Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Úprava knihovny Hilbert-Huangovy transformace

Plzeň 2016 Michal Veverka

Poděkování

Rád bych poděkoval Ing. Tomáši Prokopovi za odborné vedení, pomoc a rady při zpracování této práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 29. dubna 2016

Michal Veverka

Abstract

Modification of the Hilbert-Huang transform library

The EEG signal evaluation is a frequent and difficult task. The used methods include the Wavelet and Fourier transform. These methods are mainly suitable for stationary and linear data. However, the signal obtained during EEG measurement is neither stationary nor linear. The Hilbert-Huang transform is an adaptive method developed for processing such data. It consists of two parts: Empirical Mode Decomposition (EMD) and the Hilbert transform. But there is a serious mode mixing phenomenon in EMD. To solve this problem, EEMD was developed. The HHT library was developed at the University of West Bohemia to examine its usage in the EEG processing.

I have added the EEMD to the library, tested it using real EEG data. I have also optimized our implementation of the EEMD and EMD method to achieve better performance.

Vyhodnocování EEG signálu je častým a nesnadným úkolem. Mezi používanými metodami je např. Waveletová a Fourierova transformace. Tyto metody jsou vhodné pro stacionární a lineární data. Signál získaný z EEG měření však není ani stacionární, ani lineární. Hilbert-Huangova transformace je adaptivní metoda navržená pro zpracování takových dat. Skládá se ze dvou částí: Empirické modální dekompozice (EMD) a Hilbertovy transformace. Při použití EMD však dochází k tzv. "Mode mixing" problému. K vyřešení tohoto problému byla vytvořena metoda EEMD. Knihovna HHT vznikla na Západočeské univerzitě pro účely testování HHT při zpracování EEG signálu.

Implementoval jsem do knihovny metodu EEMD a otestoval ji na reálných EEG datech. Také jsem optimalizoval metody EEMD a EMD pro dosažení lepší výkonnosti.

Obsah

1 Úvod

Elektroencefalografie (EEG) je důležitá lékařská vyšetřovací metoda, která hraje nezastupitelnou roli při diagnozách a následné léčbě závažných onemocnění (např. epilepsie) a také v řadě výzkumů. I přes vznik moderních vyšetřovacích metod, jako jsou počítačová tomografie (CT) a magnetická rezonance (MRI), se stále těší širokému užití. To je dáno zejména vysokým časovým rozlišením, nízkou cenou potřebného vybavení a přenositelností měřících přístrojů. Častým cílem vyhodnocení EEG je, zejména při experimentech, detekce evokovaných potenciálů (ERP). Kvůli způsobu měření obsahuje výsledný signál řadu nechtěných částí a detekování ERP je nesnadným úkolem. Používanými metodami jsou např. Waveletová transformace a Fourierova transformace, které však rozkládají signál do předem daných funkcí a jsou vhodné zejména pro lineární a stacionární data. EEG data jsou však obecně nelineární a nestacionární. Metoda navržená pro zpracování nelineárních a nestacionárních dat je Hilbert-Huangova transformace, která rozkládá signál Empirickou modální dekompozicí (EMD) do tzv. Intrinsic mode functions (IMF). Tato metoda je adaptivní, jelikož IMF jsou dány zpracovávanými daty. Jejím častým problémem je však tzv. *mode mixing*, při kterém není signál správně rozložen do jednotlivých IMF. Možným řešením tohoto problému je modifikace metody EMD, při které je využito bílého šumu, nazvaná Ensemble empirical mode decomposition (EEMD).

Prvním cílem této práce je uvést čtenáře do světa EEG/ERP experimentů a HHT a seznámit čtenáře s knihovnou HHT. Ta vznikla jako součást diplomové práce J. Ciniburka v roce 2011. Druhým cílem je tuto knihovnu optimalizovat a doimplementovat do ní metodu EEMD. Nakonec je nutné otestovat navrženou optimalizaci a implementaci metody EEMD na reálných EEG datech a výsledky porovnat.

2 Elektroencefalografie

Elektroencefalografie (EEG) je metoda elektrofyziologického vyšetření, při které dochází k měření časové změny elektrického potenciálu, způsobeného mozkovou aktivitou, na povrchu lebky. Elektrická aktivita mozku je důsledkem pohybu elektrogenních iontů přes buněčné membrány, v jejichž okolí vzniká elektromagnetické pole. Elektrický potenciál jednoho neuronu je však příliš slabý na to, aby byl zachycen při EEG měření. Při měření tak dochází k sumaci několika milionů jednotlivých neuronů, přičemž velikost elektrické aktivity jednotlivých neuronů závisí na jejich vzdálenosti od elektrody. Neurony, které se nacházejí blíže elektrodě, se na konečném výsledku měření projeví více, zatímco vzdálenější neurony se projeví méně. Z těchto důvodů nelze rozlišit elektrickou aktivitu jednotlivých neuronů. Výsledkem takového měření je elektroencefalogram - grafický záznam průběhu potenciálových rozdílů elektrického pole mozku. Vlivem nízké vodivosti lebky je získaný signál velice slabý, v průměru mezi 10 μV a 100 μV.

S nástupem moderních zobrazovacích metod (CT, MRI) význam EEG ustupuje. Díky řadě výhod, např. malé velikosti potřebných zařízení a tedy jejich snadnou přenositelností, malé ceně a snadné použitelnosti, však v některých oblastech stále najde využití. Vyšetření EEG se nejčastěji používá v neurologii a psychiatrii, při diagnostice a monitorování epilepsie, kóma, migrény, mozkové smrti či hloubky anestezie a spánku. Hojně je však využíváno i při experimentální a vědecké činnosti, kdy je sledována a zkoumána činnost mozku.

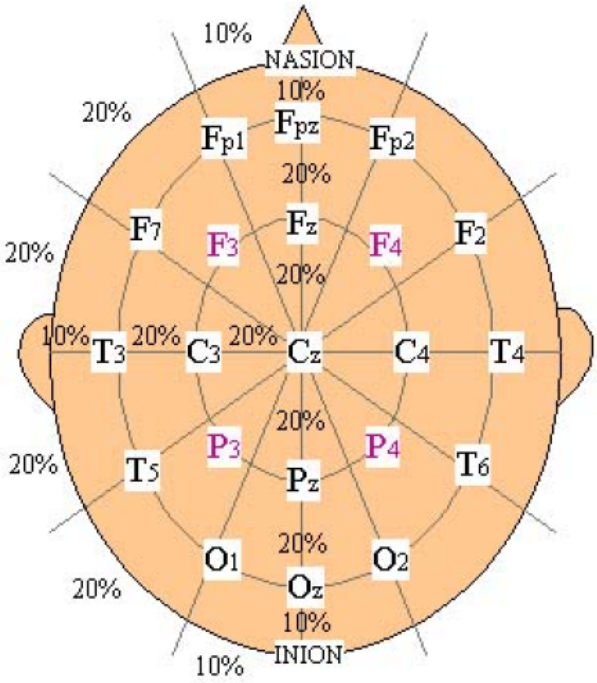
2.1 Měření EEG signálu

Vybavení potřebné k měření EEG signálu se z pravidla skládá z elektrod, diferenčního zesilovače, A/D převaděče a zaznamenávacího a zobrazovacího zařízení. Pro dosažení co nejlepších výsledků je nejprve třeba očistit povrch hlavy, poté se na určená místa nanese vodivý gel a přiloží elektrody, které budou číst EEG signál. Diferenční zesilovač zesílí signál z μV tak, aby mohl být následně digitalizován A/D převaděčem. Nakonec jsou data uložena a zobrazena na zaznamenávacím zařízení, typicky stolním počítači.

Elektrody se na povrch hlavy umisťují podle mezinárodního standardizo-

vaného systému nazvaného 10-20 (viz. obrázek 2.1). Povrch hlavy je rozdělen tak, aby elektrody rovnoměrně pokrývaly všechny důležité části mozku. V běžné klinické praxi se používá 19 elektrod na povrchu lebky a 2 elektrody připevněné na ušních lalůčcích. Při některých experimentech se využívá i menšího počtu elektrod, v některých případech například pouze 3 na povrchu hlavy, 1 na ušním lalůčku a 1 na obočí. Správná funkčnost elektrod

Obrázek 2.1: Rozmístění elektrod podle mezinárodního standardu 10-20. [Teplan(2005)]



je pro výsledek měření velmi důležitá. Vysoká impedance mezi elektrodou a povrchem hlavy může vést ke zkreslení výsledného signálu, proto by před začátkem měření měla být každá elektroda zkontrolována, zda je její impedance nižší než 5000 ohmů. Jelikož je navíc kvůli slabé vodivosti lebky amplituda výsledného signálu velmi malá (v řádech μV), je třeba signál zesílit pomocí zesilovače tak, aby mohl být dále zpracován A/D převaděčem a posléze zobrazovacím zařízením. Správně navržené zesilovače dokáží ze signálu odstranit nechtěné části, např. ruch elektrické sítě a ruch způsobený kontaktem kůže a elektrod. Takto získaný analogový signál je dále převeden A/D převaděčem na signál digitální. Po uložení dat můžeme použít digitální filtrování a jiné metody (FIR, IIR, HHT), které signál ještě více vyčistí a získat tak přesnější

data.

2.2 Základní mozková aktivita

Výsledný elektroencefalogram má za normálních okolností vlnovitý tvar. V průběhu času se ukázalo, že pro některé stavy měřené osoby mají tyto vlny stálé vlastnosti, zejména frekvenci a amplitudu. Došlo tedy ke klasifikaci těchto vzorů a k jejich rozdělení do tříd. Základní rytmy (obrázek 2.2) jsou uvedeny zde:

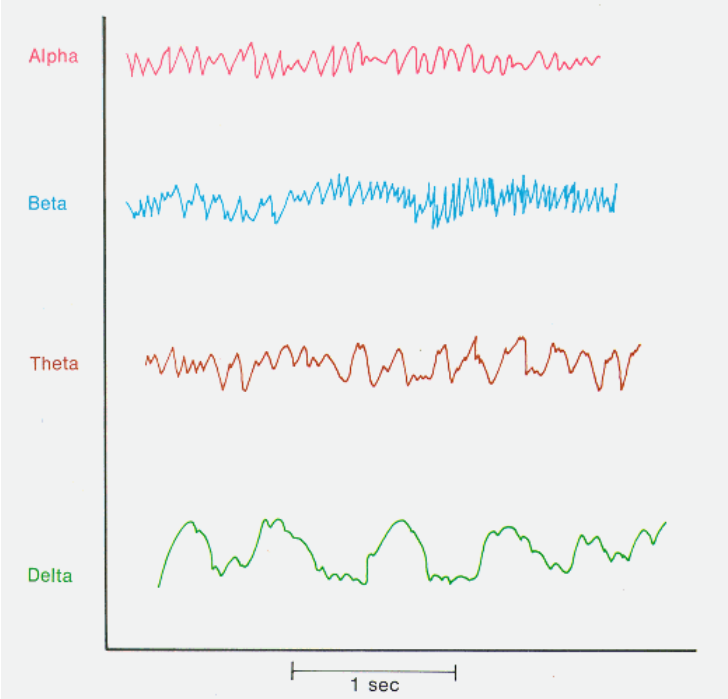
- Alfa rytmus je charakteristický pro EEG záznam bdělé osoby v relaxovaném stavu se zavřenýma očima. Amplituda vln nabývá hodnot v rozmezí od 5 do 100 μV a je výraznější nad okcipitálními oblastmi mozku. Frekvence se pohybuje mezi 8-13 Hz. Při mentální činnosti, soustředění nebo otevření očí dochází k přechodu alfa vln na vlny beta.
- U osob v bdělém stavu s otevřenýma očima, jejichž mozek vykazuje zvýšenou pozornost či soustředěnost, se vyskytuje rytmus beta. Charakteristická je frekvence vyšší než 13 Hz a amplituda v rozmezí od 10 do 20 μV. Beta vlny se rovněž vyskytují v REM fázi spánku.
- Théta vlny jsou přítomny na EEG záznamu během spánku. Jejich frekvence se pohybuje v rozmezí od 3 do 7 Hz.
- Delta vlny se stejně jako vlny théta vyskytují během spánku, ovšem jejich počet přibývá se zvětšující se hloubkou spánku. Delta rytmus má v porovnání s ostatními vlnami velice nízkou frekvenci od 1 do 4 Hz a vysokou amplitudu vln od 20 až do 200 μV.

