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

Bakalářská práce

Implementace algoritmu Empirical-Mode Decomposition (EMD) pro vícerozměrná data

Plzeň 2017 Jan Vampol

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 3. května 2017

Jan Vampol

Abstract

EEG signal processing is a frequently approached problem. Commonly used methods include Wavelet Transformation and Matching Pursuit. Those methods are limited to process only single channel signals and can't be used on multichannel signals directly. This problem can be solved by applying mentioned algorithms on each channel separately. A consequence of this approach is a loss of information among channels. Several methods were proposed to work directly with multichannel signals. One of those methods is Multivariate Empirical Mode Decomposition (MEMD). It is an extension of Empirical Mode Decomposition. EMD is a method, which decomposes signal into a set of so-called Intrinsic Mode Functions (IMF). MEMD algorithm was implemented into the existing EEGHHT library and it was tested on real EEG data. Testing showed that MEMD solved the problem of information loss and also ensured the same amount of IMFs per channel.

Abstrakt

Zpracování EEG signálu je stále aktuální problém. Mezi běžně používané časově-frekvenční metody patří např. waveletová transformace nebo Matching Pursuit. Ty ale umí pracovat pouze se signálem o jediném kanálu. Jejich aplikace na každý kanál zvlášť vede ke ztrátě informací projevujících se napříč kanály. Z toho důvodu bylo navrženo několik metod, které dokážou pracovat s vícekanálovým signálem. Mezi tyto metody patří algoritmus vícerozměrné empirické modální dekompozice. Jedná se o rozšíření empirické modální dekompozice (EMD). Algoritmus rozkládá signál na tzv. vlastní modální funkce (IMF). Algoritmus MEMD byl implementován do existující knihovny EEGHHT a otestován na reálných EEG datech. Z testování se ukázalo, že MEMD řeší problém ztráty informací a také zajišťuje stejný počet IMF pro každý kanál signálu.

Poděkování

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

Obsah

1 Úvod

Na katedře informatiky probíhá výzkum elektrické aktivity lidského mozku (EEG) a vývoj metod pro zpracování a vyhodnocování naměřených dat. Výzkum je specializovaný zejména na evokované potenciály (ERP). Sběr těchto dat se obvykle provádí snímáním elektrického potenciálu z několika elektrod umístěných na skalpu pacienta. Každá elektroda snímá hodnoty na jiné části skalpu. Výsledkem takového měření je signál složený z tolika kanálů, kolik bylo pro jeho sběr použito elektrod.

Pro vyhodnocování dat se v současné době používají převážně metody, které nedokážou najednou zpracovat signál přicházející více než jedním datovým kanálem. Mezi ně patří například waveletová transformace, Matching Pursuit nebo EMD. Tyto algoritmy je možné aplikovat na každý kanál signálu samostatně. Takový přístup má za následek ztrátu informací, které jsou znatelné pouze napříč kanály. Pro EMD existuje několik rozšíření pro vícekanálová data. Hlavním cílem této práce je najít, implementovat a otestovat vhodné rozšíření EMD pro zpracování vícekanálových dat z EEG/ERP experimentů. To by mělo zvýšit přesnost analýzy naměřených dat.

První část práce se zabývá základními aspekty elektroencefalografie (EEG). Zejména EEG signálem, jeho vlastnostmi a způsoby měření. Důležitou částí je i seznámení s evokovanými potenciály (ERP).

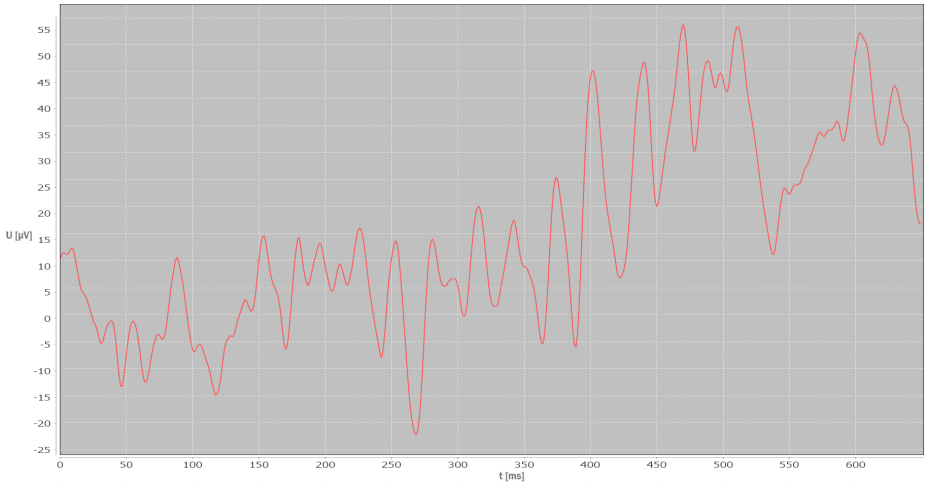
Další část práce popisuje Hilbert-Huangovu transformaci (HHT) a základní algoritmy, ze kterých se HHT skládá. Mezi ně patří empirická modální dekompozice (EMD), která rozkládá vstupní signál na tzv. vlastní modální funkce (IMF), a Hilbertova transformace pro výpočet okamžitých vlastností zpracovávaného signálu.

V poslední části jsou popsány některé rozšíření EMD pro vícekanálová data. Na základě jejich vlastností bude vybrán a implementován vhodný algoritmus do existující knihovny EEGHHT. Ta byla vytvořena v rámci disertační práce J. Ciniburka v roce 2011. Vybraný algoritmus umožní zpracování signálu složeného z více datových kanálů bez ztráty informací. Implementace bude důkladně otestována na reálných EEG datech a výsledky budou porovnány s výsledky EMD.

2 Úvod do elektroencefalografie

EEG je zkratkou pro elektroencefalografii, tedy diagnostickou metodu používanou k záznamu elektrické aktivity mozku. Neurofyziologická data naměřená při měření elektrické aktivity mozku pacienta se označují jako elektroencefalogram a měřící přístroj se nazývá elektroencefalograf. Přístroj snímá mozkovou aktivitu elektrodami umístěnými na skalpu pacienta a dokáže naměřené hodnoty zesílit.

EEG signál, který se tímto procesem získá, určuje časovou změnu elektrického potenciálu mezi dvěma elektrodami na pacientově hlavě. Naměřený signál je sumou signálů velkého množství neuronů z různých částí mozku. Intenzita signálu neuronu závisí na vzdálenosti mezi elektrodami a neuronem. Neurony blíže k elektrodám mají větší vliv na výsledný signál než vzdálenější neurony. [1]



Obrázek 1: EEG signál

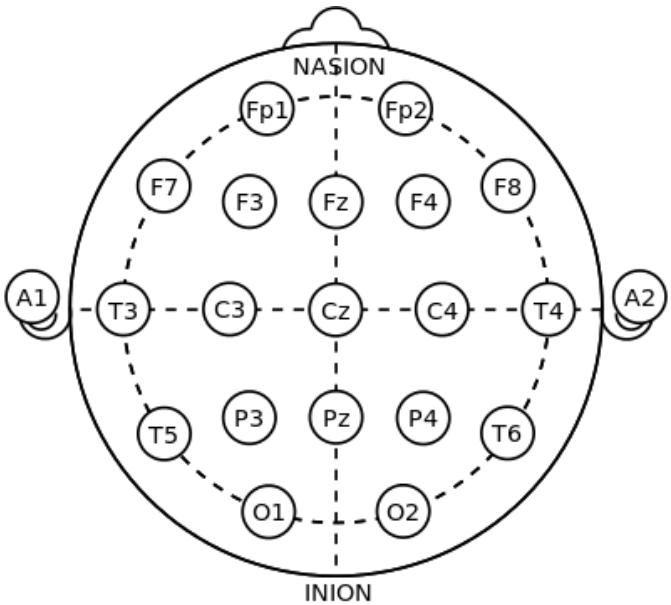
EEG se v dnešní době používá hlavně v medicíně pro vyšetření správné funkce mozku a smyslů, nebo pro monitorování pacientů v komatu. Dále se aplikuje v kriminalistice jako detektor lži. Velký potenciál má výzkum v oblasti tzv. Brain-Computer interface, který se zabývá ovládáním počítačů a jiných zařízení „pomocí myšlenek".

2.1 Měření EEG

Elektroencefalogram používaný k měření EEG se běžně skládá z:

- Elektrod
- Zesilovače
- Filtrů
- A/D převodníku
- Záznamového zařízení

Elektrody se připevňují k pacientově skalpu a získávají signál z povrchu kůže. Pro snížení impedance mezi pacientovo kůží a elektrodou se používá elektrovodivý gel. Tím se zlepší kvalita signálu pro jeho zpracování. Pro optimální umístění elektrod na správné místo na hlavě se používá systém 10/20 viz. obrázek č.2. Tento systém určuje umístění jednotlivých elektrod na skalpu. Používají se tři druhy elektrod: zemnící, referenční a aktivní.



