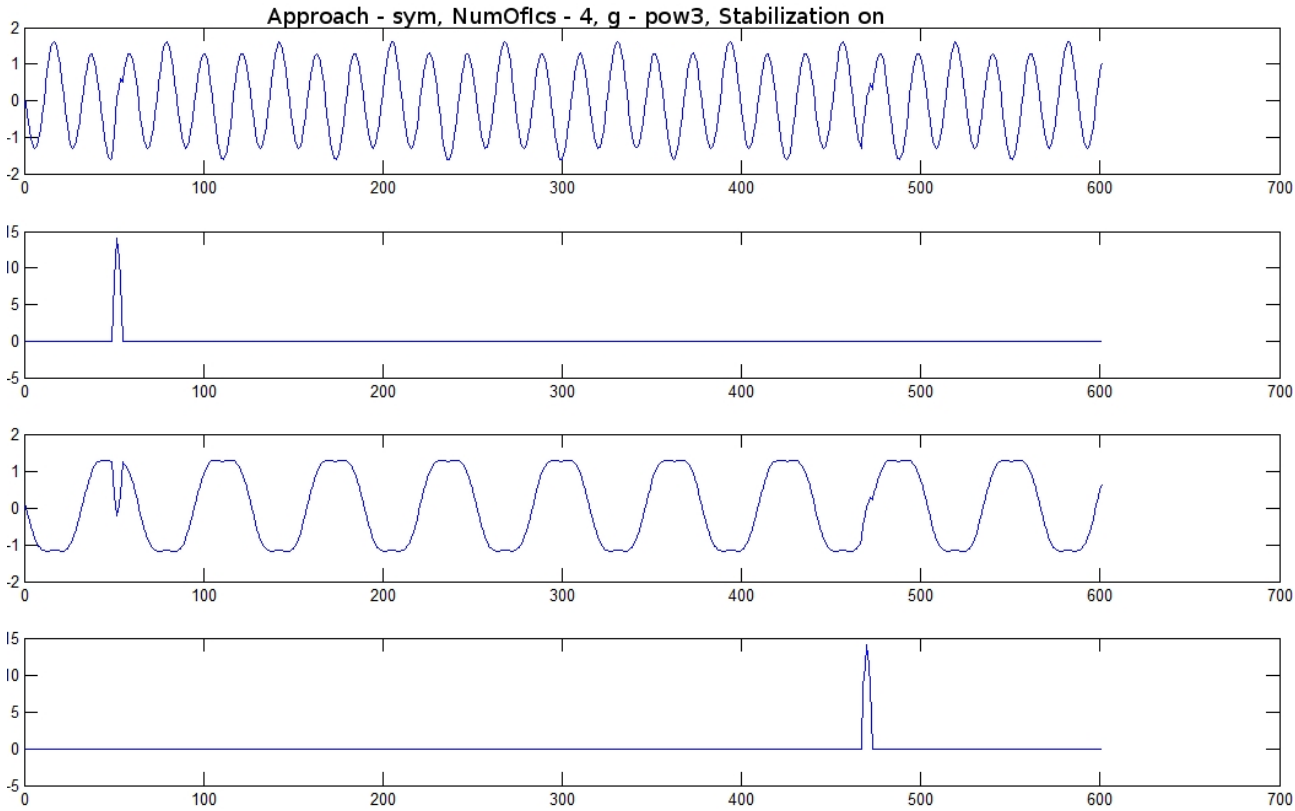
Obrázek 6.8: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



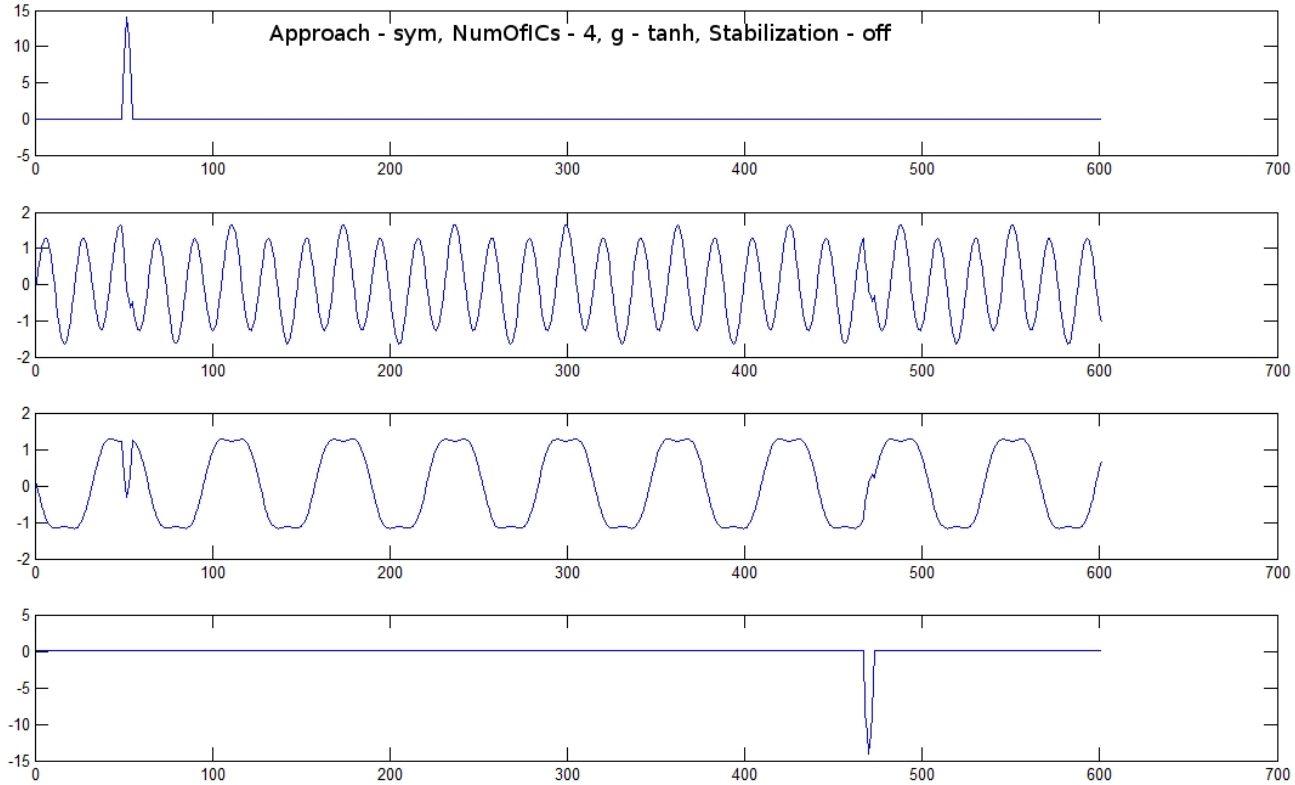
Obrázek 6.9: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