Kromě těchto rytmů se na EEG záznamu objevují ještě další, méně časté vzory, např. gama, lambda a mí.

2.3 Evokované potenciály

Evokované potenciály (Event-related potencials) jsou části EEG signálu, které představují reakci mozku na určitý audiovizuální podnět (stimul). Je to časový úsek signálu, vázaný na dobu vzniku daného stimulu. Jejich amplituda

Obrázek 2.2: Alfa, beta, théta a delta EEG rytmy. [EL-Bab(2001)]



je velmi malá a při měření je proto potřeba provést několik opakování a výsledky zprůměrovat. Eliminují se tím také vlivy některých artefaktů (viz kapitola 1.5.1 Artefakty). Potenciály se rozlišují do několika komponent, podle jejich amplitudy a zpoždění vůči stimulu. Podle orientace amplitudy se značí symboly P (pozitivní), N (negativní), C (proměnlivá). Číslem se značí podle pořadí vlny, např. P3 představuje třetí pozitivní ERP vlnu po vyslání stimulu. Podle původu stimulu se komponenty rozdělují do několika skupin, např. zrakové, sluchové a další. Komponenty z různých skupin mohou sdílet stejné označení, např. P3, to však neznamená, že mají stejné vlastnosti.

2.3.1 Zrakové stimuly

C1 je první hlavní komponentou. Pozitivita její amplitudy se může měnit, proto se k jejímu označení nepoužívají písmena P a N, nýbrž C. Typicky začíná mezi 40-60ms a vrcholí mezi 80-100ms.[Luck(2005)]

P1 následuje za vlnou C1. Dosahuje nejvyšší amplitudy na laterální okcipitální elektrodě, začíná mezi 60-90ms a vrcholí mezi 100-130ms. [Luck(2005)]

Vlna **N1** se objevuje po vlně P1 a bývá obvykle rozdělena do několika sub-komponent. První sub-komponenta vrcholí mezi 100-150ms, poté následují alespoň dvě další sub-komponenty, které vrcholí mezi 150-200ms. [Luck(2005)]

P2 navazuje na vlnu N1. Často bývá špatně odlišitelná od vln N1, N2 a P3. Zatím toho však o ní není mnoho známo. [Luck(2005)]

2.3.2 Sluchové stimuly

Velmi brzké ERP vlny se mohou objevovat již v prvních 10ms od zvukového stimulu. Jsou generovány sluchovými cestami v mozkovém kmeni.

Sluchová komponenta **N1** má stejně jako ta zraková několik sub-komponent, přičemž první z nich má největší amplitudu kolem 75ms po vyslání stimulu. Druhá sub-komponenta vrcholí kolem 100ms a třetí kolem 150ms. Zpoždění této vlny může být ovlivněno pozorností měřené osoby. [Luck(2005)]

Vizuální i sluchové komponenty třídy **N2** byly hojně studované a výzkumníci nalezli hned několik sub-komponent spadajících do této třídy. Jedná se o vlny s negativní amplitudou nacházející se mezi 200ms a 350ms po vyslání stimulu. Amplituda komponent této třídy je nižší, pokud se stimuly vyskytují častěji.[Luck(2005)]

Ve třídě **P3** existuje několik rozlišitelných komponent. Dvě hlavní komponenty jsou P3a a P3b. Amplituda komponenty P3b je větší, pokud se stimulus opakuje mezi podněty méně a objevuje se nečekaně. Vlna P3 se objevuje poté, co byl stimulus zpracován a rozpoznán. Je tedy reprezentací kognitivní funkce mozku, zatímco všechny předchozí vlny byly projevem senzorické funkce mozku. Amplituda vlny P3b je také závislá na pozornosti měřené osoby a stavu. Pokud je v místnosti, ve které probíhá měření, rušno, je vlna P3b těžko rozpoznatelná, podobně pokud je měřená osoba např. nachlazená či pod vlivem medikamentů a jiných látek. [Luck(2005)]

2.3.3 ERP experimenty

Mezi nejčastější ERP experimenty patří experiment založený na schématu anglicky zvaném *Oddball paradigm*. *Oddball paradigm* experiment je typ ERP

experimentu, při kterém jsou měřené osobě opakovaně pouštěny zvukové či vizuální stimuly (necílové, non-target stimuly) z předem určené sady stimulů. Mezi těmito stimuli se vyskytuje jeden speciální stimul (cílový, target stimul), na který má měřená osoba reagovat. Příkladem může být experiment, který byl proveden na Západočeské univerzitě v Plzni, kdy cílovým stimulem bylo písmeno "Q" (pravděpodobnost výskytu 15%) a necílovým stimulem bylo písmeno "O" (pravděpodobnost výskytu 85%). Tato písmena jsou po dobu několika minut opakovaně promítána na obrazovku počítače a měřená osoba má za úkol počítat výskyt písmene "Q". Cílem experimentu je vyvolat u měřené osoby komponentu P3. Výsledný signál byl zaznamenán pomocí zařízení BrainAmp a uložen na počítači. Zařízení BrainAmp ukládá pro každý stimul značku popisující typ stimulu a čas jeho vyslání. Tato značka spojuje daný stimul s následným časovým úsekem signálu (epocha). Bez těchto značek by nebylo možné signál dále vyhodnocovat. Při měření byla použita vzorkovací frekvence 1 kHz a rozlišení 0.1μV. Ze získaných epoch se poté vytvářejí průměry. Průměrováním epoch se sníží vliv náhodných artefaktů v signálu (např. mrknutí oka) a zvětší se projev komponenty P3. [Ciniburk(2011)]

Data z tohoto experimentu byla použita pro testování navržených modifikací v kapitole 8.

Pro lepší výsledky ERP experimentů je nutná dobrá soustředěnost měřené osoby. Při experimentech se využívá zvukotěsných komor, aby měřená osoba nebyla rušena okolím. Měřená osoba by také neměla být pod vlivem nemoci, medikamentů a jiných látek, jelikož se tím zpožďuje a snižuje její reakce na vyslaný stimul.

2.4 Artefakty

Kvůli způsobu měření dostáváme EEG signál obsahující nežádoucí části, vzniklé např. vlivem měřících přístrojů či samotné měřené osoby. Při zpracování signálu pak chceme tyto části ze signálu odstranit. Nechtěné části se nazývají artefakty a podle vzniku se dělí na fyziologické a technické.

2.4.1 Fyziologické artefakty

Svalová aktivita

Tyto artefakty jsou způsobené kontrakcí svalů, zejména pak svalů v obličejové část hlavy. V EEG signálu se vyskytují nejčastěji a mohou trvat různě dlouhou dobu. Nejčastěji jsou způsobené skousnutím či pohybem jazyka.

Pohyb očí a mrkání

Nejvlivnějším očním artefaktem je artefakt způsobený rozdílem potenciálů mezi rohovkou a sítnicí. Vzniká při pohybu očí a mrkání. Vzhledem k tomu, že člověk mrká několikrát za minutu a pohyb očí je téměř neustálý, jsou tyto artefakty velmi časté.

Srdeční aktivita a pulz

Artefakty vznikají jednak změnou potenciálů vlivem srdce, jednak vlivem vln způsobených pulzem. Tyto vlny ovlivňují kontakt mezi elektrodami a skalpem a v EEG signálu se projevují periodickými vlnami sinusového či trojúhelníkového tvaru. [Fisch()]

Kůže

Další artefakty vznikají vlivem pocení. Když se kůže na skalpu potí, mění se kontakt mezi skalpem a elektrodou a v EEG signálu se tak objevují pomalé vlny, obvykle delší než 2 sekundy.[Fisch()]

2.4.2 Technické artefakty

Elektrody

Pohyb pacienta či samovolný pohyb elektrod na hlavě může způsobit náhlé změny impedance elektrod a projeví se hroty v EEG signálu. Impedance se také může měnit pozvolně, vlivem vysychání vodivého gelu mezi elektrodami a povrchem skalpu. Tyto artefakty se týkají převážně jednotlivých elektrod.

Elektrická síť

Další artefakty jsou způsobené napájením měřících přístrojů z elektrické sítě a mají frekvenci 50 Hz (Evropa) nebo 60 Hz (Severní amerika).

3 Hilbert-Huangova transformace

EEG je signál obecně nelineární a nestacionární, proto je jeho zpracovávání nesnadným úkolem. Existují sice použitelné způsoby (Waveletová trasformace nebo Fourierova transformace), ty jsou ovšem uzpůsobené ke zpracování lineárních a stacionárních signálů. Tyto metody mají předem dané bázové funkce, např. u Fourierovy transformace funkce trigonometrické, na které se signál rozkládá. Nelze však předpokládat, že tyto funkce budou reálně reprezentovat fyzikální a biologické procesy, jejichž výsledkem je naměřený EEG signál. Proto je nutné použití metod s adaptivní bází.

Takovou metodou je Hilbert-Huangova transformace (HHT). HHT se skládá z Empirické modální dekompozice (EMD) a Hilbertovy transformace. Je to adaptivní metoda pro zpracovávání dat, navržená pro nelineární a nestacionární data. Klíčovou částí Hilbert-Huangovy transformace je EMD, která rozloží zpracovávaný signál na konečný a často i nízký počet komponent nazývaných Intrinsic mode functions (IMF). Použitím Hilbertovy transformace na takto získané IMF pak můžeme získat okamžitou frekvenci a amplitudu původního signálu, které jsou dobrou reprezentací reálných fyzikálních pochodů. Narozdíl od Fourierovy transformace není HHT teoreticky prokázána, spíše se jedná o empirický přístup, algoritmus, skládající se ze dvou kroků:

1. Pomocí EMD rozlož původní signál do Intrinsic mode funkcí a ulož je do seznamu *IMF*.
2. Pro každé IMF_i ze seznamu IMF získaném v předchozím kroku:
   (a) Získej okamžitou amplitudu a frekvenci aplikováním Hilbertovy transformace na IMF_i.