Obrázek 2: 10/20 systém rozmístění elektrod [2]

Signál, který elektrody získají, je ale příliš slabý (v řádu μV) pro aktivaci diferenčního zesilovače. Musí se tedy před-zesílit, aby bylo možné ho

dále zpracovávat [1]. Signál v tomto stavu je ale zkreslený rušením z okolních zařízení (počítače, rentgeny apod.). Vliv tohoto rušení je možné odstranit diferenčním zesilovačem. Ten dokáže zesílit pouze tu část signálu, která obsahuje rozdíl potenciálů mezi dvěma elektrodami.

Nakonec se signál pomocí analogově-digitálního převodníku změní do diskrétní podoby. Signál se navzorkuje po stejných časových intervalech a převede do digitální formy. [1]

2.2 Vlastnosti EEG signálu

EEG signál se skládá z mnoha složek. Ty se dají rozdělit do několika hlavních kategorií:

- Mozkové vlny (rytmy)
- Fyziologické artefakty
- Vnější artefakty

Mozkové vlny reprezentují samotnou činnost mozku. Jsou vyvolané potenciály mezi neurony v mozku a reprezentují naše myšlenky, pohyby těla apod. Podle frekvence dělíme mozkové vlny na pět základních pásem: α vlny, β vlny, γ vlny, δ vlny a θ vlny. Každá z těchto vln se liší svojí frekvencí a svým účelem.

Fyziologické artefakty jsou artefakty vyvolané pacientem. Mají nežádoucí vliv na signál, protože ho zkreslují. Jedná se například o pohyb svalů, pohyb očí nebo puls srdce.

Vnější artefakty jsou způsobené okolním rušením nebo samotným měřícím zařízením. Zkreslují signál stejně jako fyziologické artefakty. Jedná se o změnu impedance elektrody (např. vlivem vysychání gelu), rušení rozvodnou elektrickou sítí, apod.

Pokud spočítáme statistické údaje EEG signálu v různých časových úsecích, zjistíme že se tyto statistiky velmi liší. Pokud by se nelišily nebo byly téměř stejné, mohli bychom o signálu říci, že je stacionární. Na základě toho můžeme stanovit, že signál je nestacionární. To je způsobeno složením signálu z mozkových vln a artefaktů. [1]

2.3 Úvod do ERP

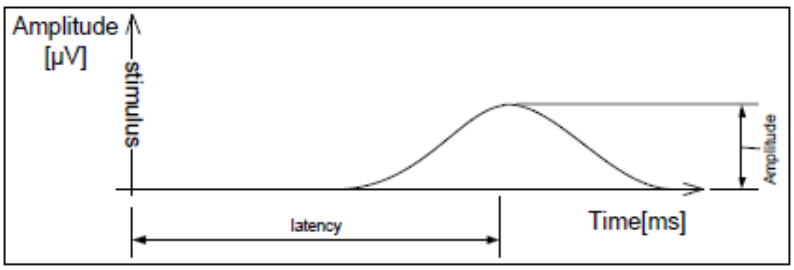
Evokované potenciály (ERP) jsou vlny v EEG signálu, které reprezentují elektrickou odezvu mozku na smyslový, kognitivní nebo pohybový podnět. Obecněji se dá říci, že je to vlnová odezva na stimul. Amplituda takových vln se pohybuje kolem 30μV a není dominantní složkou signálu (ostatní vlny mají amplitudu až 100μV). [1]

2.3.1 Vlastnosti ERP vlny

ERP vlna se popisuje třemi parametry:

- Amplituda [μV]
- Zpoždění [ms]
- Pozice

Amplituda vlny popisuje intenzitu mozkové aktivity při odezvě na stimul. Zpoždění popisuje jak dlouho trvalo mozku zareagovat na stimul. Pozice určuje nejaktivnější část mozku při odezvě. Pomocí pozice se dá určit, která elektroda bude nejlépe snímat odezvu na konkrétní stimul.



Obrázek 3: Příklad ERP vlny [1]

V ideálním případě bude vrchol amplitudy přesně na místě ERP vlny. V reálném případě artefakty vrchol amplitudy posunují, tudíž se nedá ERP vlna vrcholem amplitudy přesně popsat. Pro popis vlny se používají jiná kritéria. Místo maximální amplitudy se používá obsah plochy pod vlnou. Pro popis odezvy se používá vzdálenost k bodu v polovině vlny viz. obrázek č.3. [1]

2.3.2 Druhy ERP vln

ERP vln existuje mnoho druhů. Každá vlna je reakcí na konkrétní stimul. Dělí se na tři základní skupiny podle druhu stimulu viz. 2.3. Označují se např. P3, N2, C1. Ve většině případů písmeno ve značení znamená polaritu vlny (P - positivní, N - negativní, C - nespecifikováno) a číslo označuje pozici v celém vlnění. Označení není spojeno s mozkovou aktivitou tzn. P1 smyslová a P1 rozpoznávací nejsou stejné vlny.

Příkladem vlny je vlna P300, která vzniká při procesu rozhodování. Obvykle se očekává při tzv. odd-ball experimentech, kde je subjekt vystaven sekvencím opakovaných stimulů, které jsou náhodně přerušované odlišnými cílovými stimuly. Vlna P300 označuje rozhodovací odezvu mozku na cílový stimul (např. poznání čísla v sekvenci náhodných čísel). Nastane pouze v případě, kdy se subjekt aktivně snaží rozpoznat cílové stimuly.

3 Časově-frekvenční metody zpracování signálu

Jednou z častých metod zobrazení signálu je závislost na čase. Tato reprezentace zobrazuje hodnotu signálu (např. napětí V) jako funkci času. Z takového zobrazení je možné vyčíst základní parametry signálu, jako jsou například jeho minima a maxima. Další užitečnou reprezentací je závislost na frekvenci, která zobrazuje hodnotu signálu jako funkci frekvence. Z té je vidět, které frekvence jsou dominantně zastoupené, a které jsou pouze na pozadí.

Pro transformování EEG signálu je třeba zjistit frekvenční složení v čase, neboli provést časově-frekvenční analýzu. Z té je vidět jak se složení signálu mění v čase, což v EEG určuje, kdy vlny proběhly a jak byly výrazné. Mezi nejrozšířenější časově-frekvenční metody patří waveletová transformace a Matching Pursuit.

3.1 Waveletová transformace

Také známá jako vlnková transformace. Vznik waveletové transformace je jedním z výsledků snahy získat časově-frekvenční popis signál, který dokáže určit i polohu jevu v čase a je tedy vhodný pro nestacionární signály [4] viz. 2.2. V tomto algoritmu se používají wavelety, což jsou funkce vytvořené škálováním a posouváním mateřské wavelety. S pomocí waveletů je možné prozkoumat celý signál a zjistit jeho frekvenční složení v čase. Pro různé úlohy se volí různé mateřské wavelety, ale všechny musí splňovat některé společné rysy. WT rozdělujeme podle spojitosti v čase na spojitou WT (CWT) a diskrétní WT (DWT) [1].

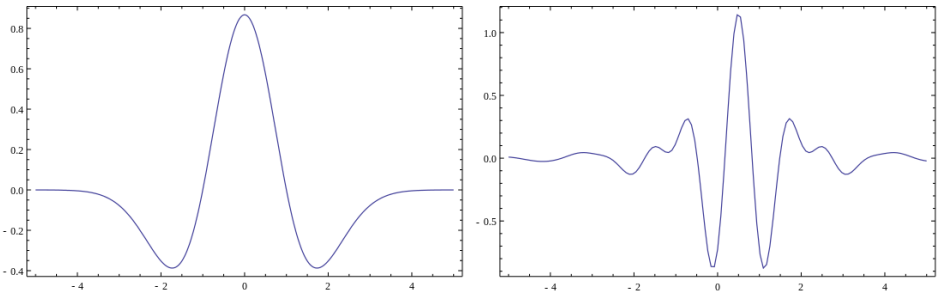
3.1.1 Wavelety

Wavelet (vlnka) se běžně popisuje jako krátká oscilace, která začíná a končí v nule. Na rozdíl od funkce jako je např. sinus, má vlnka nepravidelný tvar a existuje pro ní kompaktní nosič. Právě tyto vlastnosti dělají z vlnek dobrý

nástroj pro analýzu nestacionárních signálů. Díky nepravidelnému tvaru můžeme analyzovat i nespojité signály nebo signály s náhlými změnami. Kompaktní nosič umožňuje zjištění vlastnosti signálu.
Waveleta je dána předpisem [5]:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{3.1}$$

kde a označuje dilataci vlnky a b označuje posun vlnky od mateřské. Původní mateřská vlnka má parametry $a = 1$, $b = 0$.