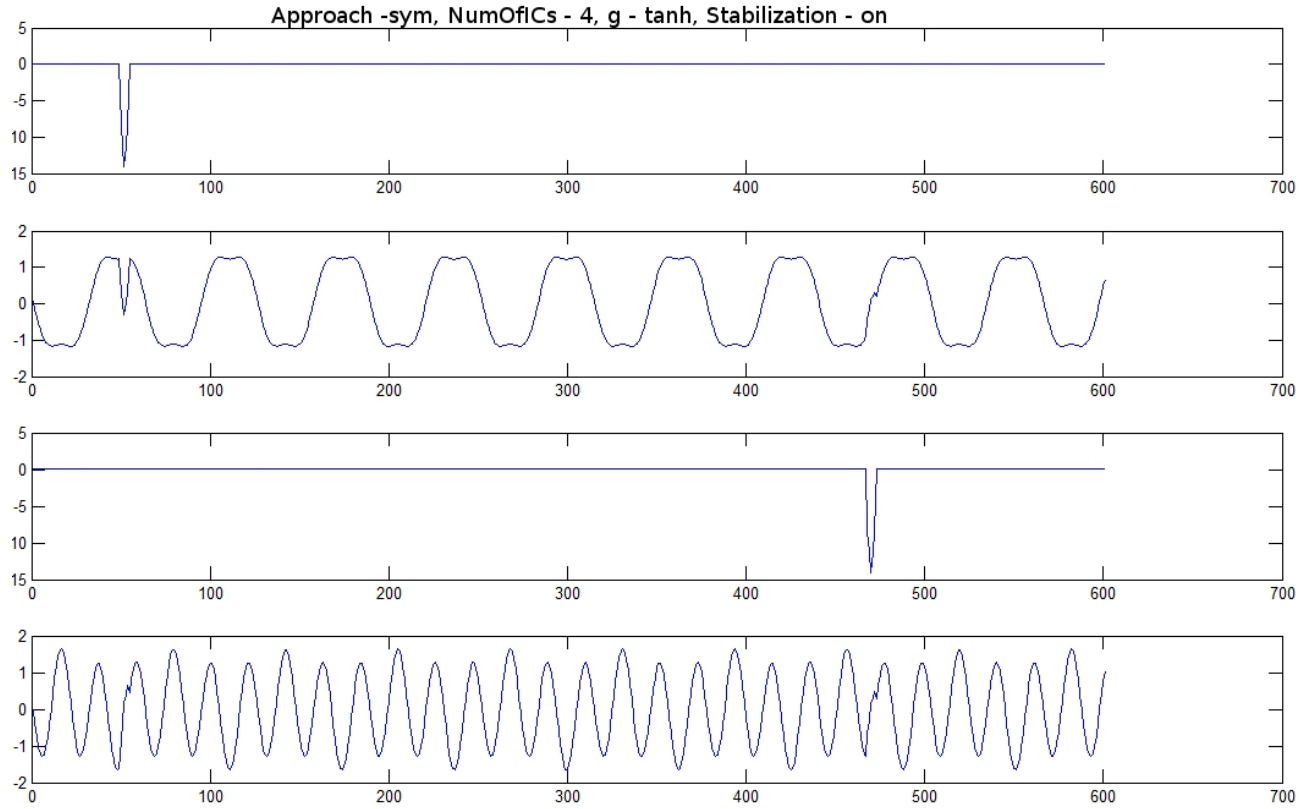
Obrázek 6.10: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