Výsledkem je energeticko-časově-frekvenční distribuce, nazývaná Hilbertovo spektrum.[N. E. Huang(2005)]

3.1 Intrinsic mode function

Intrinsic mode function je funkce, která splňuje následující dvě vlastnosti:

- V celém data setu se počet extrémů a počet přechodů přes nulu musí rovnat, nebo se lišit maximálně o jedna.
- V jakémkoliv bodě musí být průměrná hodnota obálky definované lokálními maximy a lokálními minimy rovna nule.

Jedno IMF tak narozdíl od harmonických funkcí s konstantní amplitudou a frekvencí představuje jednoduchou oscilující funkci s proměnlivou frekvencí a amplitudou. [N. E. Huang(2005)]

3.2 Empirická modální dekompozice

Cílem EMD je rozložit zpracovávaný signál na jednotlivé IMF, dokud ze signálu nezbyde jen monotónní funkce. Rozkládání signálu (anglicky *sifting*) se provádí pomocí následujícího algoritmu[Ciniburk(2011)]:

1. Inicializujeme zbytek signálu na původní signál $r_0(t) = x(t)$ a počítadlo IMF $i = 1$
2. Extrahujeme i-té IMF:
   (a) Inicializujeme $h_0 = r_{i-1}(t)$ a počítadlo kroků $k = 1$
   (b) Nalezneme všechna lokální maxima a minima v $h_{k-1}(t)$
   (c) Vytvoříme horní obálku spojením lokálních maxim kubickým splinem
   (d) Vytvoříme dolní obálku spojením lokálních minim kubickým splinem
   (e) Spočítáme průměr $m_{k-1}(t)$ zprůměrováním horní a dolní obálky
   (f) Spočítáme $h_k(t) = h_{k-1}(t) - m_{k-1}(t)$
   (g) Zkontrolujeme zastavovací podmínku (viz kapitola 3.2.1)
      i. Pokud je podmínka splněna, tak $IMF_i = h_k(t)$
      ii. Jinak $k = k + 1$ a pokračujeme krokem 2b.
3. Nový zbytek signálu je $r_i(t) = r_{i-1}(t) - IMF_i$
4. Zkontrolujeme zastavovací podmínku pro EMD

(a) Pokud má $r_i(t)$ alespoň dva extrémy, a tedy není monotonní, pak $i = i + 1$ a pokračujeme krokem 2.

(b) Jinak je rozklad dokončen a $r_i(t)$ je zbytkem rozkladu.

3.2.1 Zastavovací podmínka

V historii použity byly použity dvě zastavovací podmínky, první je dána Cauchyho testem konvergence, tedy

$$SD_k = \frac{\sum_{t=0}^{T} |h_{k-1}(t) - h_k(t)|^2}{\sum_{t=0}^{T} h_{k-1}^2(t)}, \tag{3.1}$$

která zastaví proces rozkladu, jakmile je hodnota SD_k menší než předem stanovená hodnota. Jedná se vlastně o normalizovaný rozdíl mezi dvěmi po sobě jdoucími fázemi rozkladu. Problémem však je, jak zvolit takovou hraniční hodnotu, která bude dostatečně malá. Proto Huang a spol. navrhli druhou zastavovací podmínku založenou na počtu extrémů a přechodů přes nulu. Je zvoleno číslo S a proces rozkladu je zastaven poté, co počet extrémů a přechodů přes nulu se rovná nebo liší maximálně o jedna v S po sobě jdoucích fázích rozkladu.[N. E. Huang(2005)]

3.2.2 Problémy EMD

První problém byl uveden již u zastavovacích podmínek a to jakým způsobem nastavit předem stanovené hranice. Dalším problémem je tvorba obálek pomocí lokálních extrémů. Musíme si uvědomit, že body, které leží na začátku před prvním extrémem a body ležící na konci signálu za posledním extrémem, budou odseknuty a nevyužity. Proto je potřeba dodefinovat některá lokální maxima a minima. Můžeme však jen odhadnout, kde se budou tyto extrémy nacházet a jak budou vypadat. Několik metod již bylo navrhnuto, jejich popis ale není obsahem této práce, proto je uvedu jen zmínkou, např. Mirror method nebo Slope-Based method. Dalším problémem je samotná detekce lokálních extrémů. Jednoduchým řešením by bylo testovat vždy 3 body signálu a pokud je amplituda prostředního bodu menší, resp. větší, jedná se o lokální minimum, resp. maximum. Avšak při experimentech se ukázalo, že ne všechny takové body jsou extrémy, některé vznikly jako vedlejší efekt předzpracování a filtrování signálu. Některé metody byly opět

navrženy, ovšem stejně jako u předchozího problému nebylo přesné řešení zatím nalezeno.

3.3 Hilbertova transformace

Okamžitá frekvence může být určena pomocí Hilbertovy transformace, podle které může být každá funkce *x(t)* převedena na funkci analytickou přidáním komplexní části *y(t)*:

$$y(t) = \frac{1}{p} P \int_{-\infty}^{\infty} \frac{x(t)}{t-\tau} d\tau \tag{3.2}$$

Analytická funkce vypadá následovně:

$$z(t) = x(t) + jy(t) = a(t)e^{j\theta(t)} \tag{3.3}$$

a

$$a(t) = (x^2 + y^2)^{1/2}; \theta(t) = tan^{-1}\frac{y}{x}. \tag{3.4}$$

a značí okamžitou amplitudu a θ je funkce fáze, okamžitá frekvence je tedy

$$\omega = -\frac{d\theta}{dt}. \tag{3.5}$$

Tímto způsobem lze ze získaných IMF určit okamžitou amplitudu a frekvenci signálu. [N. E. Huang(2005)]

4 Ensemble Empirical Mode Decomposition

Jedním z hlavních problémů EMD je tzv. *mode mixing*, jev při kterém EMD nedokáže správně rozložit signál na jednotlivé IMF a v důsledku toho jedna IMF obsahuje několik složek velice rozličných frekvencí. Takový jev nastává zejména v případech, kdy původní signál obsahuje nesouvislé a ojedinělé oscilace s vysokou frekvencí v signálu s jinak malou frekvencí. Výsledné IMF pak neodpovídají opravdovým fyzikálním pochodům. K vyřešení tohoto problému Huang a spol. navrhli tzv. *"intermittence test"*, který sice v jednoduchých případech dokáže *mode mixing* odstranit, ovšem v reálných signálech, které jsou velmi složité, není příliš efektivní. Proto byla navržena nová metoda datové analýzy za pomoci šumu (noise-assisted data analysis method - NADA) nazvaná *Ensemble Empirical Mode Decomposition* (EEMD). Při EEMD jsou jednotlivé IMF získány zprůměrováním výsledků několika EMD, při kterých byl k původnímu signálu přidán náhodný bílý šum konečné amplitudy.

4.1 Bílý šum

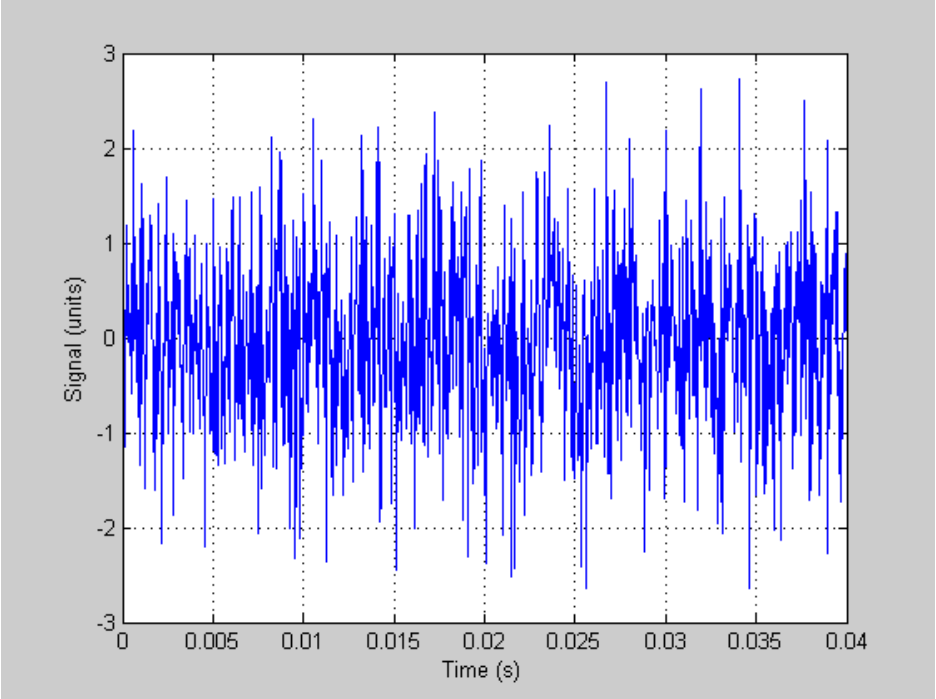
Pojmem bílý šum, analogie k bílému světlu, je označován náhodný signál, jehož výkonová spektrální hustota je konstantní. Výkonová spektrální hustota signálu je závislost jeho výkonu na frekvenci a její graf je v tomto případě plochý (obr. 4.2). To tedy znamená, že všechny frekvence jsou v signálu zastoupeny stejně silně a jeho výkon je tak v libovolné šířce pásma stejný. Např. mezi frekvencemi 100 Hz a 120 Hz je suma výkonu stejná jako mezi frekvencemi 400 Hz a 420 Hz. Ukázka bílého šumu je na obrázku 4.1.

4.2 Algoritmus

Princip EEMD je založen na třech bodech:

1. Kolekce bílých šumů se při průměrování vyruší a jediné co zůstane, je tak původní signál.

Obrázek 4.1: Ukázka bílého šumu. [Sek()]



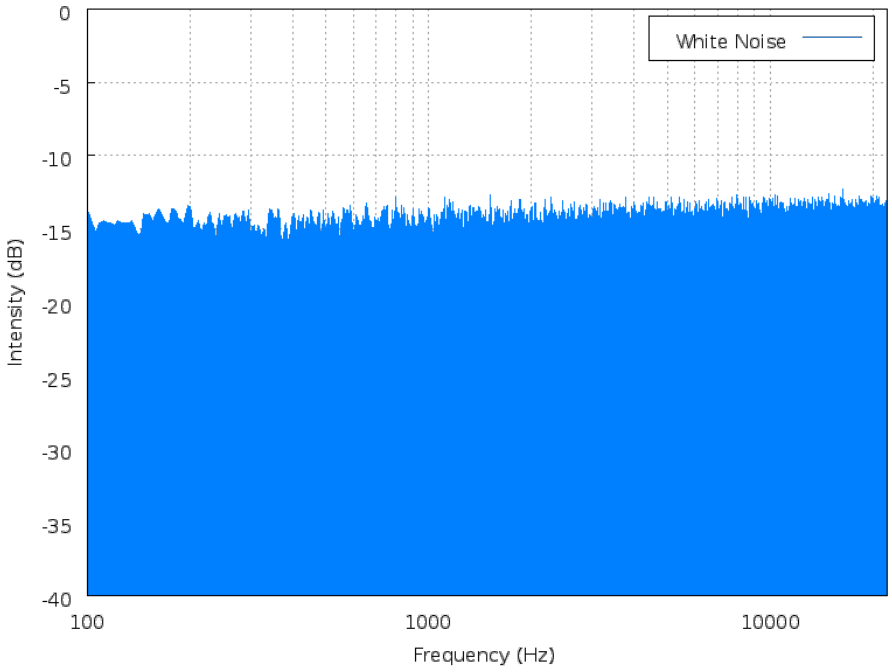
2. Přidaný šum s konečnou amplitudou vyústí ve správný rozklad signálu do jednotlivých IMF.
3. Správná a fyzikálně smysluplná odpověd EMD není ta bez přidaného šumu, nýbrž průměr velkého počtu opakování, pokaždé s jiným bílým šumem.