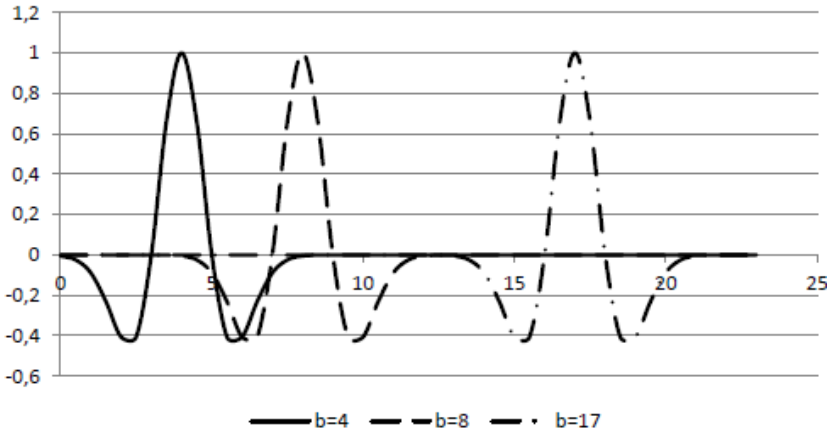


Obrázek 4: Vlnka typu Mexican hat (vlevo) a Meyer (vpravo) [3]

3.1.2 Princip WT a použití v ERP

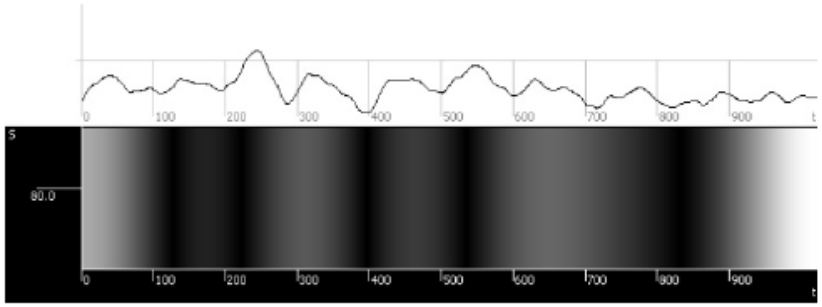
Algoritmus waveletové transformace se skládá ze 6 základních kroků:

1. Výběr mateřské vlnky (včetně určení a a b) a nastavení počáteční a koncové hodnoty posunu a dilatace, spočítání délky kroku.
2. Vypočítání hodnoty korelace vlnky vůči signálu.
3. Změna hodnoty posunu b o vypočítaný krok.
4. Pokud není posun větší než koncová hodnota posunu, jdi zpět na krok 2.
5. Změna hodnoty dilatacea a o vypočítaný krok, nastavení hodnoty posunu na počáteční hodnotu.
6. Pokud není dilatace větší než počáteční hodnota dilatace, jdi zpět na krok 2. V opačném případě konec.



Obrázek 5: Posouvání vlnky [1]

Vypočítaná korelace v každém kroku říká, jak se daná část signálu podobá vlnce. Jeho výsledkem je škálogram viz. obrázek 6. Světlejší barvy určují lepší korelaci. Tmavé barvy určují místa kde je korelace nízká. [1]



Obrázek 6: Příklad škálogramu [1]

V ERP experimentech se musí nejdříve určit místo, kde by se mohla ERP vlna nacházet. Na vybranou epochu je aplikován WT algoritmus. Vlnka reprezentující hledanou ERP vlnu se musí naškálovat tak, aby odpovídala její očekávané velikosti. Tyto úpravy je možné provést změnou parametrů a a b.

3.2 Matching Pursuit

Matching Pursuit označuje algoritmus, který hledá aproximaci signálu pomocí atomických funkcí z vybrané množiny (slovníku) funkcí. Signál je rozložen na atomy z tohoto slovníku. Každý nalezený atom je odebrán ze signálu a algoritmus pokračuje další iterací. Po zpracování je signál popsán jako suma všech vybraných atomických funkcí.

$$f(t) = \sum_{n=0}^{\infty} a_n g_n \tag{3.2}$$

Zde g_n označuje funkci ze slovníku (n je index funkce) a a_n její váhový parametr.

V EEG se často používá množina funkcí nazývající se Gaborovy atomy. Jedná se o Gaussovy křivky $g(t) = e^{-\pi t^2}$ modulované využitím sinových funkcí. Dají se popsat jako [1]:

$$g = g_{s,u,v,w}(t) = g(\tfrac{t-u}{s}) \cos(vt + w) \tag{3.3}$$

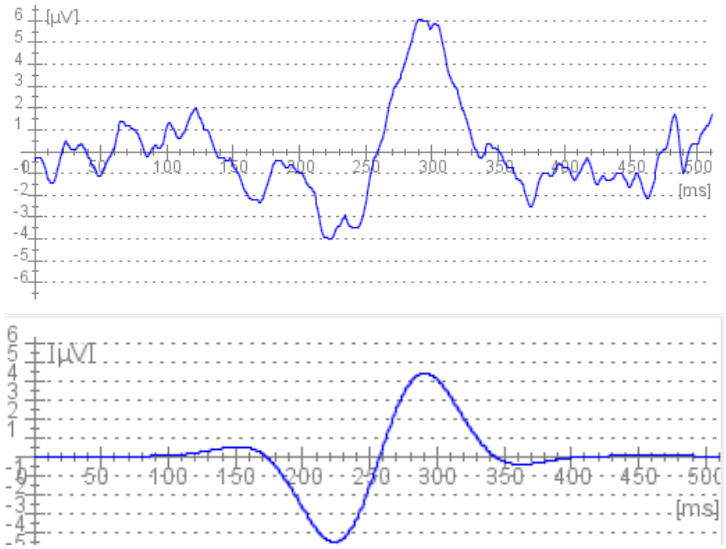
Každý atom je definován uspořádanou čtveřicí, ve které s zastupuje velikost, u posun, v frekvenci a w fázový posun [1]. Gaborovy atomy umožňují časově-frekvenční lokalizaci.

3.2.1 Matching Pursuit a ERP

Princip MP je v rozložení signálu na atomické funkce. Tím se aproximují všechny vlastnosti signálu, včetně signálových trendů. Při snímání mozkové aktivity se ERP vlny vyskytují stejně jako signálové trendy, tedy ve stejné velikosti a po stejných časových intervalech. Po několika iteracích MP je vypočítán trend signálu a s ním ERP. Čím více iterací proběhne, tím detailnější aproximaci získáme. [1]

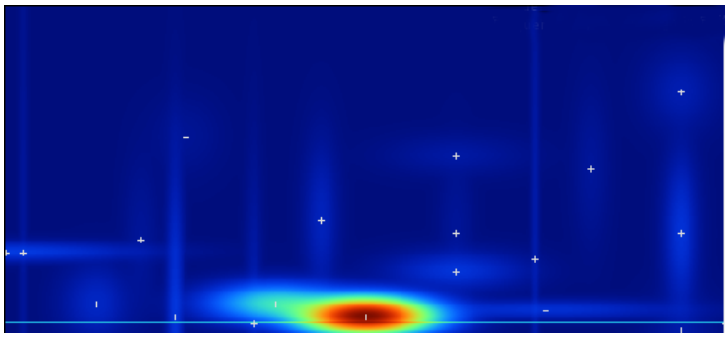
O Gaborových atomech je známo, že se dají popsat pomocí uspořádané čtveřice s, u, v, w [1]. Každý takový atom v ERP zastupuje určitou mozkovou vlnu. Při provádění MP algoritmu vznikne pro každý atom modulo, které určuje úroveň korelace atomu a signálu. Trend tohoto atomu se dá vyčíst z takového modula [1]. V ERP můžeme určit vznik signálového trendu na

místě, kde je hodnota modula vysoká. Zároveň s tím posun u určuje místo očekávaného jevu [6]. Nalezením signálových trendů pro určitý atom se zjistí zda-li se daná vlna v signálu nachází.



Obrázek 7: Naměřený EEG signál s P3 vlnou (nahoře), Gaborův atom aproximující P3 vlnu (dole) [6]

Na obrázku č.7 (nahoře) je vidět naměřený EEG signál. K němu se přiloží Gaborův atom zastupující P3 vlnu z obrázku č.7 (dole). V každém bodě se vypočítá modulo určující úroveň korelace. Výsledný škálogram obr. č.8 ukazuje podobnost signálu s hledanou P3 vlnou v každé části signálu (obrázek č.8 byl vytvořen Wigner-Villovou transformací pro zobrazení výsledku MP [6]).