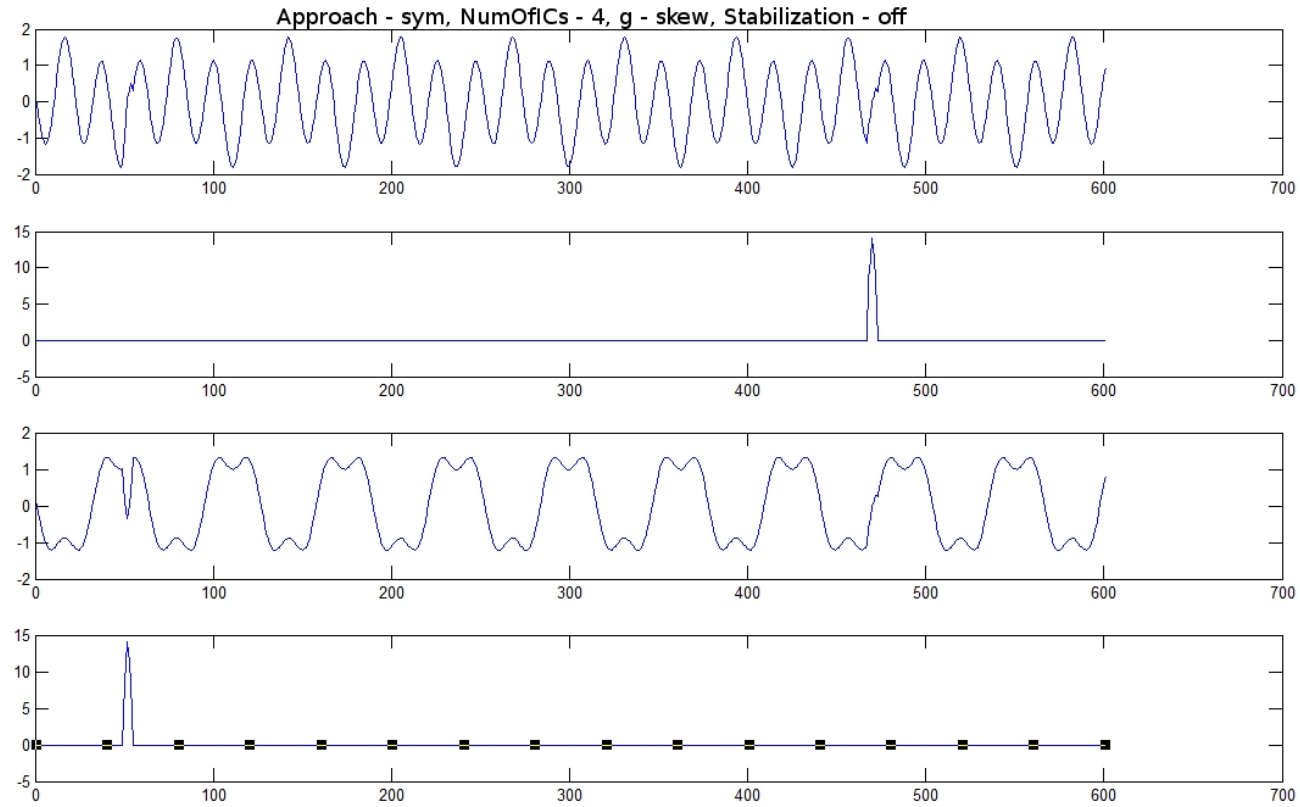
Obrázek 6.11: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



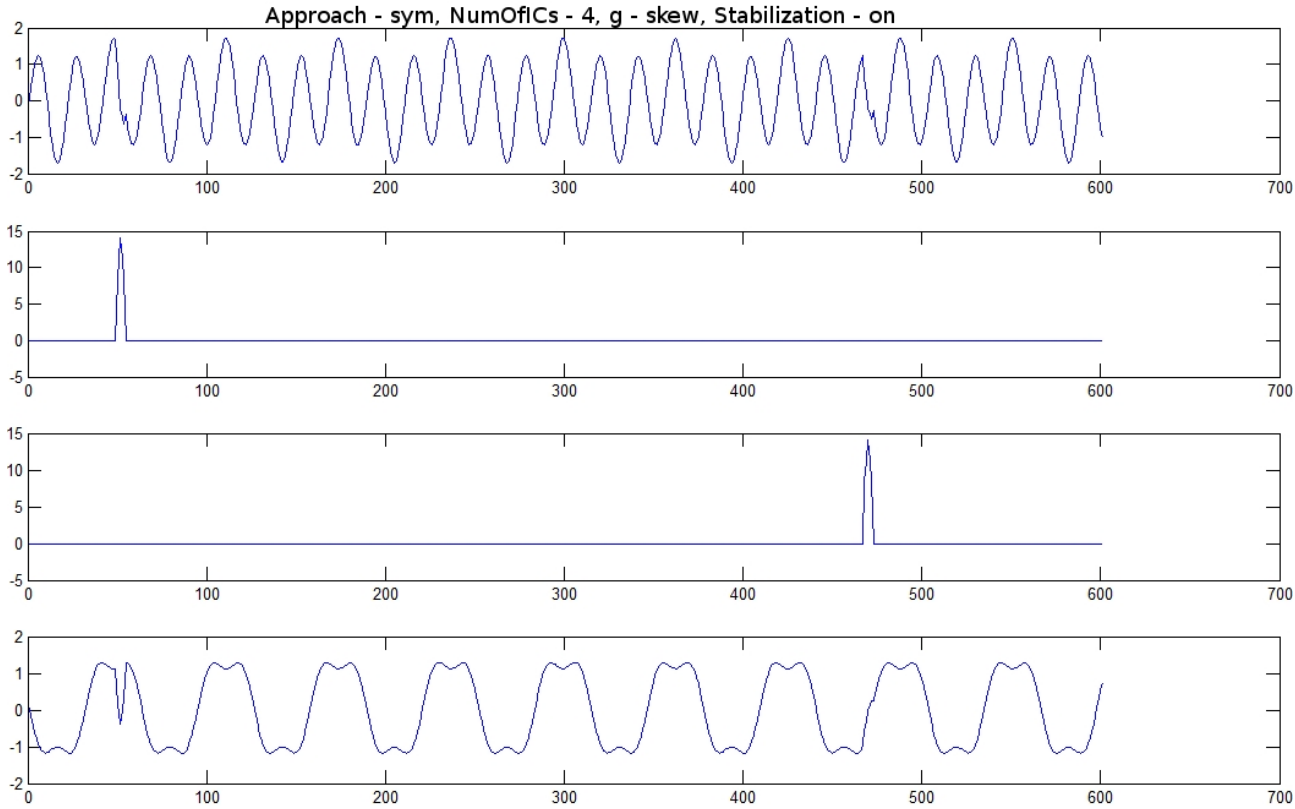
Obrázek 6.12: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