Algoritmus vypadá takto:

1. Přidáme náhodný bílý šum do původního signálu
2. Provedeme rozklad podle EMD do jednotlivých IMF
3. Opakujeme kroky 1 a 2, pokaždé ale s jiným přidaným šumem.
4. Zprůměrujeme jednotlivé IMF ze všech opakování, tento průměr je výsledkem.

EEMD automaticky odstraní *"mode mixing"* a je velikým vylepšením původní EMD.[Z. Wu(2009)]

Obrázek 4.2: Spektrum bílého šumu.[Kinlay()]



5 Knihovna HHT

Knihovna HHT vznikla jako součást diplomové práce J. Ciniburka v roce 2011 na Západočeské univerzitě v Plzni. Od vytvoření byla široce doplněna a je vhodná pro testování HHT a navržených modifikací. Vytvořena byla v programovacím jazyce Java a skládá se ze tří částí [Ciniburk(2011)]:

- jádro HHT
- logování a vizualizace
- testování

V následující části stručně popíši jádro HHT, jelikož se většina mé práce bude odehrávat v této části. O všech částech knihovny se můžete detailně dočíst v [Ciniburk(2011)].

5.1 Jádro knihovny HHT

Jádro obsahuje potřebné třídy pro provedení všech částí Hilbert-Huangovy transformace včetně detekce extrémů, odhadu krajních extrémů a zastavovacích podmínek. Knihovna je navržena tak, aby bylo možné tyto třídy jednoduše nahradit. Struktura je zobrazena na diagramu tříd (viz. obr. 5.1).

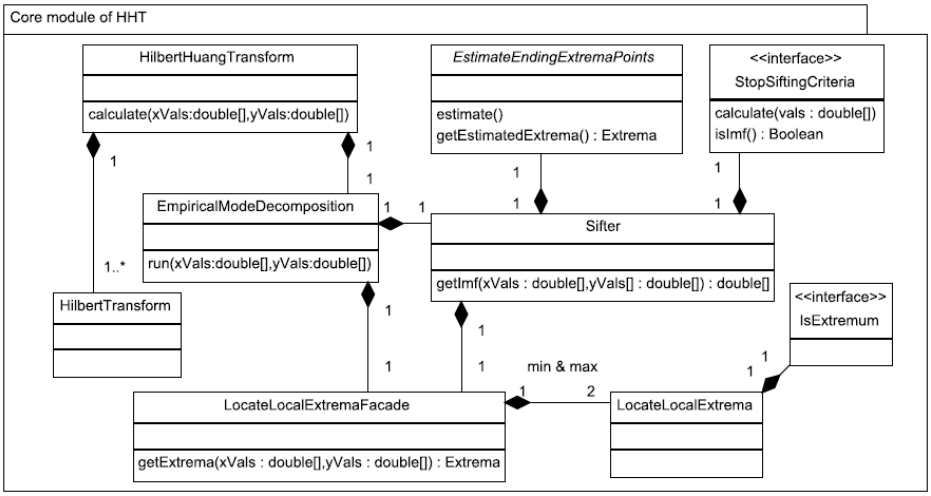
Třída EmpiricalModeDecomposition

Tato třída provede Empirickou modální dekompozici na vstupních datech a výsledné IMF uloží do typu *Vector<double[]>* jako instanční atribut. Potřebuje instanci tříd LocateLocalExtremaFacade pro určování lokálních extrémů a Sifter pro proces *siftingu*. [Ciniburk(2011)]

Třída Sifter

Třída Sifter vykonává na vstupních datech proces *siftingu*, jehož výsledkem je jedna IMF vrácena jako pole typu double. Pro tento proces využívá služeb tříd EstimateEndingExtremaPoints pro určení chybějících krajních lokálních extrémů, StopSiftingCriteria pro kontrolu zastavovací podmínky

Obrázek 5.1: Diagram tříd jádra knihovny HHT.[Ciniburk(2011)]



siftingu a LocateExtremaFacade pro určení extrémů. Vnitřně pak využívá služeb třídy Enveloper pro tvorbu horních a dolních obálek. [Ciniburk(2011)]

Třída LocateLocalExtremaFacade

Třída hledá lokální extrémy ve vstupních datech. Vstupní data jsou postupně procházena a každý bod je kontrolován třídou implementující rozhraní IsExtremum, zda je lokálním maximem či minimem. [Ciniburk(2011)]

Rozhraní StopSiftingCriteriaInterface

Toto rozhraní obsahuje metody k zastavení procesu *siftingu*. V knihovně jsou implementovány obě podmínky zmíněné v kapitole 3.2.1 a to třídami StandartDeviation a CauchyConvergence. [Ciniburk(2011)]

Třída HilbertTransform

Tato třída provede Hilbertovu transformaci na jednotlivých IMF. Nejprve určí analytický signál Rychlou Fourierovou transformací a poté vypočítá okamžité amplitudy, fáze a frekvence metodou *calculate(...)*. Tyto hodnoty jsou uloženy jako atributy této třídy.

Třída HilbertHuangTransform

Třída HilbertHuangTransform použitím výše uvedených tříd aplikuje HHT

na vstupní data. Nejprve algoritmem EMD získá potřebné IMF a poté na každou z nich aplikuje HT (třída HilbertTransform). Výsledky jsou uloženy v seznamu instancí třídy HilbertTransform, jedna pro každé IMF. [Ciniburk(2011)]

6 Optimalizace knihovny

Hilbert-Huangova transformace může být využita na mnoha místech, kde je dlouhé čekání na výsledek nežádoucí, ať už se jedná o vyhodnocení EEG vyšetření v nemocnicích nebo zpracování finančních dat na burze. Tato data jsou často velmi objemná a obsahují i několik milionů hodnot. Z tohoto důvodu je nutné výpočet HHT urychlit.

6.1 Paralelizace

Při vyhodnocování EEG meření zpracováváme data ze všech použitých elektrod nezávisle na sobě. Např. pokud využijeme 10 elektrod, musíme provést HHT pro data získaná ze všech těchto elektrod. Použijeme-li navíc EEMD, výpočet několikrát opakujeme, pokaždé s jiným šumem. Pro 10 různých bílých šumů tak musíme provést EEMD celkem 100-krát. Proto je velmi výhodné výpočet paralelizovat. To je uskutečněno ve dvou úrovních:

- paralelizace HHT pro jednotlivé elektrody
- paralelizace EEMD pro různé bílé šumy

Při řešení paralelizace se dnes používají zejména navrhové vzory Thread-Per-Request a ThreadPool.

6.1.1 Návrhový vzor ThreadPerRequest

Při tomto řešení je pro každý úkol vytvořeno nové vlákno. Implementace je tak celkem jednoduchá, avšak ztrácíme kontrolu nad počtem vláken a může tak dojít k vyčerpání zdrojů OS. Je také nevhodný pro časově nenáročné úkoly, jelikož čas potřebný k vytvoření nového vlákna je pak v poměru k času zpracování úkolu relativně veliký. Tento vzor je tak použitelný spíše pro servery při řešení spojení server - klient.

6.1.2 Návrhový vzor ThreadPool

ThreadPool namísto vytváření jednoho vlákna pro každou úlohu vytvoří předem daný počet vláken, což šetří čas a zdroje. Umožní nám to určit počet vláken a tím přizpůsobit návrh danému problému a dosáhnout tak nejlepšího výkonu. Vytvořená vlákna čekají ve frontě vláken na přidělení úkolu. Po skončení práce se opět vracejí do fronty, kde čekají na další úkol.

V závislosti na výše uvedených důvodech jsem se rozhodl při paralelizaci výpočtu HHT použít vzor ThreadPool, kvůli jeho přizpůsobitelnosti, ovladatelnosti a lepší práci se zdroji.

7 Implementace

Knihovna HHT je naprogramována v Javě 1.7, proto jsem se této verze při implementaci držel také. Při návrhu řešení jsem se snažil postupovat tak, aby mnou vytvořené modifikace nevyžadovali úpravu stávajícího kódu. Mým úkolem bylo implementovat EEMD a poté jako součást optimalizace knihovny paralelizovat EMD i EEMD pro souběžné zpracování více dat najednou.

7.1 Implementace EEMD

Ensemble empirical mode decompostion je v zásadě rozšíření klasické Empirical mode decomposition o přidání bílého šumu do vstupních dat. Nejprve bylo tedy zapotřebí implementovat generátor bílého šumu a poté ho spojit s EMD. V průběhu implementování jsem používal verzovací nástroj Git zajištěný webovou službou BitBucket.

7.1.1 Třída WhiteNoiseGenerator

Třída WhiteNoiseGenerator slouží ke generování bílého šumu. Obsahuje jediný instanční atribut *double amplitude* omezující generované hodnoty. Ten je možné nastavit v konstruktoru nebo pomocí příslušného *setru*. Pokud není jeho hodnota nastavena, je použita defaultní hodnota uložená ve statickém atributu *double DEFAULT_AMPLITUDE*. K samotnému generování šumu slouží metoda *double[] generate(int size)*. Její parametr *size* určuje velikost vraceného pole a počet generovaných hodnot. Pro generování náhodných hodnot je použita třída Random z balíku *java.util* a její metoda *nextDouble()*. Ta sice generuje pouze pseudonáhodná čísla, avšak ta jsou pro běh EEMD dostatečně náhodná. Třída také obsahuje metodu *generateGaussian(int size)*, která generuje náhodná čísla s normálním rozdělením pomocí metody *nextGaussian()* třídy Random.

7.1.2 Třída EnsembleEmpiricalModeDecomposition

Pro implementaci EEMD jsem použil již v knihovně implementovanou verzi EMD třídou EmpiricalModeDecomposition. Pomocí ní je možné spustit EMD metodou *run(...)*. Třída EnsembleEmpiricalModeDecomposition dědí od této třídy a překývá její metodu *run(...)*. Ve zbytku knihovny tak není nutná jakákoliv úprava. Při vytváření instance třídy HilbertHuangTransform pouze dodáme do konstruktoru místo instance třídy EmpiricalModeDecomposition instanci třídy EnsembleEmpiricalModeDecomposition.