Obrázek 8: Výsledný škálogram signálu obr. č.7 [6]

4 Hilbert-Huangova Transformace

Hilbert-Huangova transformace byla vytvořena pro analýzu nelineárního a nestacionárního signálu. HHT je algoritmus řízený daty. To znamená, že nepotřebujeme žádnou aproximační funkci (wavelet, atom apod.) pro analýzu signálu. Skládá se z empirické modální dekompozice (EMD) a Hilbertovy spektrální analýzy (HSA) [7], také známé jako Hilbertova transformace.

4.1 Empirická modální dekompozice

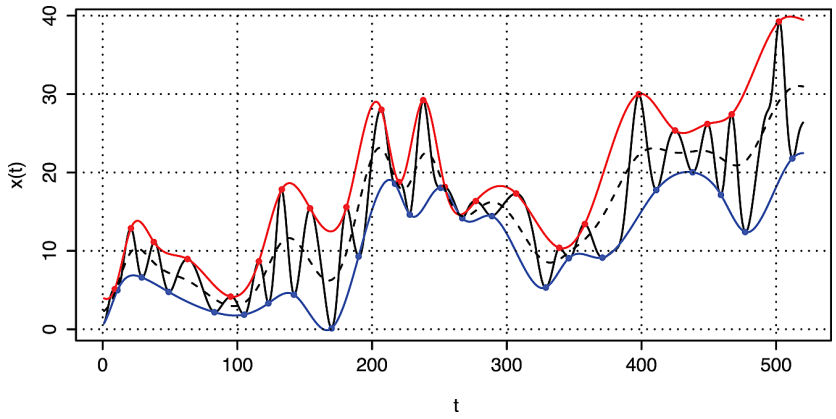
Empirická modální dekompozice je jádrem HHT. EMD stejně jako WT a MP rozkládá signál na jednotlivé komponenty. Tyto komponenty se v EMD nazývají vlastní modální funkce (Intrinsic Mode Function), které nesou vlastnosti analyzovaného signálu [7].

4.1.1 Vlastní modální funkce (Intrinsic Mode Function)

IMF jsou funkce splňující následující podmínky [12]:

1. Průměrná hodnota obálky definované lokálními maximy a minimy se rovná nule v každém bodě.
2. Celkový počet lokálních extrémů musí odpovídat počtu průsečíků s osou x, nebo se lišit maximálně o 1.

Obálky jsou definované jako spojení všech maxim nebo minim kubickou křivkou. IMF reprezentuje jednoduchou oscilační funkci podobnou harmonické funkci [1]. Liší se ale svojí definicí (viz. výše), takže je mnohem obecnější než harmonická funkce. Na rozdíl od harmonické funkce může mít proměnlivou frekveci a amplitudu.



Obrázek 9: Příklad vytvoření obálky signálu [7]

4.1.2 Algoritmus EMD

Cílem EMD je rozložení původního signálu na množinu vlastních modálních funkcí a reziduum signálu. Vlastní modální funkce obecně zastupují v signálu menšinu jeho obsahu. Vstupní data mohou obecně obsahovat více než jednu oscilační složku, a proto by jejich zpracování pomocí HSA neposkytovalo kompletní popis všech frekvenčních složek. Proces rozložení signálu je popsán podle algoritmu [1]:

1. Inicializace funkce $r_0(t)$ jako originální signál a čítač IMF $i = 1$
2. Extrakce i-té IMF
   (a) Inicializace $h_0(t) = r_{i-1}(t)$ a čítač kroků $k = 1$
   (b) Nalezení lokálních extrém (maxima a minima) signálu $h_{r-i}(t)$
   (c) Vytvoření maximové a minimové splajny (obálky)
   (d) Spočítání střední hodnoty $m_{k-1}(t)$ zprůměrováním horní a dolní obálky
   (e) Výpočet $h_k(t) = h_{k-1}(t) - m_{k-1}(t)$
   (f) Kontrola koncové podmínky IMF viz. 4.1.3
      i. Pokud je podmínka splněna pak $IMF_i(t) = h_k(t)$
      ii. Jinak $k = k + 1$ a zpět na 2b
3. Spočítání nové zbytkové funkce $r_i(t) = r_{i-1}(t) - IMF_i(t)$
4. Kontrola koncové podmínky EMD viz. 4.1.4

(a) Pokud má $r_i(t)$ alespoň 2 extrémy pak $i = i + 1$ a zpět na 2

(b) Jinak je proces ukončen s reziduem po dekompozici $r_i(t)$

Extrakci IMF ze signálu se říká sifting (česky prosévání).

4.1.3 Zastavovací podmínka siftingu

Zastavovacích kritérií iterace existuje více druhů. Různá zastavovací kritéria mají jiné účinky na různé typy signálů. Výběr kritéria je pro sifting velmi důležitý z hlediska správného rozložení signálu. Jako příklad uvádím podmínku směrodatné odchylky [12]:

$$SC = SD = \sum_{t=0}^{T} \frac{|h_{k-1}(t) - h_k(t)|^2}{h_{k-1}^2(t)} \tag{4.1}$$

Mezi další podmínky patří např. kritérium Cauchyovské konvergence [13]:

$$SC = CCT = \frac{\sum_{t=0}^{T} |h_{k-1}(t) - h_k(t)|^2}{\sum_{t=0}^{T} h_{k-1}^2(t)} \tag{4.2}$$

V případě že SC překročí nastavenou hranici, skončí iterační proces a výsledná funkce se označí jako IMF. Potom se pokračuje v dekompozici.

4.1.4 Zastavovací podmínka EMD

Zastavovací kritérium EMD určuje monotónost residua po nalezení IMF. Pokud je počet extrémů signálu větší než 2, residuum není monotóní a obsahuje další IMF. V případě, že tato podmínka není splněna, není možné pokračovat v dekompozici a proces je u konce. Obvykle je počet vlastních modálních funkcí relativně malý (tzn. menší než 10).

4.2 Hilbertova transformace

Hilbertova transformace vytvoří datovu sekvenci známou jako analytický signál z realných dat. Analytiký signál $z = x + i \cdot y$ obsahuje reálnou část x, která reprezentuje původní data, a imaginární část $i \cdot y$, která obsahuje Hilbertovu transformaci. Imaginární část je reálný signál s fázovým posunem 90 stupňů. Funkce sinus se tedy změní na cosinus a naopak [8]. Analytická funkce vypadá následovně [12]:

$$z(t) = x(t) + i \cdot y(t) = a(t)e^{i\theta(t)} \tag{4.3}$$

$$a(t) = (x^2 + y^2)^{1/2} \tag{4.4}$$

$$\theta(t) = \tan^{-1} \frac{y}{x} \tag{4.5}$$

$a(t)$ značí okamžitou amplitudu a $\theta(t)$ okamžitou fázi, okamžitá frekvence je pak:

$$\omega = -\frac{d\theta}{dt} \tag{4.6}$$

Spočítáním HT je možné určit okamžité vlastnosti signálu. Je možné vypočítat okamžitou amplitudu, frekvenci a fázi. Okamžitá amplituda je amplitudou imaginární části signálu. Okamžitá frekvence označuje rychlost změny fázového úhlu. V případě sinusoidy je okamžitá amplituda a frekvence konstantní. [1]

Když získáme všechny IMF signálu, nebude již problém s aplikací HT na každou z nich. Po provedení HT na každou IMF získáme její vlastnosti [12]. Reziduum se z HT obecně vynechává.

5 EMD pro vícerozměrná data

MEMD je rozšířením EMD pro zpracování vícekanálových dat. Metoda EMD funguje bohužel správně pouze v případě, kdy je signál vysílán jedním kanálem. Pro praktické využití této metody při snímání EEG signálu, který je vysílán více kanály zároveň, je třeba využít jinou metodu.

Použití EMD na vícerozměrný signál může v některých případech fungovat. Obecně se ale správná funkce nedá očekávat z těchto důvodů [11]:

- Standardní EMD nenajde stejný počet IMF pro každý kanál.
- IMF stejného indexu nemusí mít stejné vlastnosti pro každý kanál.
- Omezení počtu IMF pro každý kanál může mít negativní vlastnosti pro výpočet.

Vzhledem k popularitě EMD bylo v nedávných letech vytvořeno několik rozšíření a modifikací pro zpracování vícerozměrných signálů. Mezi tyto metody patří Komplexní EMD a obecná vícerozměrná EMD, které dokáže zpracovat signál o libovolném počtu kanálů. [9]

Hlavním problémem při provádění EMD na vícerozměrných datech je vypočítání střední hodnoty signálu. Vlastnosti komplexních a vícerozměrných čísel totiž zabraňují přímému výpočtu, který je uveden v kapitole 4.1 krok 2(d). Každý z uvedených algoritmů má unikátní přístup k počítání střední hodnoty.