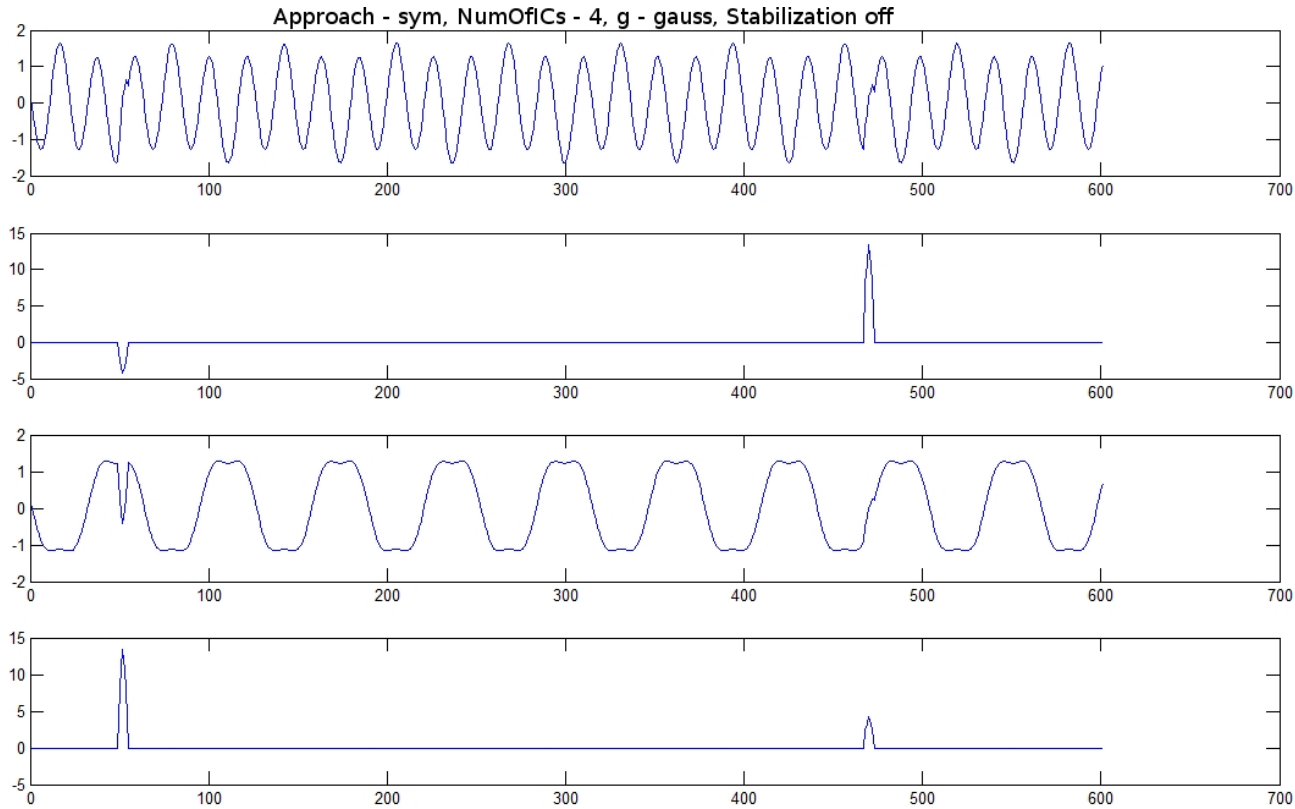
Obrázek 6.13: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