Pro vytvoření instance třídy EnsembleEmpiricalModeDecomposition jsou, stejně jako pro vytvoření jejího předka, potřebné instance tříd Sifter a LocateLocalExtremaFacade a navíc instance třídy WhiteNoiseGenerator. Počet opakování s bílým šumem je možné nastavit v konstruktoru nebo příslušným *setrem*. Instanci je také možné vytvořit pomocí XmlBeanFactory frameworku Spring metodou *getEmd(...)*. Metoda se schválně jmenuje *getEmd(...)* namísto getEemd(...), aby překrývala metodu *getEmd()* třídy EmpiricalModeDecomposition. Jejím parametrem je cesta ke Xml konfiguračnímu souboru (Příloha: Listing 1).

EEMD dekompozice se spustí zavoláním metody *run(double[] xVals, double[] yVals)*, která ukládá výsledné zprůměrované IMF do instančního atributu *imfs*. Pro každý běh je v metodě nejprve do vstupních dat *yVals* přidán bílý šum a poté je s těmito daty spuštěna předkova metoda *run(...)*. Takto získané IMF jsou uloženy do seznamu s nezprůměrovanými IMF *nonAveragedImfs*. Po dokončení všech dílčích EMD s náhodnými bílými šumy jsou tyto IMF zprůměrovány metodou *averageImfs(...)*. K dispozici jsou také následující přetížené verze metody *run(...)*:
run(double[] xVals, double[] yVals, int noOfThreads)
Slouží k běhu paralelní verze dekompozice (více viz. Kapitola 7.2). Nejprve je vytvořena instance třídy ThreadPool s počtem vláken daným parametrem *noOfThreads*. Poté jsou vytvořeny instance třídy EMDTask a vloženy do fronty úkolů. Po dokončení všech úkolů jsou IMF zprůměrovány a uloženy do instančního atributu *imfs*.

run(double[][] xVals, double[][] yVals)
Spustí jednovláknovou verzi dekompozice pro více dat. Nad všemi daty je provedena dekompozice pomocí metody *run(double[] xVals, double[] yVals)*. Nezprůměrované IMF jsou ukládány do instančního atributu *nonAveragedImfsVector*. Po dokončení dekompozice všech dat jsou IMF zprůměrovány

a uloženy do instačního atributu *imfsVector*.

run(double[][][] xVals, double[][][] yVals, int noOfThreads)
Spustí paralelní verzi dekompozice pro více dat. Nejprve je vytvořena instance třídy ThreadPool s počtem vláken daným parametrem *noOfThreads*. Postupně jsou vytvořeny instance třídy EMDTask pro všechny data a vloženy do fronty úkolů. Po dokončení všech úkolů jsou IMF zprůměrovány metodou *averageImfs(...)* a uloženy do instančního atributu *imfsVector*.

7.2 Paralelizace knihovny

Jak již bylo uvedeno v kapitole 6.1, rozhodl jsem se pro paralelizaci použít návrhvový vzor Threadpool. Celkem bylo nutné provést tři paralelizace:

- paralelní běh EEMD s různými bílými šumy
- paralelní běh EEMD pro více dat
- paralelní běh EMD pro více dat

Ukázalo se, že se jedná o totožné problémy a to o souběžný běh několika EMD s různými daty. Jelikož však knihovna pro EMD často používá instanční atributy, není možné použít jednu a tu samou instanci EmpiricalModeDecomposition pro všechny běhy. Není ale nutné vytvářet novou instanci pro každá data. Lepším řešením je každému vláknu přiřadit jednu instanci třídy EmpiricalModeDecomposition, pomocí které bude spouštěna EMD. Java obsahuje vlastní Threadpool v balíku *util.concurrent*, avšak řešení pomocí tříd tohoto balíku se ukázalo jako zbytečně složité. Rozhodl jsem se proto implementovat vlastní Threadpool. Vytvořil jsem celkem tři třídy: Threadpool, představující samotný Threadpool, WorkerThread, představující pracující vlákna a EmdTask pro předávání úkolů.

7.2.1 EmdTask

Instance třídy EmdTask představují úkoly pro pracující vlákna WorkerThread. Instanční atributy *double[] xVals* a *double[] yVals* slouží k uložení dat, nad

kterými má být provedena EMD. Atribut *Vector<double[]> resultStorage* slouží k uložení výsledných IMF.

7.2.2 WorkerThread

Třída WorkerThread dědí od třídy Thread. Konstruktor obsahuje dva parametry, instanci třídy ThreadPool a EmpiricalModeDecomposition. Po spuštění vlákna ve své metodě *run()* opakovaně přistupuje ke frontě úkolů threadpoolu a vybírá instance EmdTask metodou *take()* třídy ArrayBlockingQueue. Po získání úkolu (instance třídy EMDTask) provede empirickou modální dekompozici pomocí své instance třídy EmpiricalModeDecomposition a výsledek uloží do atributu *resultStorage* instance EmdTask. Po uložení IMF je potřeba vymazat IMF z instance EmpiricalModeDecomposition, jelikož při provádění další dekompozice touto instancí by došlo k míchání nových IMF se starými. To se provede zavoláním instanční metody *clearImfs()*.

7.2.3 ThreadPool

Jedná se o klasický Threadpool. Konstruktor obsahuje 3 parametry: instanci EmpiricalModeDecomposition, počet vláken a maximální počet úkolů. V konstruktoru jsou vytvořeny instance WorkerThread a uloženy do *ArrayList<WorkerThread>*. Při vytváření těchto instancí dochází ke klonování instance EmpiricalModeDecomposition. Instance EmdTask jsou ukládány do *ArrayBlockingQueue<EmdTask>*. Z této fronty si instance WorkerThread berou úkoly pomocí její instanční metody *take()*. Pokud je fronta prázdná, jsou vlákna zablokována, dokud není do fronty vložen nový úkol.

7.2.4 Klonování

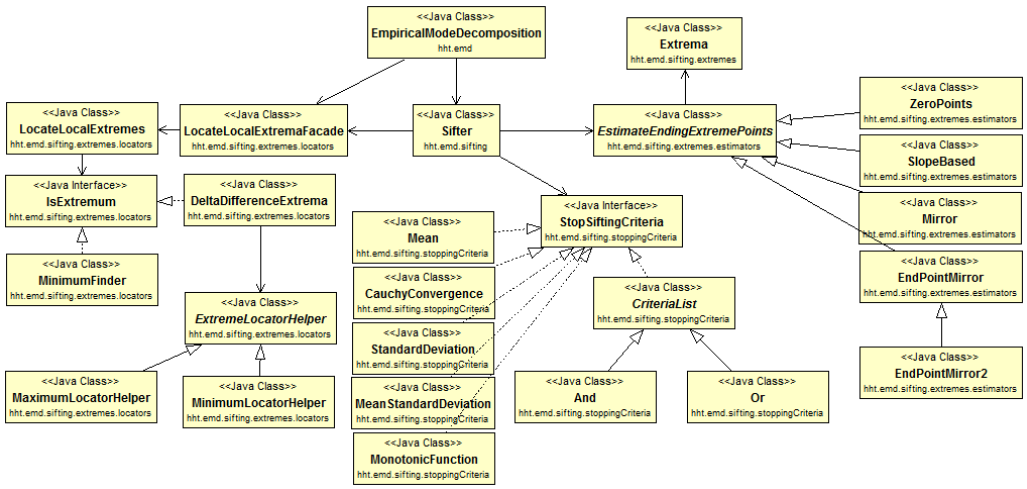
Protože každé vlákno používá vlastní instanci EmpiricalModeDecomposition, je nutné tyto instance klonovat. Každý objekt v Javě dědí metodu *clone()* ze třídy Object. Tato metoda vytvoří novou instanci dané třídy a poté naplní její atributy hodnotami korespondujících atributů kopírované instance. Toto naplnění je provedeno přiřazením. To však znamená, že pouze primitivní datové typy jsou opravdu zkopírovány, zatímco u referenčních datových typů dojde pouze ke zkopírování reference na datový objekt, ne samotných dat.

Tomuto kopírování se říká mělké. Pro naše účely potřebujeme zkopírovat i hodnoty referenčních typů, tedy kopírování hluboké. To se v Javě provádí implementací rozhraní Cloneable a překrytím metody *clone()* třídy Object. Správné hluboké klonování by mělo splňovat tři podmínky:

1. Výraz `x.clone() != x` je pravdivý, tedy metoda *clone()* nevrátí pouze referenci klonovaného objektu.
2. Výraz `x.clone().getClass() == x.getClass()` bude pravdivý, tedy metoda *clone()* vrátí instanci klonované třídy.
3. Výraz *x.clone().equals(x)* bude pravdivý.

Tyto podmínky jsou však spíše doporučením a není nutné jejich absolutní splnění. Prvním příkazem v metodě *clone()* by mělo být zavolání metody *clone()* předka:
`super.clone()`
Za ním následují příkazy zajišťující správné zkopírování všech referenčních instančních atributů dané třídy. Podporu klonování bylo nutné zajistit ve třídách balíku *hht.emd*. UML diagram všech tříd, které podporují klonování je na obrázku 7.1.

Obrázek 7.1: Diagram tříd, podporujících hluboké klonování. Kvůli lepší přehlednosti obsahuje diagram jen důležitější vazby.



7.3 Unit testy

Pro účely testování EEMD jsem implementoval sadu unit testů pomocí frameworku JUnit. Testy se nacházejí ve třídě EnsembleEmpiricalModeDecompositionTest v balíku hht.eemd.testing. Při návrhu unit testů jsem byl omezen následujícími vlastnostmi EEMD:

1. Kvůli náhodnému generování bílého šumu nejsou výsledky dvou EEMD nad stejnými daty totožné.
2. Kvůli náhodnému generování bílého šumu nelze předem určit přesný výsledek EEMD.

Programové testování správnosti výsledků EEMD tak není snadným úkolem. Možným řešením by bylo zvolit hodnotu ε a porovnání provést následnovně:

$$\text{Hodnoty } a \text{ a } b \text{ se rovnají, pokud platí } |a - b| < \epsilon.$$

S tím však vzniká problém volby hodnoty ε. Z těchto důvodů jsem se rozhodl při testování použít bílý šum s nulovou amplitudou. IMF získané metodou EEMD se díky tomu budou rovnat IMF získaným metodou EMD a bude možné je proti těmto hodnotám porovnat. Pro porovnání výsledků jsem vytvořil vlastní metodou *compareImfs(...)*, jelikož metoda *assertEquals(...)* frameworku JUnit neporovnává správně typ *Vector<double[]>*, ve kterém jsou uloženy IMF. Vytvořil jsem celkem 4 testy:
testRunSingleThreadZeroNoise()
Tento test slouží k ověření neparalelní verze metody *run(...)* třídy EnsembleEmpiricalModeDecomposition.

testRunParallelZeroNoise()
Ověřuje paralelní verzi metody *run(...)* třídy EnsembleEmpiricalModeDecomposition.

testRunMultipleDataSingleThreadZeroNoise()
Ověřuje neparalelní verzi metody *run(...)* pro více dat najednou.

testRunMultipleDataSingleThreadZeroNoise()
Ověřuje paralelní verzi metody *run(...)* pro více dat najednou.