5.1 Komplexní rozšíření EMD

Pojem komplexní EMD neoznačuje jediný algoritmus, ale celou rodinu algoritmů využívající komplexních čísel pro zpracování signálu. Většina z těchto algoritmů dokáže pracovat se signálem ze dvou kanálů. Žádná z těchto metod ale není dostatečně obecná na to, aby pracovala správně pro libovolné množství kanálů.

Komplexní metody využívají analytického signálu k tomu, aby aplikovaly EMD na obě části signálu, tedy na reálnou i imaginární část zvlášť. Toho je dosaženo rozložením vstupního signálu na dva analytické signály - jeden pro kladné frekvenční komponenty a druhý pro záporné. Tím vytvoří dvě množiny IMF, které ale nemusí obsahovat stejné množství IMF. Tato metoda se nazývá **komplexní EMD (CEMD)**. [9]

Dalším algoritmem je **rotačně-invariantní EMD (RIEMD)**. Ten využívá vlastností komplexních čísel přímo k výpočtu lokální střední hodnoty signálu. K určení této hodnoty je třeba nejprve najít společná minima a maxima pro obě složky signálu. Z průměru jejich obálek je již možné zjistit lokální střední hodnotu. [9]

Následníkem předešlého algoritmu je **bivariantní EMD (BEMD)**. Na rozdíl od RIEMD, které hledá extrémy pouze ve dvou projekcích signálu, má BEMD možnost vytvářet obálku z libovolného množství projekcí. Problémem tohoto algoritmu je výběr equidistantních vektorů pro projekce signálu. [9]

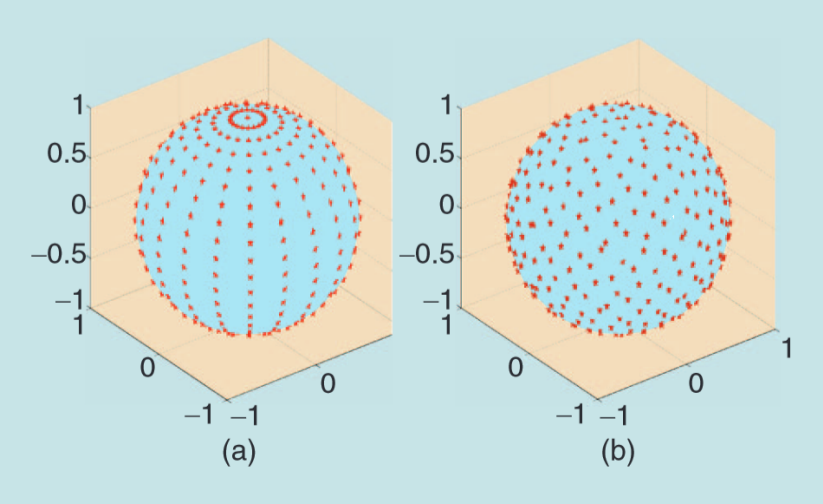
5.2 Vícerozměrná EMD

Vícerozměrná EMD (Multivariate EMD) je metoda, která dokáže zpracovat data o jakémkoliv počtu kanálů. Pro určení lokální střední hodnoty N-kanálového signálu využívá stejně jako předešlé metody průměrování obálek. Ty získá projekcí signálu libovolným počtem N-rozměrných vektorů. To algoritmu umožní zjistit extrémy v každém z těchto směrů. S jejich pomocí vytvoří obálky pro každou projekci. Střední hodnota je potom průměrem hodnot všech takto vytvořených obálek. [9]

Každý vektor zobrazuje původní signál jiným směrem viz. obrázek 10. Přesnost výpočtu lokální střední hodnoty závisí, jak na počtu vybraných vektorů, tak na vybraných vektorech samotných. Samotný výpočet je aproximací integrálu všech obálek v N-rozměrném prostoru.

$$m(t) \approx \frac{1}{V_1 V_2 \ldots V_{n-1}} \times \sum_{v_1=1}^{V_1} \sum_{v_2=1}^{V_2} \cdots \sum_{v_{n-1}=1}^{V_{n-1}} e_{\{\theta_{v_1}, \theta_{v_2}, \ldots, \theta_{v_{n-1}}\}} \quad (5.1)$$

Kde V_{n-1} označuje počet rozměrů směrového vektoru, e označuje obálku ve směru θ a $\theta_{v_{n-1}}$ označuje jednotlivé složky směrového vektoru [11].



Obrázek 10: Příklad umístění vektorů na jednotkové kouli pro trivariantní EMD (a) a multivariantní EMD (b) [9]

Hlavním problémem tohoto algoritmu je výběr vektorů, ze kterých se budou obálky počítat. V některých metodách (např. trivariantní EMD) se pro výběr vektorů používá nerovnoměrné rozdělení [9], jako je na obrázku 10(a). To ale nezaručuje správnou funkci pro libovolný počet kanálů. Používají se proto vektory vygenerované z hodnot tzv. Hammersleyho posloupnosti. Takto vytvořené vektory jsou rovnoměrně rozdělené po povrchu koule viz. obrázek 10(b).

Konečný algoritmus pak vypadá takto [11]:

1. Nalezení vektorů $\mathbf{x}_{\theta_v}$ v N-rozměrném prostoru na kouli pomocí Hammersleyho sekvence (obrázek 10(b))
2. Vypočítání zobrazení $q_{\theta_v}(t)$ signálu $s(t)$ ve směru vektorů $\{\mathbf{x}_{\theta_v}\}_{v=1}^{V}$ pro získání všech zobrazení $\{q_{\theta_v}(t)\}_{v=1}^{V}$
3. Nalezení bodů v čase $\{t^i_{\theta_v}\}_{v=1}^{V}$, které odpovídají výskytu extrémů v obrazech signálu $\{q_{\theta_v}(t)\}_{v=1}^{V}$
4. Interpolace $[t^i_{\theta_v}, s(t^i_{\theta_v})]$ pro získání vícerozměrné obálky signálu $\{e_{\theta_v}(t)\}_{v=1}^{V}$
5. Výpočet střední hodnoty $m(t)$ z obálek podle: $m(t) = \frac{1}{V}\sum_{v=1}^{V} e_{\theta_v}(t)$
6. Vytvoření nového zbytku $d(t) = s(t) - m(t)$

(a) Pokud $d(t)$ splňuje zastavovací kritérium IMF, opakuj proces pro $s(t) - d(t)$

(b) Pokud $d(t)$ nesplňuje zastavovací kritérium IMF, opakuj proces pro $d(t)$

7. Proces končí pokud rozdíl extrém a přechodů nulovou osou předávaného signálu je ≤ 1, nebo pokud je počet extrémů ≤ 3

Potom co je první IMF nalezena, je odečtena ze signálu $s(t)$ a sifting začne znovu pro již změněný signál $s(t)$. Proces se opakuje dokud $s(t)$ neobsahuje pouze reziduum původního signálu. Ve vícekanálovém signálu je takový signál určen hledáním počtů extrém v jeho projekcích. Pokud žádná projekce neobsahuje více než dva extrémy, signál je monotónní a neobsahuje už žádné IMF. Pro zastavení siftingu se můžou použít stejné podmínky jako pro EMD, nebo podmínky navržené pro vícekanálové signály viz. kapitola 5.2.2.

5.2.1 Generování vektorů

Pro generování vektorů v prostoru se používají posloupnosti s malou diskrepancí (také známé jako kvazináhodné posloupnosti). Jedná se o posloupnosti čísel, které řeší problém shlukování náhodných čísel a zároveň se nedají považovat za naprosto rovnoměrné rozdělení (mřížka). Mezi algoritmy pro počítání takových posloupností patří např. Hammerley-Halton vzorkování, Larcher-Pillichshammer vzorkování, úhlově uniformní rozdělení a další [10].

Směrové vektory používá mnoho variací EMD. Liší se v každém algoritmu množstvím rozměrů, počtem a rozložením. MEMD je v tomto ohledu velmi flexibilní. Na rozdíl od komplexních rozšíření EMD se všechny parametry mohou měnit podle potřeby.