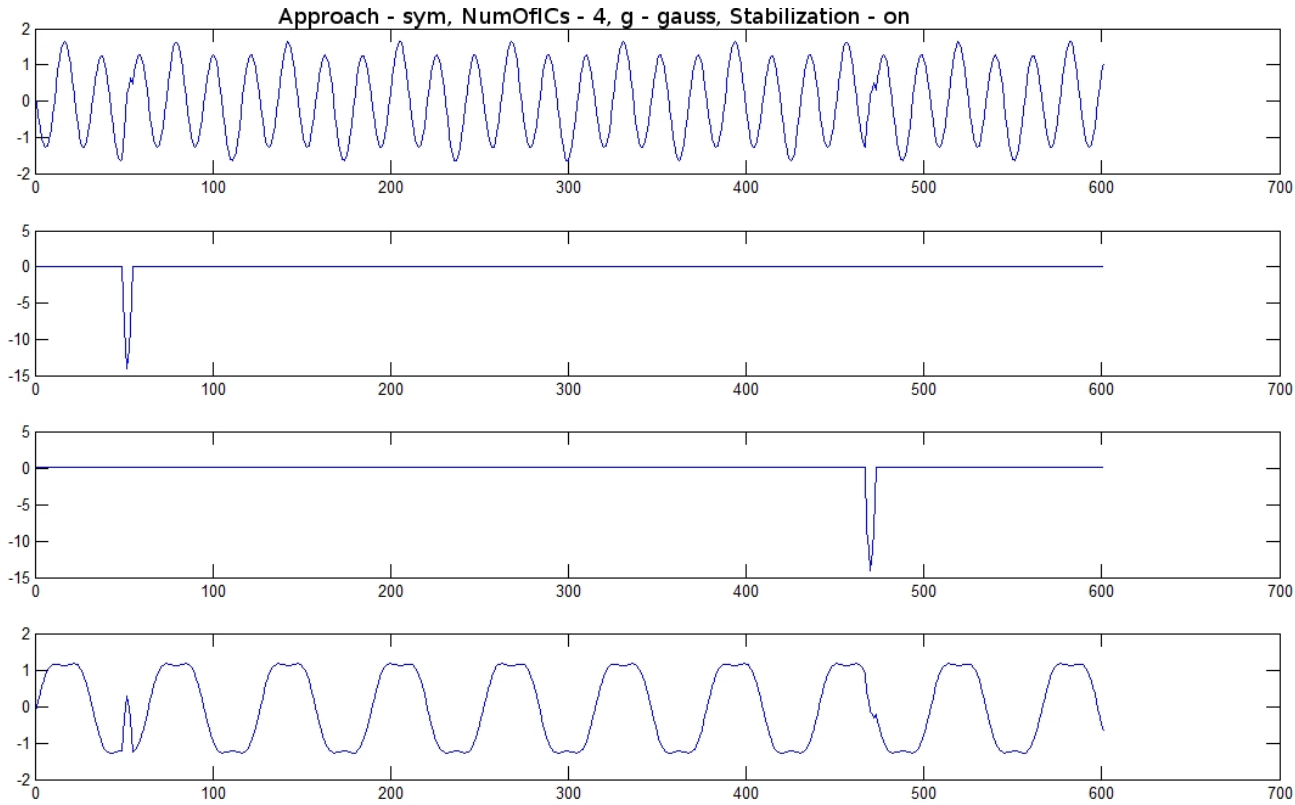
Obrázek 6.14: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



Obrázek 6.15: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



Obrázek 6.16: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)



Obrázek 6.17: Výstup algoritmu FastICA (nastavení viz. Tabulka 6.1)

Nyní si v tabulkách 6.2 a 6.3 shrneme výsledky analýzy nezávislých komponent našeho testu.

	Approach	***Num. Of ICs***	***Nonlinearity***	***Stabilization***	***IC 1***	***IC 2***	***IC 3***	***IC 4***	***Sum***
Obrázek 6.2	deflation	4	pow3	off	5	6	10	2	23
Obrázek 6.3	deflation	4	pow3	on	5	5	5	2	17
Obrázek 6.4	deflation	4	tanh	off	12	6	6	2	26
Obrázek 6.5	deflation	4	tanh	on	10	5	8	2	25
Obrázek 6.6	deflation	4	skew	off	9	5	6	2	22
Obrázek 6.7	deflation	4	skew	on	6	5	6	2	19
Obrázek 6.8	deflation	4	gauss	off	5	4	8	2	19
Obrázek 6.9	deflation	4	gauss	on	31	5	5	2	43

Tabulka 6.2: Počet kroků výpočtu jednotlivých komponent s daným nastavením parametrů

Tabulka 6.2 obsahuje jednotlivé kombinace nastavení parametrů pro algoritmus ***deflation***. V tomto algoritmu se zajímáme o počet kroků analýzy jednotlivých komponent (IC 1 – IC 4). Jak si z tabulky můžeme všimnout, nejlepší separaci jsme získali nelinearitou *pow3* při *zapnuté* stabilizaci. Musíme si však uvědomit nejednoznačnost FastICA a ICA. Nemůžeme tedy na základě tohoto testu

říci, že použijeme danou kombinaci nastavení, ve které jsme dosáhli nejlepšího výsledku. Je totiž nutností provádět testovaní vícekrát na stejných datech a výsledky nakonec zprůměrovat. Pak lze teprve nejlepší nastavení prohlásit za optimální. Proč tak postupovat? Pokud bychom totiž provedli tento pokus znova, a to na stejná data, tak téměř jistě dostaneme trochu odlišné výsledky. To je tedy důvod, proč je potřeba provádět opakování našich testů.

Co je však optimální pro jedny data, nemusí být optimální pro data jiná. Z tohoto důvodu bychom měli provádět vždy nejprve na datech testování různých kombinací, vybrat vhodné nastavení a posléze provést finální separaci.