7.4 Použití EEMD v aplikaci

Použití implementované EEMD v jiné aplikaci je jednoduché a skládá se z následujících kroků:

1. Vytvoření instance *eemd* třídy EnsembleEmpiricalModeDecomposition konstruktorem nebo metodou *getEmd(string cfgXml)*.
2. Zavolání jedné z přetížených verzí metody *run(...)* nad instancí *eemd*, získané v kroku 1.
3. Výsledné IMF se nacházejí v instančním atributu *imfs*, případně *imfsVector*, pokud byla zavolána verze metody *run(...)* pro zpracování více dat.

7.5 Spuštění HHT s EEMD

EEMD lze nejsnadněji spustit pomocí třídy hht.HhtSimpleRunner. Třída obsahuje metodu *main*, která zpracovává parametry z příkazové řádky. Parametry příkazové řádky jsou následující:

1. cesta k adresáři, do kterého budou uloženy výsledky
2. cesta ke XML konfiguračnímu souboru (Příloha: Listing 1), který slouží k nakonfigurování EEMD
3. cesta k souboru *.txt*, který obsahuje testovací data
4. vzorkovací frekvence (v tomto případě 1000)
5. počet hodnot v datovém souboru (testovací data obsahují 1024 hodnot)

Příklad parametrů je vidět zde:

```
slozka/vysledky cesta/ke/konfiguraci/konfig.xml
cesta/kdatum/data.txt 1000 1024.
```

Pokud chcete použít HHT s EEMD namísto EMD, stačí v metodě *runHHT(...)* změnit instanci EmpiricalModeDecomposition na instanci EnsembleEmpiricalModeDecomposition.

Po dokončení HHT bude složka určená prvním paramentrem příkazové řádky obsahovat textové soubory se záskanými IMF, okamžitými amplitudami a okamžitými frekvencemi. Kromě toho také obsahuje spustitelný batch soubor *createImfsPdf.bat*, který výsledné IMF vykreslí do grafu a uloží do PDF (Příloha: obrázek 2).

8 Testování

Navržené modifikace byly otestovány na reálných EEG datech. Účinnost metody EEMD jsem ověřil na detekci komponenty P3 v EEG datech. Výslednou účinnost jsem porovnal s účinností obyčejné EMD. U paralelizace jsem nejprve ověřil, že paralelní běh dává stejné výsledky jako běh na jednom vláknu a poté jsem změřil časy běhů EEMD.

8.1 Testovací data

Testovací data byla získána při laboratorním experimentu popsaném v sekci 2.3.3. Experiment byl proveden s 20 různými osobami. Z měření byly zaznamenány 1s dlouhé epochy při vzorkovací frekvenci 1kHz. Jedna epocha tak obsahuje 1000 hodnot. Epochy byly průměrovány a testovací data obsahují průměry ze 2, 5, 10, 20 a 30 epoch. Na datech byla dále provedena baseline korekce. K epochám jsou známé vyslané stimuly (*target* nebo *non-target*). Epochy, které představují reakci měřené osoby na *target* stimul, obsahují hledané komponenty P3, zatímco epochy představující reakci na *non-target* stimul tyto komponenty neobsahují.

8.2 Testování EEMD

Pro nastavení EEMD jsem použil konfiguraci, při které bylo v [Ciniburk(2011)] dosaženo nejlepších výsledků. Konfigurační soubor je vidět v příloze (Příloha: Listing 1).

8.2.1 Porovnání získaných IMF

Pro účely porovnání získaných IMF jsem na stejných datech provedl dekompozici metodou EMD a metodou EEMD. Průměrná amplituda původních dat byla 6,7μV. EEMD jsem testoval s amplitudami 0,05μV, 0,1μV, 0,5μV, 1,0μV, 2,0μV a 4,0μV. Amplitudy vyšší než 0,5μV byli příliš vysoké a původní signál znatelně zkreslily. Nejlepší výsledky poskytly amplitudy 0,1μV

a 0,5 μV, což se potvrdilo i při testování úspěšnosti klasifikace komponenty P3 (sekce 8.2.2). I přes takto nízkou amplitudu bílého šumu oproti průměrné amplitudě původního signálu je ve výsledcích dekompozice značný rozdíl. Metoda EMD rozložila signál do 4 IMF (Příloha: obrázek 1). Metoda EEMD s amplitudou 0,1μV rozložila signál do 5 IMF (Příloha: obrázek 2), zatímco s amplitudou 0,5μV do 6 IMF (Příloha: obrázek 3). První čtyři IMF získané metodou EMD a metodou EEMD s amplitudou 0,1μV jsou téměř totožné, avšak EEMD v tomto případě našla ještě páté, metodou EMD neobjevené, IMF. EEMD s amplitudou 0,5μV potřebovala jednu počáteční iteraci *siftingu* navíc, kvůli značnému vlivu bílého šumu, který se v prvním IMF projevil "zubovitostí". Dalších pět IMF je opět téměř totožných s IMF u EMD a EEMD s amplitudou 0,1μV.

Nejedná se o ojedinělí případ. Podobné výsledky bylo možné pozorovat u více testovaných dat. Fakt, že EEMD našla poslední IMF s velice nízkou frekvencí, metodou EMD původně neobjevené, naznačuje, že EEMD opravdu řeší *mode mixing* problém.

8.2.2 Úspěšnost klasifikace vlny P3

Komponentu P3 jsem v IMF získaných pomocí EEMD hledal pomocí třídy testing.classificators.FreqAmlThreshold knihovny HHT. Třída používá k detekci vlny P3 následující algoritmus [Ciniburk(2011)]:

1. i = 0
2. Vezmi i-té IMF
3. Spočítej průměrnou frekvenci v úseku mezi 150ms a 650ms.
4. Pokud průměrná frekvence spadá mezi $< 0,2Hz, 3Hz >$, pokračuj krokem 4a, jinak přejdi na krok 5.
   (a) Spočítej průměrnou amplitudu v úseku mezi 150ms a 650ms.
   (b) Pokud je průměrná amplituda větší než daná hraniční hodnota, pak signál obsahuje komponentu P3.
5. Pokud existuje další IMF, inkrementuj i a pokračuj krokem 2, jinak signál neobsahuje komponentu P3.

Hraniční hodnotu amplitudy jsem nastavil na 3,0μV. Klasifikátor byl nastaven s ohledem na vlastnosti komponenty P3. Úspěšnost jsem testoval na průměrech ze 2, 5, 10, 20 i 30 epoch a za použití bílého šumu s různými amplitudami a různým počtem běhů. Stejný postup jsem provedl při použití obyčejné EMD. Výsledky jsou vidět v tabulce na obrázku 8.1. Nejlepší výsledky jsou vyznačeny zelenou barvou. Nejlepších výsledků bylo dosaženo při velmi malých hodnotách amplitudy (0,05μV a 0,1μV), průměrná amplituda signálů se pohybovala kolem 6,7μV. U těchto dvou amplitud bylo dosaženo lepších výsledků pro všechny průměry epoch. Pro průměry ze 2 a 5 epoch bylo dosaženo lepších výsledků u všech amplitud. To je výhodné, jelikož ne vždy je možné měření opakovat a výsledky průměrovat.

Obrázek 8.1: Úspěšnost detekce P3 komponenty v EEG signálu.

Počet běhů	Úspěšnost klasifikace [%]						
10	Amplituda bílého šumu						EMD
Průměrovaných epoch	0,05	0,1	0,5	1	2	4	
2	59,5238	57,1429	57,1429	57,1429	64,2857	57,1429	52,3810
5	64,2857	61,9048	69,0476	61,9048	64,2857	66,6667	59,5238
10	76,1905	73,8095	71,4286	71,4286	73,8095	73,8095	73,8095
20	80,9524	80,9524	78,5714	73,8095	76,1905	73,8095	76,1905
30	90,0000	82,5000	77,5000	77,5000	82,5000	75,0000	90,0000
Průměr [%]	74,1905	71,2619	70,7381	68,3572	72,2143	69,2857	70,3810
Počet běhů	Úspěšnost klasifikace [%]						
25	Amplituda bílého šumu						EMD
Průměrovaných epoch	0,05	0,1	0,5	1	2	4	
2	61,9048	54,7169	52,3810	54,7619	61,9048	54,7619	52,3810
5	61,9048	64,2857	66,6667	66,6667	69,0476	71,4286	59,5238
10	76,1905	73,8095	71,4286	71,4286	73,8095	73,8095	73,8095
20	78,5714	80,9524	76,1905	73,8095	78,5714	71,4286	76,1905
30	90,0000	82,5000	82,5000	85,0000	77,5000	75,0000	90,0000
Průměr [%]	73,7143	71,2529	69,8334	70,3333	72,1667	69,2857	70,3810
Počet běhů	Úspěšnost klasifikace [%]						
50	Amplituda bílého šumu						EMD
Průměrovaných epoch	0,05	0,1	0,5	1	2	4	
2	59,5238	57,1429	57,1429	52,3810	54,7619	57,1429	52,3810
5	64,2857	64,2857	66,6667	54,7619	66,6667	66,6667	59,5238
10	76,1905	76,1905	76,1905	71,4286	73,8095	71,4286	73,8095
20	80,9524	80,9524	78,5714	78,5714	77,5000	73,8095	76,1905
30	92,5000	82,5000	82,5000	80,0000	80,9524	77,5000	90,0000
Průměr [%]	74,6905	72,2143	72,2143	67,4286	70,7381	69,3095	70,3810

8.2.3 Spuštění testování

Pro účely testování je nejprve nutné provést HHT na testovacích datech. Nejjednodušším způsobem je využít třídu HhtDataRunner z balíku *testing*. V metodě *main* se pouze doplní cesta ke XML konfiguračnímu souboru (Příloha: Listing 2). Tím je možné provést HHT na všech testovacích datech najednou. [Ciniburk(2011)]

Po získání výsledků HHT se dá spustit klasifikace ERP komponent třídou RunClassificationAll z balíku *hht.classificators*. V její metodě *main* je třeba opět určit cestu ke konfiguračnímu souboru (Příloha: Listing 3). Ve složce s výsledky klasifikace lze poté najít HTML soubory obsahující informace o úspěšnosti klasfikace. Ukázka takového souboru je na obrázku 8.2.

Obrázek 8.2: HTML soubor s výsledky klasifikace.