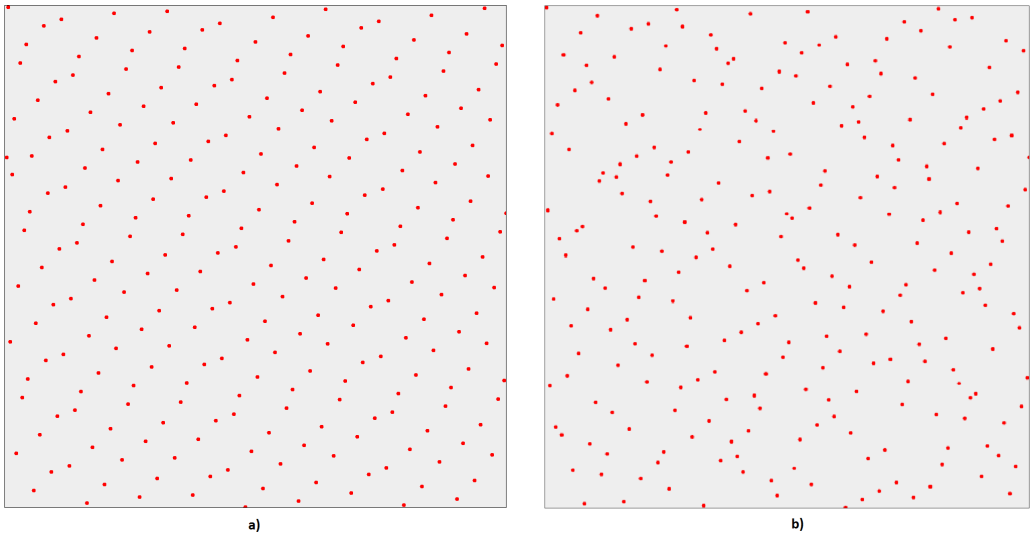
Směr a počet vygenerovaných směrových vektorů má velký vliv na přesnost výpočtu. Pokud jsou vektory nerovnoměrně rozmístěné v prostoru, v zobrazení signálů potom můžou chybět důležité informace, které se neprojevují v každém směru. Stejný problém může nastat i při výběru malého množství vektorů. Pro správnou funkci by vždy měl být počet směrových vektorů větší než počet kanálů signálu (a díky tomu i větší než počet rozměrů).

Hyperkoule

Nedílnou součástí generování směrových vektorů je tzv. hyperkoule nebo n-sféra (anglicky n-sphere). Je definována jako množina bodů, které mají v n-rozměrném prostoru od daného bodu vzdálenost menší nebo rovnou danému poloměru r. Pro generování směrových vektorů je vždy používán nulový bod jako střed a poloměr r je vždy roven jedné. Všechny směrové vektory mají vzdálenost od bodu také rovnou jedné. Nachází se tedy na povrchu n-sféry.

Hammersleyho posloupnost

V kapitole 5.2 je Hammerleyho sekvence vybrána jako nejvhodnější řešení hledání směrových vektorů. Patří mezi tzv. Kvazi Monte Carlo metody, které jsou často používané pro numerický odhad hodnoty integrálů. Výpočet střední hodnoty průměrováním obálek je odhad hodnoty integrálu, proto je možné takovou metodu aplikovat ve vícekanálovém EMD. [14]



Obrázek 11: Body vytvořené pomocí Hammersleyho posloupnosti (a) a Haltonovy posloupnosti (b) na 2D ploše

.

Metodou generování vícerozměrných bodů s malou diskrepancí je kombinace Haltonovy a Hammerleyho posloupnosti. Definujme $x_1, x_2, \ldots, x_n$ jako prvních n prvočísel. Pak můžeme označit i-té číslo jednorozměrné Haltonovi posloupnosti r_i^x jako [14]:

$$r_i^x = \frac{a_0}{x} + \frac{a_1}{x^2} + \cdots + \frac{a_s}{x^{s+1}} \tag{5.2}$$

A i jako [14]:

$$i = a_0 + a_1 \times x + a_2 \times x^2 + \cdots + a_s \times x^s \tag{5.3}$$

Hammersleyho posloupnost je možné vypočítat pouze pokud známe počet generovaných vzorků n předem. Potom je i-tý vzorek definován jako [14]:

$$\left(i/n, r_i^{x_1}, r_i^{x_2}, \ldots, r_i^{x_{n-1}}\right) \tag{5.4}$$

Body vygenervané tímto způsobem se nacházejí v intervalu $[0, 1)$. Není možné je přímo použít a je třeba je transformovat tak, aby tvořily směrové vektory na n-rozměrné hyperkouli.

Pro příklad 2-sféry (3D koule) se používá lineárního zvětšení vygenerovaných čísel do intervalu $[1, -1]$. Transformované body ale stále nejsou použitelné, protože se nenachází na povrchu 2-sféry. Musí se znovu transformovat [15]:

$$(\phi, t) \rightarrow (\sqrt{1-t^2}\cos(\phi), \sqrt{1-t^2}\sin(\phi), t)^T \tag{5.5}$$

Generování stejných vektorů na obecné n-sféře se provádí lineárním zobrazením vygenerovaných bodů do prostoru $n-1$, zjištěním jejich úhlových koordinátů a vytvoření nových vektorů ležících na n-sféře z nalezených úhlů. [15]

5.2.2 Zastavovací podmínka pro vícerozměrná data

Zastavení procesu siftingu v MEMD je podobný problém jako zastavování siftingu pro EMD. Je možné použít stejné zastavující podmínky. Z důvodu většího množství signálových kanálů se ale musí aplikovat jiným způsobem.

Pro každý kanál je vytvořena samostatná zastavovací podmínka. Všechny podmínky musí být stejného typu. Tak je možné sledovat změny na každém kanálu zvlášť a na jejich základě určovat, zda-li je třeba sifting zastavit. K vyhodnocení podmínek je možné přistupovat buď zastavením ve chvíli, kdy je alespoň jedna z podmínek splněna, nebo zastavením až při splnění podmínek pro všechny kanály.

Takový přístup má ale problémy se správným zastavením ve chvíli, kdy je počet kanálů vysoký [15]. Proto byl navržen nový způsob zastavení siftingu pro MEMD, který pracuje se všemi kanály najednou. Funguje na principu ověřovací funkce definované jako [15]:

$$f(t) = |\frac{m(t)}{a(t)}| \tag{5.6}$$

Kde definujeme $m(t)$ jako zprůměrované střední hodnoty ze všech směrů a $a(t)$ jako rozdíl všech obálek ze všech směrů:

$$a(t) = \frac{\sum_{v=1}^{V} \frac{e_{max}(t) - e_{min}(t)}{2}}{V} \tag{5.7}$$

Podmínka kontroluje poměr zprůmerovaného součtu a zprůmerovaného rozdílu obálek. Určení zastavení probíhá kontrolou všech hodnot funkce $f(t)$. Pokud je jakákoliv hodnota vyšší než zadaná hranice, pak je třeba v siftingu pokračovat. Jinak je možné signál $m(t)$ označit jako IMF.

6 Implementace

Mým úkolem byla implementace algoritmu MEMD a všeho co je potřebné pro jeho funkčnost. Algoritmus jsem implementoval do existující knihovny EEGHHT, která je napsána v Javě 1.7, proto jsem se této verze držel při rozšiřování HHT. Knihovna obsahuje implementaci HT a EMD. Některé třídy, jako je např. *Enveloper*, *Extremes* a další, jsem mohl bez modifikací použít pro stejné účely. Jiné třídy bylo třeba rozšířit nebo modifikovat. Při změnách v těchto třídách jsem kladl největší důraz na to, aby mnou vytvořené modifikace nevyžadovaly změny stávajícího kódu.

Největší část práce se skládala z implementace nových tříd a modulů potřebných pro správnou funkci a integraci MEMD do knihovny EEGHHT. MEMD je principem podobné EMD. Díky tomu jsou i nové moduly podobné jménem, strukturou i funkcí (např. Sifter - MultivariateSifter). Liší se však strukturou vstupních a výstupních dat, způsobem zastavování siftingu a také potřebou uplně nových modulů.

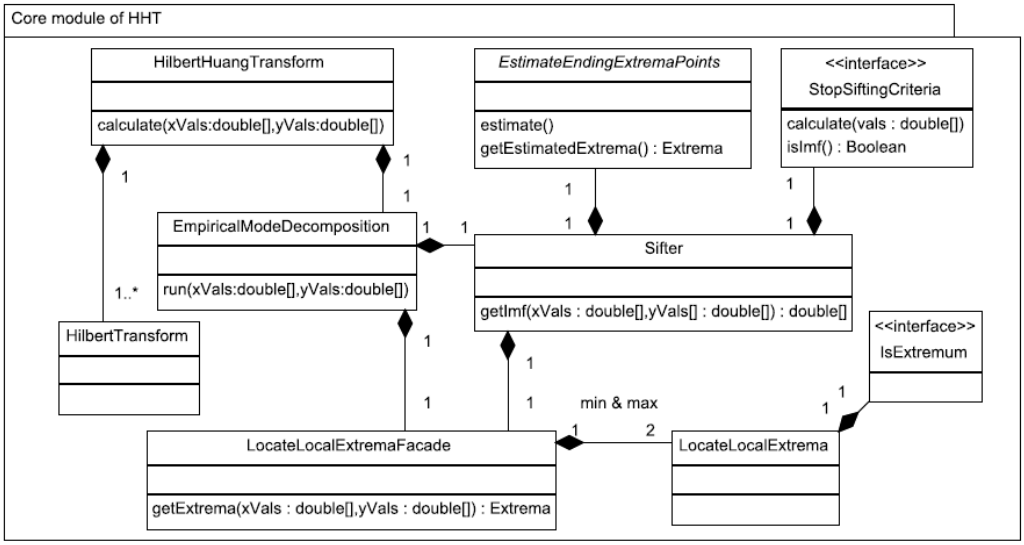
Každý mnou vytvořený modul obsahuje komentáře typu javadoc. Během implementace jsem využíval verzovací nástroj Git zajištěný webovou službou BitBucket[1]. UML diagram vytvořených modulů je v příloze C na obr. 1.