Jak vidíme v tabulce 6.3, výsledky pro algoritmus ***symmetric*** jsou na zvolených datech experimentu trochu matoucí. Nejlepší konvergence nám vyšla pro nelinearity typu *pow3*, *skew* a *gauss* bez stabilizace. Podle většiny dostupných zdrojů na internetu se však pro *symmetric* nedoporučuje použití nelinearity typu *gauss*. Chtěl bych tady podotknout, že daný experiment je triviálního typu, pouze pro ověření a zjištění, jak algoritmus FastICA (ICA) pracuje. Můžeme se snadno v různých literaturách a zdrojích dozvědět, že velmi časté „***doporučené“*** nastavení je:

- Approach – *symmetric*
- Nonlinearity – *skew*
- Stabilization – *on*

	Approach	***Num. Of ICs***	***Nonlinearity***	***Stabilization***	**Convergence steps**
Obrázek 6.10	symmetric	4	pow3	off	6
Obrázek 6.11	symmetric	4	pow3	on	9
Obrázek 6.12	symmetric	4	tanh	off	7
Obrázek 6.13	symmetric	4	tanh	on	12
Obrázek 6.14	symmetric	4	skew	off	6
Obrázek 6.15	symmetric	4	skew	on	8
Obrázek 6.16	symmetric	4	gauss	off	6
Obrázek 6.17	symmetric	4	gauss	on	12

Tabulka 6.3: Počet kroků dosažení konvergence algoritmu s daným nastavením parametrů.

Samozřejmě, že toto nastavení je pouze doporučené. Pro nejlepší výsledek separace je nutné nejprve provést vlastní testování. Dále si pak můžeme všimnout, že výsledné (separované) signály můžou mít jiné znaménko a také dojde ke změně původního měřítka.

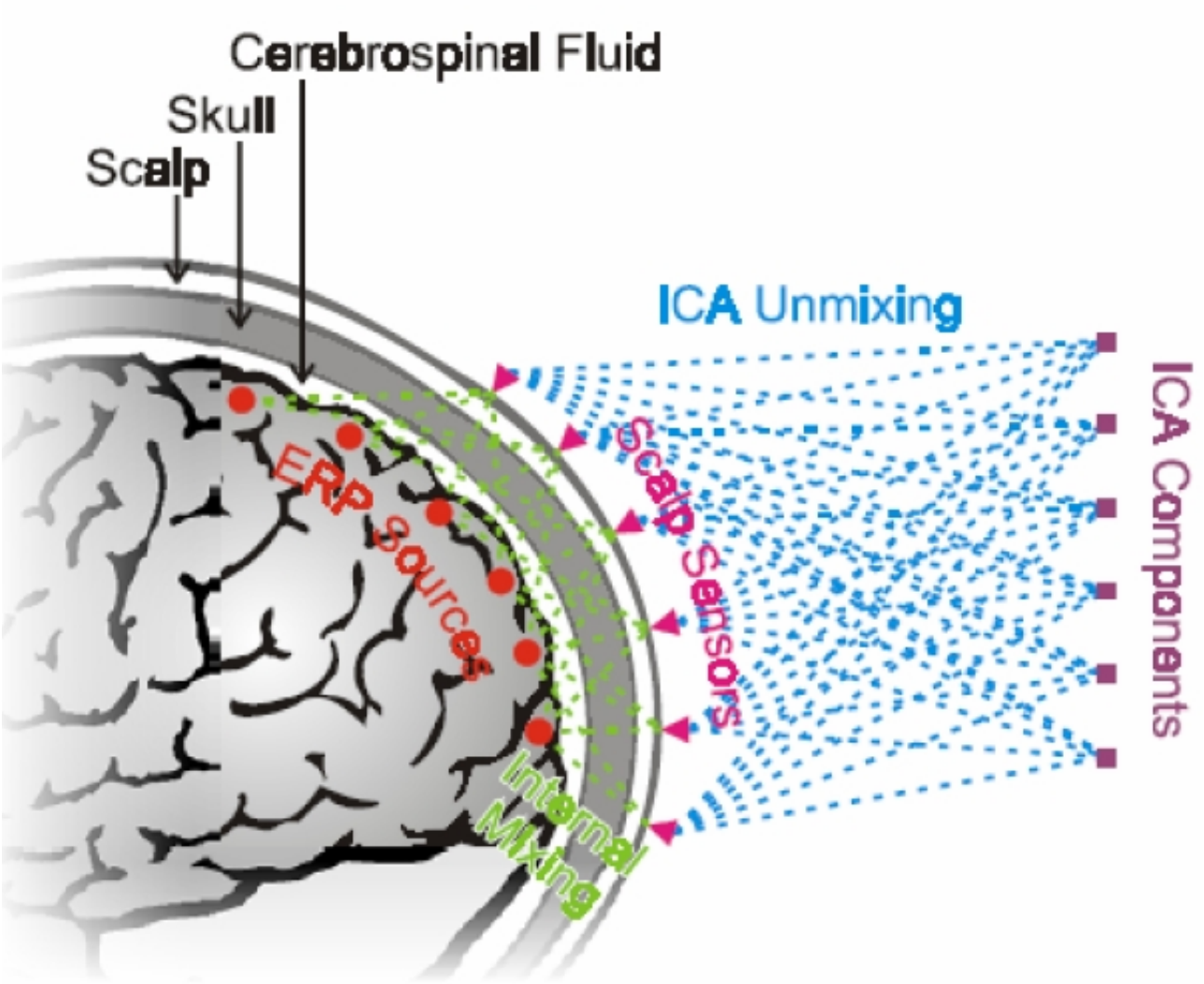
6.3 Postřehy z provedeného testování

V předchozí části jsme si shrnuli dosažené výsledky našeho testu (experimentu) algoritmu FastICA. Nyní si je ještě uveďme jeho důležité vlastnosti, které při experimentech musíme brát v úvahu:

- pořadí nezávislých komponent nelze odhadnout
- znaménko komponent nelze odhadnout
- odhadnuté komponenty mají jednotkový rozptyl (*vlastnost FastICA algoritmu*)

7 FASTICA při detekci a odstranění artefaktů z EEG

Využití nezávislých komponent na EEG data je i v současné době stále předmětem výzkumu, neboť vlastnosti nezávislých komponent nejsou doposud známy. Jednou z nejčastěji diskutovaných oblastí využití nezávislých komponent je problematika odstraňování nežádoucích artefaktů. Model využití ICA komponent na EEG data zobrazuje obrázek 7.1.



Obrázek 7.1: Model (předpokladů) ICA komponent při využití na EEG/ERP data.

Takovou nežádoucí komponentou může být signál obsahující impulzy vzniklé např. mrkáním očí apod.. Odstranit však můžeme také chyby vzniklé např. uvolněním elektrody. K těmto úkonům se v praxi používá tzv. **PCA**[8]. Komponenty získané PCA proto mohou obsahovat mnoho

8 Principal Component Analysis

užitečných informací, které jejich smazáním ztrácíme. **PCA** se tedy používá jako další krok v ***předzpracování*** dat pro ICA (FastICA).

7.1 Proč algoritmus FastICA?

Jak již víme z úvodu 7. kapitoly, k získávání nezávislých komponent z EEG dat existuje řada různých metod (algoritmů), které se liší přesností, výpočetní náročností a efektivností na dané data. Po prostudování několika metod jsem se rozhodl pro použití algoritmu FastICA. Co mě k tomu vedlo? FastICA patří mezi nejrychlejší ICA algoritmy. V efektivnosti a přesnosti se také řadí mezi ostatní spolehlivé algoritmy v této oblasti.

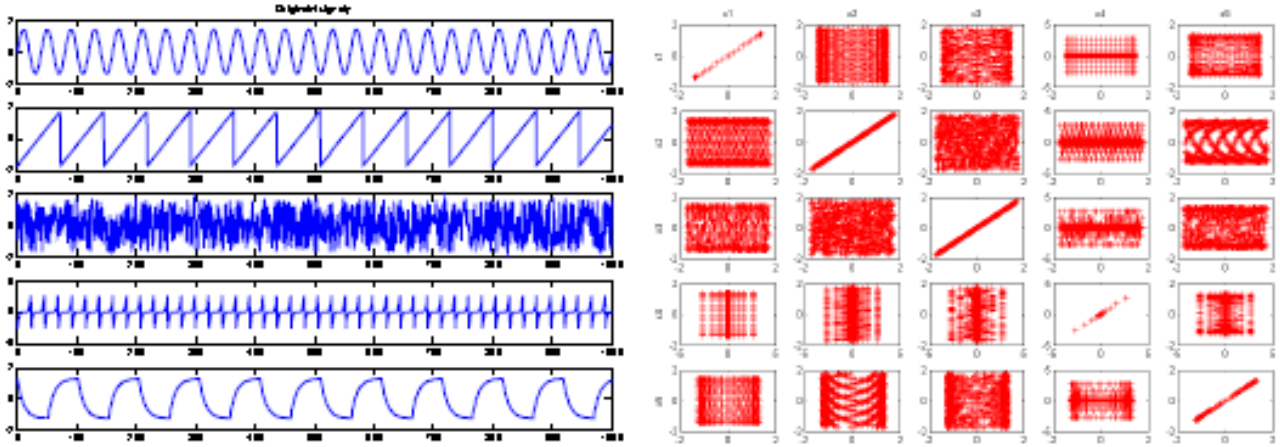
7.2 Předpoklady zpracování EEG prostřednictvím ICA/FastICA

V této části se zmíníme o hlavních předpokladech pro použití ICA na EEG data. Pro lepší představu se po přečtení těchto předpokladů vraťte k obrázku 7. Nyní si tedy uveďme nutné předpoklady:

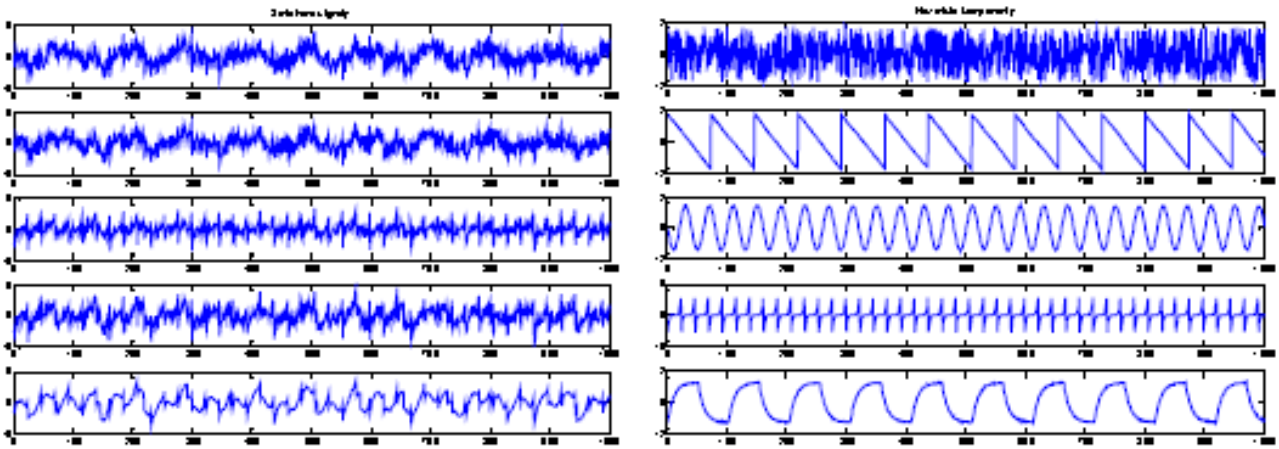
1. Proudy tekoucí z různých zdrojů směrem k snímacím elektrodám jsou mixovány lineárně.
2. Na cestě od zdroje k elektrodě je časové zpoždění signálů zanedbatelné.
3. Počet hledaných nezávislých komponent (IC) je menší nebo roven počtu elektrod.
4. Nezávislé komponenty mají negaussovské rozdělení (ve skutečnosti jedna komponenta IC může mít gaussovské rozdělení).