Success:87,5000%

success[%]	success	ERPs correctly detected	no ERPs correctly detected	classificator	configuration	data name
87,5000%	35/40	16/20	19/20	FreqAmp15	CauchybestEEMD	30_Cz_baseline_

8.3 Testování paralelizace

Při testování jsem nejprve ověřil, že paralelizovaná i neparalelizovaná verze metody EEMD dává stejné výsledky pomocí unit testů popsaných v sekci 7.3. Poté jsem měřil celkové časy běhu. Testování jsem prováděl na počítači s procesorem Intel Pentium B980 (dvě jádra o frekvenci 2,4Ghz) se 4GB operační paměti. EEMD jsem spustil na datech obsahujících 4096 hodnot a každý test jsem opakoval dvacetkrát. Jednotlivé časy jsem zprůměroval. V tabulce na obrázku 8.3 jsou vidět dosažené časy.

Paralelizovaná verze byla podle očekávání rychlejší. Nejlepších výsledků bylo dosaženo při běhu pomocí dvou vláken. To je dáno tím, že procesor počítače, na kterém test běžel má dvě jádra. Při použití více než dvou vláken tak dochází k pouhému pseudoparalelismu. Pokud by byl test spuštěn na procesoru s více jádry, došlo by k většímu urychlení. Jelikož se při běhu EEMD nečeká na I/O akce, jsou všechna vlákna neustále vytížena. Ideální počet vláken je tedy počet jader procesoru, zároveň by však neměl přesáhnout počet opakování s různými bílými šumy, protože vlákna převyšující počet opakování by neobdržela žádný úkol ke zpracování.

Obrázek 8.3: Naměřené časy běhu (v milisekundách) EEMD na datech obsahujících 4096 hodnot.

Počet opakování s šumy	Výsledné časy [ms]								
	Počet vláken								
	1	2	3	4	5	8	10	20	40
10	180,6	141,0	141,6	160,8	157,1	144,5	148,0	142,9	155,4
20	326,4	243,5	252,7	259,7	236,0	229,8	251,6	254,1	236,5
40	635,8	436,4	454,5	458,7	465,6	466,3	458,9	476,1	446,5

9 Závěr

V první části práce jsem shrnul EEG/ERP experimenty a Hilbert-Huangovu transformaci. V druhé části jsem do knihovny HHT doimplementoval metodu EEMD a otestoval jsem ji na reálných EEG datech. Potvrdilo se, že metoda je vylepšením původní EMD a dokáže rozložit signál do správných IMF i v případech, kdy EMD nikoliv. Dále jsem ověřil účinnost této metody při detekci ERP komponent, konktrétně vlny P3. Metoda EEMD byla pro nízké amplitudy bílého šumu až o 4% úspěšnější při detekci vlny P3 než-li metoda EMD. Pro data získaná průměrováním menšího počtu epoch byla ještě úspěšnější, což je výhodné v případech, kdy průměrování není možné.

Dále jsem paralelizoval metodu EEMD a metodu EMD pro souběžné zpracování více dat. K paralelizaci jsem použil návrhový vzor Threadpool, díky čemuž je možné určit počet vláken pro běh obou metod. Bohužel bylo nutné využít při paralelizaci klonování, čímž vzniká povinnost, zajistit podporu klonování i pro některé třídy, které budou do knihovny přídány v budoucnu. Při testování však dosahovala paralelizovaná verze lepších časů než při běhu s jedním vláknem.

Úkoly vytyčené na začátku práce jsem splnil. V budoucnu by bylo dobré provést navazující testování metody EEMD na dalších datech, případně při jiných úkolech, než je detekce komponenty P3, aby se plně ukázali její výhody a nevýhody.

Přehled zkratek

EEG - elektroencefalogram
EMD - Empirical mode decomposition (Empirická modální dekompozice)
EEMD - Ensemble empirical mode decomposition
HHT - Hilbert-Huang transform (Hilbert-Huangova transformace)
IMF - intrinsic mode function
ERP - Event-related potential (Evokovaný potenciál)

Přílohy

Listing 1: XML konfigurační soubor pro EEMD.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
                http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd">

        <bean  id="eemd" class="hht.eemd.EnsembleEmpiricalModeDecomposition" >
                <constructor-arg  ref="sifter"/>
                <constructor-arg ref="extremesLocator"/>
                <constructor-arg  ref="whiteNoiseGenerator"/>
                <constructor-arg index="3" type="int" value="10"/>
                <constructor-arg index="4" type="int" value="1"/>
        </bean>

        <bean id="whiteNoiseGenerator" class="signalGenerators.WhiteNoiseGenerator" >
                <property name="amplitude" value="0.1"/>
        </bean>

        <bean id="extremesLocator" class="hht.emd.sifting.extremes.locators.
             LocateLocalExtremaFacade" >
                <constructor-arg ref="maxLocator" />
                <constructor-arg ref="minLocator" />
        </bean>

        <bean id="minLocator" class="hht.emd.sifting.extremes.locators.
             DeltaDifferenceExtrema">
                <constructor-arg ref="minHelper"/>
        </bean>

        <bean id="maxLocator" class="hht.emd.sifting.extremes.locators.
             DeltaDifferenceExtrema">
                <constructor-arg ref="maxHelper"/>
        </bean>

        <bean id="minHelper" class="hht.emd.sifting.extremes.locators.
             MinimumLocatorHelper">
                <property name="delta" value="0.25"/>
        </bean>

        <bean id="maxHelper" class="hht.emd.sifting.extremes.locators.
             MaximumLocatorHelper">
                <property name="delta" value="0.25"/>
        </bean>

        <bean id="sifter" class="hht.emd.sifting.Sifter">
                <constructor-arg ref="Or1"/>
                <constructor-arg ref="EndPointMirrorEstimator2"/>
                <constructor-arg ref="extremesLocator"/>
                <constructor-arg ref="addcrit1"/>
        </bean>

        <bean id="addcrit1" class="hht.emd.sifting.stoppingCriteria.additionalCriteria.
             EnvelopeMeanCurveMeanValue">
                <constructor-arg value="0.9"/>
        </bean>

        <bean id="addcrit2" class="hht.emd.sifting.stoppingCriteria.additionalCriteria.
             MeanDistanceFromZero">
                <constructor-arg value="0.4"/>
```

```
        </bean>

        <bean id="ZeroEstimator" class="hht.emd.sifting.extremes.estimators.ZeroPoints"/
            >
        <bean id="MirrorEstimator" class="hht.emd.sifting.extremes.estimators.Mirror"/>
        <bean id="EndPointMirrorEstimator2" class="hht.emd.sifting.extremes.estimators.
            EndPointMirror2"/>

        <bean id="Or1" class="hht.emd.sifting.stoppingCriteria.Or">
                <property name="criteria">
                        <list>
                                <ref bean="CauchyConvergence" />
                                <ref bean="MonotonicFunction" />
                        </list>
                </property>
        </bean>

        <bean id="CauchyConvergence" class="hht.emd.sifting.stoppingCriteria.
            CauchyConvergence">
                <property name="deviationThreshold" value="0.005" />
        </bean>

        <bean id="MonotonicFunction" class="hht.emd.sifting.stoppingCriteria.
            MonotonicFunction">
                <property name="diferenceThreshold" value="0.005" />
        </bean>

        <bean id="ResultsConfigs" class="data.ResultsConfigs">
                <property name="genPdfFromImfs" value="false"/>
                <property name="genPdfFromImfsMaps" value="false"/>
                <property name="genImfsAllMapPdf" value="false"/>
                <property name="genImfMapToSignalPdf" value="false"/>
        </bean>
</beans>
```

Listing 2: XML konfigurační soubor pro třídu HhtDataRunner.

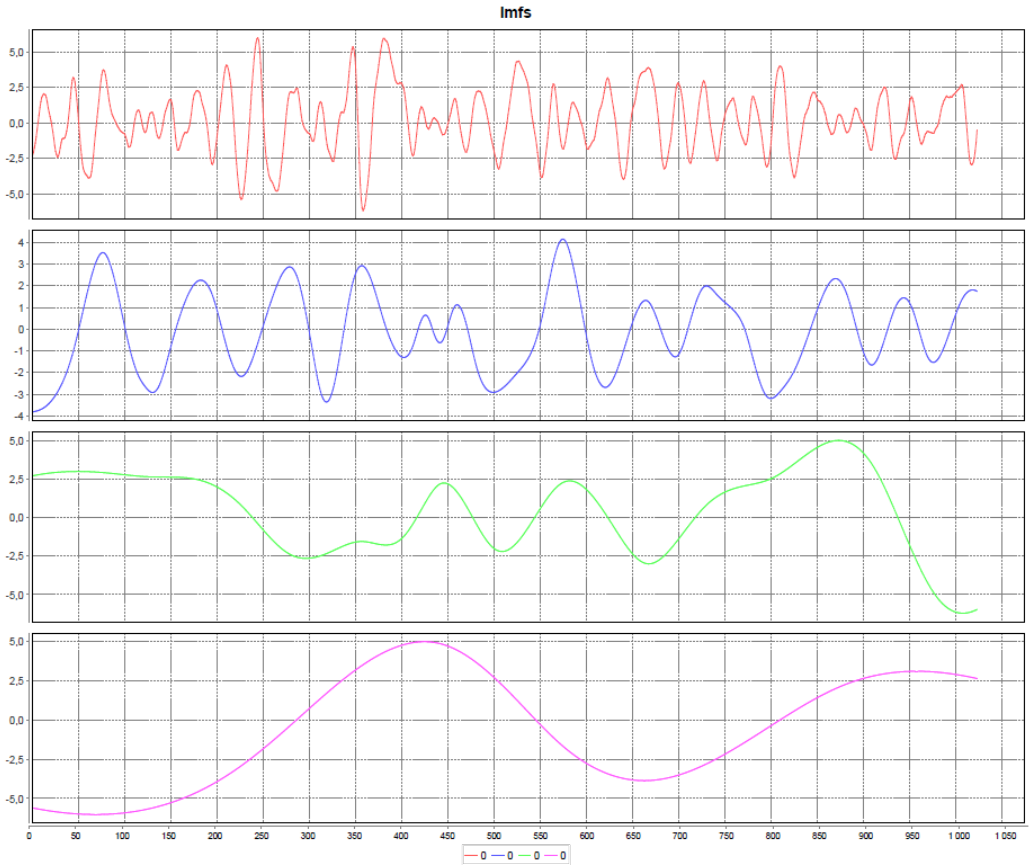
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="runnerCfg">
        <logpath>log//</logpath>
        <configsPath>configs//eemd//test</configsPath>
        <dataPath>data//eeg//preproccessedForHHT_cz</dataPath>
        <samplingFrequency>1000</samplingFrequency>
        <useNSamples>1024</useNSamples>
        <resultPath>c://hhtResults//eemdNoCutOff//imfs//all_50_0 -05</resultPath>
        <timeout>10000</timeout>
        <cfgsList>
                <name>configs//eemd//test//CauchybestEEMD.xml</name>
        </cfgsList>
        <fileNameList>
                <name>02_Cz_(baseline_)(target|nonTarget){1}\.txt</name>
                <name>5_Cz_(baseline_)(target|nonTarget){1}\.txt</name>
                <name>10_Cz_(baseline_)(target|nonTarget){1}\.txt</name>
                <name>20_Cz_(baseline_)(target|nonTarget){1}\.txt</name>
                <name>30_Cz_(baseline_)(target|nonTarget){1}\.txt</name>
        </fileNameList>
</configuration>
```

Listing 3: XML konfigurační soubor pro třídu RunClassificationAll.