6.1 Knihovna EEGHHT

Knihovna EEGHHT vznikla v rámci disertační práce J. Ciniburka v roce 2011 na Západočeské univerzitě v Plzni. Implementace HHT byla v té době volně přístupná pouze jako knihovna do Matlabu. Je vhodná pro testování a aplikaci HHT v jiných programech napsaných v Javě. Skládá se ze tří hlavních částí [1]:

1. Jádro EEGHHT
2. Logování a vizualizace
3. Testování

[1]Dostupné na https://bitbucket.org/jciniburk/eeghht.git



Obrázek 12: Diagram tříd jádra knihovny EEGHHT [1]

V následujících kapitolách se budu stručně zabývat jádrem knihovny EEGHHT, protože účelem mojí práce je rozšíření jádra EEGHHT o druhou metodu rozkladu signálu na IMF. Detailní popis tříd je možné najít v [1].

6.1.1 Jádro knihovny EEGHHT

Jádro knihovny obsahuje všechny části nutné pro správnou funkci HHT. Patří mezi ně detekce extrémů, zastavovací kritéria a odhad koncových bodů. Knihovna je navržena tak, aby bylo možné tyto třídy jednoduše nahradit. Náhled části UML diagramu je na obrázku 12.

Třída EmpiricalModeDecomposition

Tato třída provádí empirickou modální dekompozici na vstupních datech. Výsledné IMF uloží do instančního atributu typu *Vector<double[]>*. Pro správnou funkci potřebuje instace tříd LocateLocalExtremaFacade pro hledání extrémů v signálu a Sifter, který provádí dekompozici voláním metody *run(. . .)*. [1]

Třída Sifter

Sifter vykonává na vstupních datech proces zvaný sifting, který vrátí nalezenou IMF v poli typu double. Pro sifting vyžaduje instance tříd EstimateEndingExtremePoints, StopSiftingCriteria a LocateLocalExtremaFacade. Sifter dále vnitřně využívá třídu Enveloper pro vytvoření obálek. IMF se vypočítají voláním metody *getImf(. . .)*. [1]

Třída LocateLocalExtremaFacade

Třída hledá ve vstupním signálu lokální maxima a minima. Toho docílí vnitřním využitím třídy LocateLocalExtrema, která projde vstupní signál a pomocí rozhraní IsExtremum zjistí, zda-li bod nepatří do lokálních extrémů. Vzhledem k tomu, že jsou obálky dvě, musí tyto rozhraní využít také dvě. [1]

Rozhraní StopSiftingCriteria

Rozhraní obsahující všechny nutné metody pro zastavení procesu siftingu. Proces musí skončit, pokud je nalezena IMF. V knihovně jsou implementované obě podmínky z kapitoly 4.1.3 třídami StandartDeviation a CauchyConvergence.

Třída EstimateEndingExtremePoints

Jedná se o abstraktní třídu obsahující informace o signálu a soubor metod potřebných pro odhadnutí koncových bodů signálu. Prodloužení obálek je provedeno voláním metody *estimate(. . .)*, která přidá k extrémům další body na začátek a konec signálu. Tyto body se následně použijí při vytváření instance třídy Enveloper. Druhou důležitou metodou je *getNewInstance(. . .)*, která vytvoří novou instanci se změněným signálem a tím resetuje funkci třídy.

Třída HilbertTransform

Třída provádí Hilbertovu transformaci na jednotlivých IMF voláním metody *calculate(double[] IMF)*. Nejprve vypočítá analytický signál Rychlou Fourierovou transformací. Z analytického signálu následně určí okamžité vlastnosti signálu. Ty jsou uložené jako atributy této třídy.

Třída HilbertHuangTransform

Tato třída provádí Hilbert-Huangovu transformaci využitím všech výše uvedených tříd. Nejprve pomocí EMD získá všechny IMF a následně na každou z nich aplikuje HT. Výsledky jsou uložené v seznamu instancí třídy HilbertTransform, jedna pro každou IMF. [1]

6.2 Multivariate EMD

Hlavní část algoritmu MEMD je implementována ve dvou třídách MultivariateEMD a MultivariateSifter. Dají se považovat za protějšek ke třídám EMD a Sifter, které jsou využívány v EMD. MultivariateEMD ale pracuje s daty ve formátu vícerozměrného pole a také umožňuje přidání generátoru vektorů.

6.2.1 Třída MultivariateEMD

Třída MultivariateEMD obaluje celou implementaci algoritmu MEMD tak, aby se dala využít ve třídě HilbertHuangTransform. Ta byla rozšířena o metodu *calculateMultivariate(...)*, která dovoluje provedení MEMD na všechny datové kanály zároveň na rozdíl od nyní používané metody *calculate(...)* dovolující pouze výpočet po jednotlivých kanálech.

Pro vytvoření instance MultivariateEMD jsou potřebné instance tříd MultivariateSifter, LocateLocalExtremesFacade a VectorGenerator. Ty si třída nastaví do svých atributů při volání konstruktoru. MultivariateEMD obsahuje ještě jeden atribut navíc *Vector<double[][]> imfs*, do kterého se během výpočtu ukládají výsledné IMF. Po provedení MEMD se můžou

funkce z instance třídy získat metodou *Vector<double[][]> getImfs()*. Spuštění MEMD se provádí metodou *run(double[] xVals, double[][] yVals)*. Při volání této metody se do parametru *xVals* přiřadí hodnoty časových úseků podle vzorkovací frekvence signálu a do parametru *yVals* všechny hodnoty z datových kanálů ve dvourozměrném poli. Volání metody *run(...)* může vyvolat generickou výjimku při neočekávaném ukončení práce sifteru.

Metoda *run(...)* přímo vykonává čtyři kroky popsané v kapitole 5.2. Prvním z nich je vygenerování směrových vektorů voláním metody *calculateDirections()*. Podrobnější vysvětlení tohoto procesu bude v kapitole 6.3. Dalším krokem je spuštění siftingu nad vstupními daty, které najde ve vstupních datech IMF. Tyto funkce se potom uloží do *imfs* a odečtou od vstupních dat. Posledním krokem je spočítání zastavovací podmínky pro MEMD popsané v posledním kroku algoritmu 5.2. Pro to je vytvořena metoda *boolean checkAllExtremes(...)*.

6.2.2 Třída MultivariateSifter

Tento modul provádí proces prosívání popsaný v algoritmu 5.2 (krok 2 až 5). Využívá k tomu instance tříd MultivariateStopSiftingCriteria, EstimateEndingExtremePoints, LocateLocalExtremesFacade a VectorGenerator, které získá voláním kontruktoru. Vnitřně dále využívá třídy Extremes a Enveloper. Instance tříd LocateLocalExtremesFacade a VectorGenerator sdílí s nadřazenou instancí třídy MultivariateEMD.

Proces siftingu se spustí po zavolání metody *double[][] getImfs(...)*. Z parametrů této metody získá vstupní data a vytvoří si jejich kopii, se kterou bude dále pracovat. Pro každý směrový vektor získaný z instace třídy VectorGenerator se vytvoří zobrazení vstupního signálu metodou *double[] translate(...)* ze třídy Utils.Signal. V každém z těchto zobrazení jsou potom nalezeny extrémy určující časové úseky, ze kterých se berou hodnoty pro obálky z každého kanálu vstupního signálu. K nalezení extrémů se používá metoda *Extremes getExtremes(...)* ze třídy LocateLocalExtremesFacade. Z nalezených extrémů se musí dále určit koncové body ohraničující celý signál. K tomu je použita třída EstimateEndingExtremePoints. Funkce těchto dvou tříd je detailně popsána v kapitole 6.4.2. Posledním krokem při práci se zobrazením signálu je nalezení jeho obálek metodou *calculate()* zavolanou nad instancí třídy Enveloper. Střední obálka je potom uložena v atributu Enveloperu a může se získat voláním metody *double[] getEnvelopesMeanCurve()*.