7.3 Ukázky separací nezávislých komponent

V této části je několik obrázků různých signálů a jejich výsledná separace prostřednictvím ICA a FastICA. Obrázky jsou získané z různých internetových zdrojů.



Obrázek 7.2: Vstupní signály (vlevo) a jejich vzájemná závislost (vpravo).



Obrázek 7.3: Lineární zmixování signálů (vlevo) a výsledná separace alg. FastICA (vpravo).

8 Závěr

Cílem této práce byl výběr vhodné ICA implementace. Vybrán byl algoritmus FastICA. Nejprve jsme se zabývali v 2. kapitole stručným seznámením s *analýzou nezávislých komponent* (ICA). V kapitole 3 jsme se seznámili s algoritmem FastICA [FASTICAHOME] [Hyvärinen Oja]. V kapitole 4 jsme si popsali implementaci algoritmu FastICA v programovacím jazyku Java. Tato implementace se nazývá *FastICA for Java* [FASTICA4JAV]. Kapitola 5 se zabývala stručným popisem úpravy implementace *FastICA for Java* do nově vznikající EEG knihovny, která vzniká v rámci školního výzkumu v oblasti EEG/ERP.

Kapitola 6 se zabývala testováním algoritmu FastICA na uměle vytvořených signálech. Jednalo se pouze o jakési *demo* tohoto algoritmu. Jako první jsme testovali algoritmus FastICA ve verzi **deflation**. Dle našich výsledků nelinearita ***gauss*** poskytla ve verzi *deflation nejhorší* výsledky ze všech. *Nejlepší* výsledky byly dosaženy ve verzi *deflation* nelinearitou ***skew***. V tomto výsledku jsme se shodli s velkou částí jiných, již provedených odborných testů. Tyto testy lze nalézt v různých veřejných zdrojích na internetu. Ve verzi **symmetric** jsme dosáhli *nejlepších* výsledků nelinearitou ***pow3***, ***skew*** a ***tanh***. *Nejhorší* výsledky byly dosaženy (stejně jako u verze deflation) nelinearitou ***gauss***. Nejlepší výsledky nám u nelinearit (ve verzi symmetric) vyšly při vypnuté *stabilizaci*. Tady jsme se dostali trošku do sporu s jinými zdroji. Je však nutné dodat, že analýza nezávislých komponent nefunguje na všechny data stejně, a tak se nelze divit, že se náš test nemusí vždy shodovat s jinými testy. To je jedna z nejednoznačností ICA (FastICA) algoritmu.

V 7. kapitole jsme se zabývali úvodem do oblasti využití ICA při detekci a odstraňování artefaktů v EEG datech (záznamech). Naměřená EEG data totiž splňují obecný lineární model, na kterém je postavena analýza nezávislých komponent (ICA). Z tohoto předpokladu vyplývá, že ICA je v této oblasti využitelná, a že může být spolu s PCA velkým přínosem a pomocníkem.

Na základě poznatků z této semestrální práce lze obecně *říci*, že *nejvhodnější* na EEG data je použití FastICA ve verzi **symmetric**. Nelinearita je již trochu nejednoznačná. Došel jsem v prováděném experimentu k nelinearitám *pow3*, *skew* a *tanh*. Hodně se doporučuje linearita *skew.* V tomto případě nám nezbude nic jiného, než provádět další experimenty. Můžeme však vynechat testování ve verzi *deflation*, která přináší mnohem horší výsledky než verze *symmetric*.

Díky této semestrální práci jsem si osvojil problematiku algoritmu ICA a především jeho modifikace zvané FastICA. Jedná se o velmi zajímavý algoritmus, i když po matematické stránce byl pro mě docela tvrdým oříškem.

Seznam použité literatury

[CICHOCKI AMARI]: Andrzej Cichocki, Shun-ichi Amari, Adaptive Blind Signal and Image Processing, 2007

[Hyvärinen Oja]: Aapo Hyvärinen, Errki Oja, Independent Component Analysis: Algorithm and Application, 2000

[FASTICAHOME]: , FastICA , , http://www.cis.hut.fi/projects/ica/fastica/index.shtml

[FASTICA4JAV]: , FastICA for JAVA, , http://www.fastica.org/