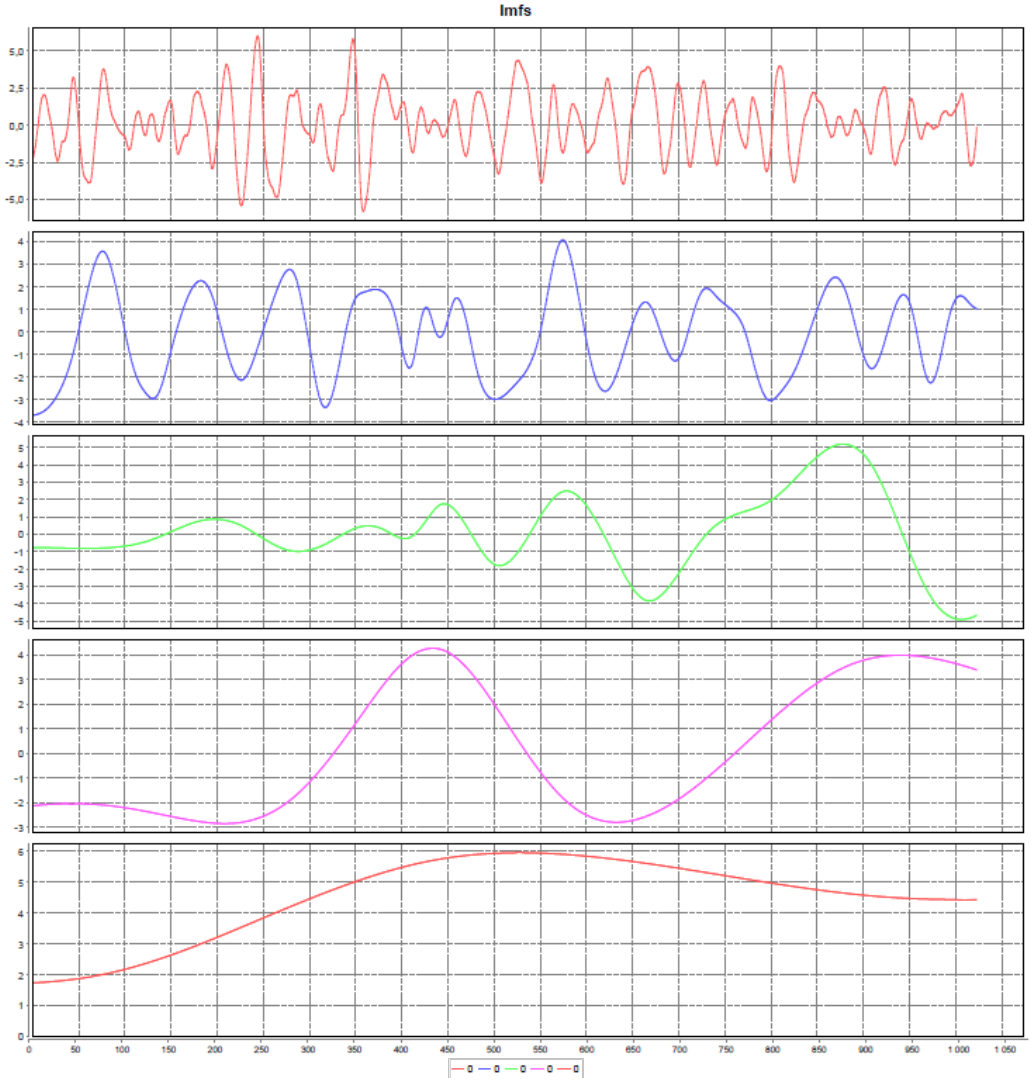
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
            xmlns:util="http://www.springframework.org/schema/util"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

       xsi:schemaLocation="
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/util http://www.springframework.
                org/schema/util/spring-util -2.0.xsd">

        <import resource="classifiers.xml"/>

        <bean id="RunClassificationAll" class="testing.classificators.
            RunClassificationAll">
                <property name="dataDirectories" ref="directories"/>
                <property name="patterns" ref="patterns"/>
                <property name="classifiers" ref="classifiers"/>
                <property name="resultsPath" value="c://hhtResults//eemdNoCutOff//
                    results//all_50_0 -05//"/>
```

```
            <property name="templateFilename" value="configs//classification//
                htmlTemplate.stg"/>
            <property name="logsDirName" value="classification"/>
        </bean>

        <util:list id="classifiers">
            <ref bean="class5"/>
        </util:list>

        <util:list id="directories">
          <value>c://hhtResults//eemdNoCutOff//imfs//all_50_0 -05//CauchybestEEMD</value>
        </util:list>

        <util:list id="patterns">
        <value>02_Cz_(baseline_)(target|nonTarget){1}</value>
            <value>5_Cz_(baseline_)(target|nonTarget){1}</value>
            <value>10_Cz_(baseline_)(target|nonTarget){1}</value>
            <value>20_Cz_(baseline_)(target|nonTarget){1}</value>
            <value>30_Cz_(baseline_)(target|nonTarget){1}</value>
        </util:list>
</beans>
```

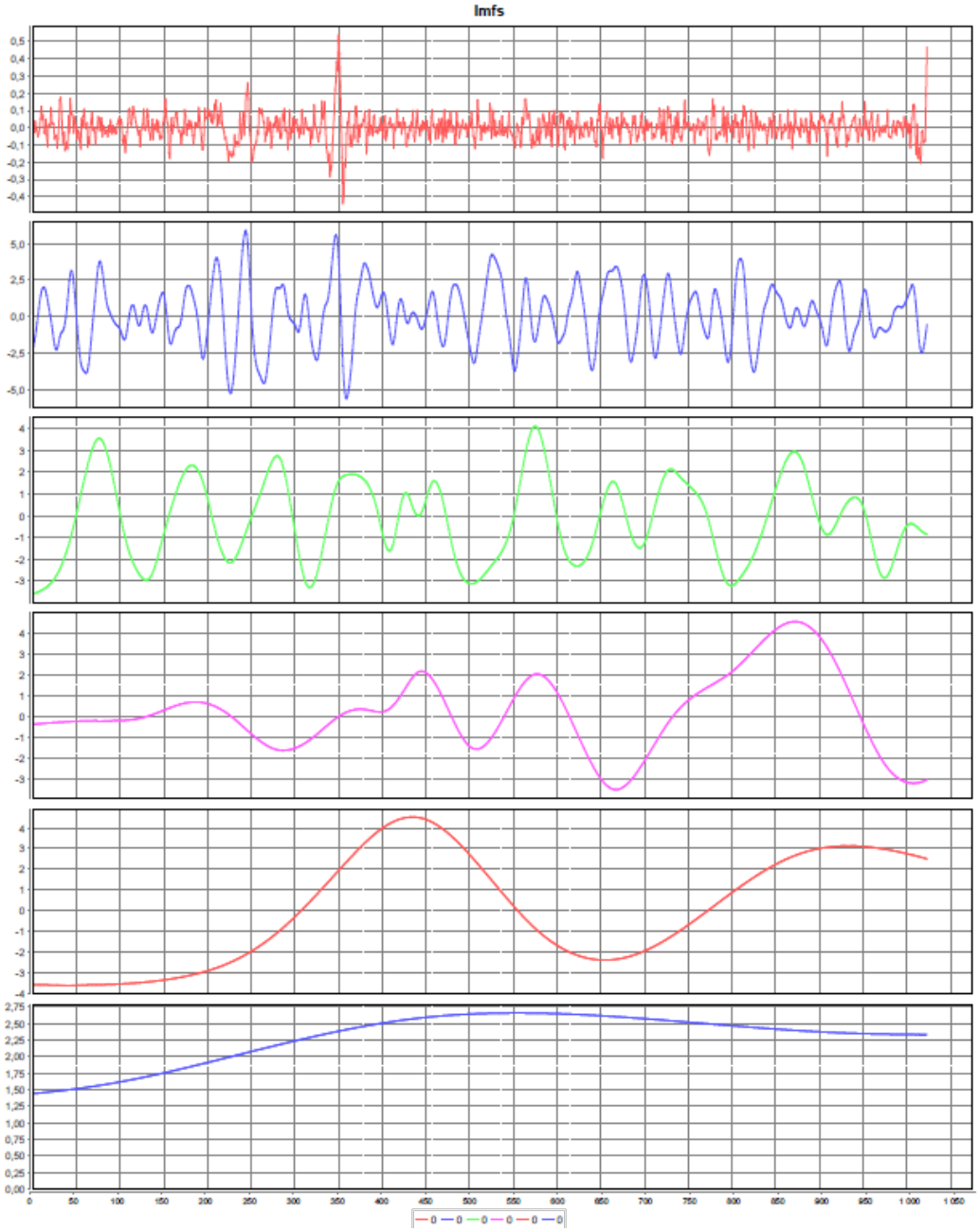
Obrázek 1: IMF získané pomocí EMD.



Obrázek 2: IMF získané pomocí EEMD (50 různých šumů, amplituda bílého šumu 0,1μV).



Obrázek 3: IMF získané pomocí EEMD (50 různých šumů, amplituda bílého šumu 0,5μV).



Literatura

[Ciniburk(2011)] CINIBURK, J. *Hilbert-Huangova transformace pro detekci evokovaných potenciálů*. PhD thesis, Západočeská univerzita v Plzni, 2011.

[EL-Bab(2001)] EL-BAB, M. F. *Cognitive Event Related Potentials During a Learning Task*. PhD thesis, University of Southampton, 2001.

[Fisch()] FISCH, B. EEG artifacts. Dostupné z: `https://wiki.umms.med.umich.edu/download/attachments/133925074/EEG+artifacts.pdf`. [Online; 1. ledna 2016].

[Kinlay()] KINLAY, D. J. Long Memory and Regime Shifts in Asset Volatility. Dostupné z: `http://jonathankinlay.com/index.php/2011/03/long-memory-and-regime-shifts-in-asset-volatility/`. [Online; 1. ledna 2016].

[Luck(2005)] LUCK, S. J. An introduction to Event-related potential technique. *The MIT Press*. 2005.

[N. E. Huang(2005)] N. E. HUANG, S. S. P. S. *Hilbert-Huang Transform and Its Applications*. World Scientific Publishing Co., 2005.

[Sek()] SEK, D. M. Examples of code relating to Fast Fourier Transform. Dostupné z: `http://www.staff.vu.edu.au/msek/FFTfile.html`. [Online; 1. ledna 2016].

[Teplan(2005)] TEPLAN, M. Fundamentals of EEG measurement. *MEASUREMENT SCIENCE REVIEW*. 2005, 2.

[Z. Wu(2009)] Z. WU, N. E. H. Ensemble empirical mode decomposition: A noise-assisted data analysis method. *Advances in adaptive data analysis*. 2009, 1.