Všechny nalezené obálky se následně zprůměrují. Tím vzniknou nové funkce, které jsou výsledkem siftingu a potenciálně IMF. Posledním krokem je kontrola zavoláním metody *calculateAll(...)* nad instancí třídy MultivariateStopSiftingCriteria. Ta zjistí zda-li jsou spočítané střední hodnoty IMF. Výsledek výpočtu je potom možné získat zavoláním metody *boolean isImf()*. Pokud funkce vyhodnotí hodnoty jako IMF, běh metody končí a funkce jsou navráceny jako IMF. Pokud funkce nejsou IMF, pak se obálky odečtou od lokální kopie signálu a proces se opakuje od vytváření projekcí signálu.

Během výpočtu může na několika místech v metodě *getImfs(...)* dojít k přerušení výpočtu buď neplatnou matematickou operací vyvolávající výjimku, nebo zadáním neplatných dat ukončující výpočet navrácením vstupního signálu.

6.3 Generátory směrových vektorů

Při vytváření modulů pro generování vektorů jsem se držel těchto klíčových bodů:

- Možnost určení počtu generovaných vektorů a prostoru, do kterého náleží
- Zachování stejného prostoru pro všechny generované vektory

Pro splnění těchto podmínek jsem vytvořil abstraktní třídu VectorGenerator.

6.3.1 Třída VectorGenerator

Třída VectorGenerator je abstraktní třída. Byla vytvořena jako společný základ pro všechny ostatní generátory. Zaručuje, že všechny generátory budou mít atributy *int count*, *int dimension* a *RealVector[] directions*. Ty určují počet generovaných vektorů, jejich dimenzi a jejich výsledný formát. Nedílnou součástí je abstraktní metoda *calculateDirections()*, kterou musí každý potomek implementovat. Slouží k samotnému výpočtu směrových vektorů po zadání počtu a dimenze.

6.3.2 Třída HammersleyVectorGenerator

Ve většině publikací je generování vektorů použitím Hammersleyho sekvence popsáno jako nedílá součást algoritmu MEMD. Hlouběji jsem popsal důvody v kapitole 5.2.1. Proto jsem v rámci implementace MEMD vytvořil právě tento generátor vektorů. Je možné ho vyměnit za jiný, ale výsledky MEMD nemusí být stejně přesné.

Třída HammersleyVectorGenerator je potomkem třídy VectorGenerator. Z té dědí atributy *int count*, *int dimension* a *RealVector[] directions*. Ty může inicializovat hodnotami buď v kontruktoru, nebo pomocí setterů. Z předka implementuje metodu *calculateDirections()*, ve které třídou HaltonSequenceGenerator z knihovny apache.commons.math3 vygeneruje vektory voláním metody *double[] nextVector(...)*. Vektor vygenerovaný Haltonovou posloupností se potom dá lehce upravit na Hammersleyho vektor viz. kapitola 5.2.1.

Množina takto vygenerovaných vektorů neleží ve správném intervalu a musí se několika matematickými úpravami transformat tak, aby se vektory nacházely na povrchu n-sféry. Proces je detailně popsán v kapitole 5.2.1. Pro jejich úpravu slouží metoda *ArrayRealVector findVectorCoordinates(...)*. Tato metoda je zavolána nad každým vygenerovaným vektorem. Výslednou množinu vektorů je možné získat voláním metody *getDirections()*.

6.4 Zjištění průběhu signálu

Při průběhu siftigu probíhá hledání lokálních maxim a minim ze zobrazení signálu a na jejich základě interpolace výsledných obálek. K tomu existuje v knihovně několik pomocných modulů. Patří mezi ně:

- Extremes
- LocateLocalExtremesFacade
- EstimateEndingExtremePoints
- Enveloper

Tyto moduly byly vytvořeny v rámci práce J. Ciniburka [1]. Obálky nalezené tímto způsobem v MEMD pak musí být upraveny průměrováním.

6.4.1 Hledání extrémů

Prvním krokem pro výpočet obálky je nalezení extrémů v signálu. K tomu slouží třídy Extremes a LocateLocalExtremesFacade. Třída Extremes slouží z větší části pouze jako přepravka. Obsahuje dva instanční atributy *double[][] mins* a *double[][] maxs*, do kterých se uloží pozice a hodnota každého nalezeného extrému.

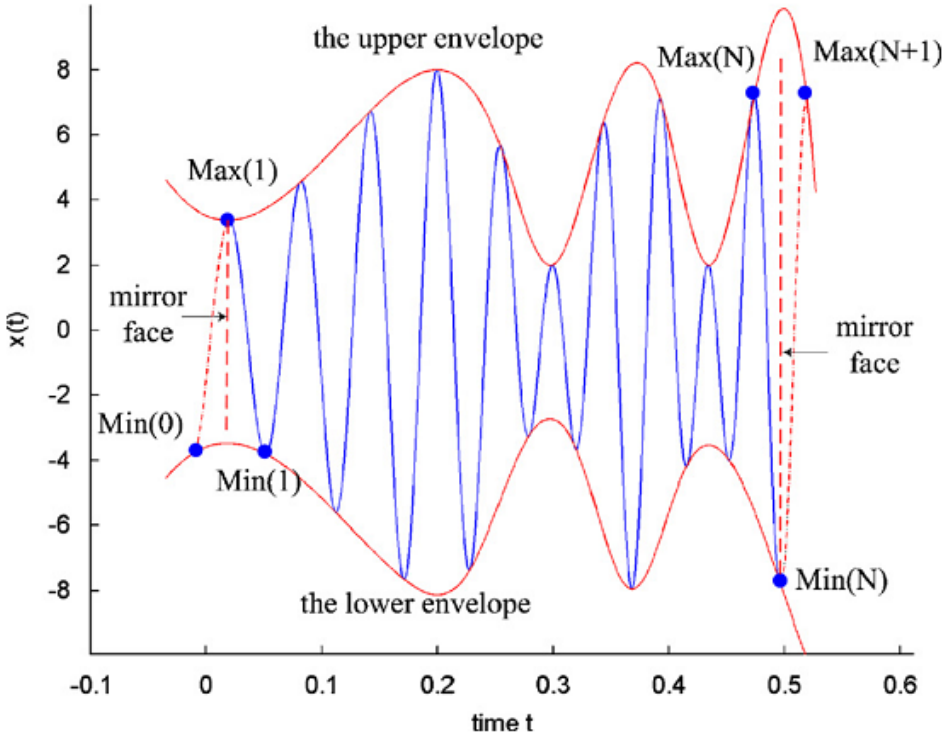
Hledání extrémů má na starosti třída LocateLocalExtremesFacade. Ta obsahuje dva instanční atributy, *LocateLocalExtremes minLocator* a *LocateLocalExtremes maxLocator*. Obě instance třídy LocateLocalExtremes slouží k hledání lokálních extrémů v signálu. K tomu jim slouží metoda *double[][] getExtremes(...)*. Instance minLocator hledá minima a instance maxLocator maxima. Nalezení obou obálek je potom zabaleno do metody *Extremes getExtremes(...)* třídy LocateLocalExtremesFacade. Nalezené extrémy jsou uloženy v instanci přepravky Extremes.

6.4.2 Odhad průběhu signálu

Po nalezení extrémů je třeba na obou koncích signálu najít nejbližší minima a maxima mimo definovaný interval. Obálka vytvořená bez takových bodů by nepokrývala celý signál a docházelo by ke ztrátám informací ze začátku a z konce signálu. Hledání takových bodů je možné realizovat několika způsoby a v knihovně pro řešení tohoto problému existuje několik modulů.

Třída EstimateEndingExtremePoints

Jedná se o abstraktní třídu sloužící jako základ pro všechny ostatní moduly. Obsahuje atributy *Extremes ex*, *double[] xVals* a *double[] yVals*. Dále obsahuje abstraktní metody *boolean canEstimate()* a *estimate()*. První jmenovaná metoda se používá k určení, zda-li je možné proces provést. To může značit počet nalezených extrémů v signálu. Voláním druhé metody se provádí odhad krajních bodů.



Obrázek 13: Zrcadlení extrémů [1]

Třída EndpointMirror

Jednou z metod hledání krajních bodů je tzv. zrcadlení bodů. Nazývá se tak proces, při kterém se krajní nalezené extrémy symetricky zrcadlí přes krajní body signálu mimo jeho interval. Výsledek tohoto proces je na obrázku 13.

Třída CustomMirror

CustomMirror jsem implementoval jako způsob odhadu průběhu signálu, který dokáže kompenzovat nepřesnosti způsobené nahrazováním nalezených extrémů za původní extrémy v algoritmu MEMD. Je implementován na základě předlohy z [15].

Tato metoda je velmi podobná zrcadlení bodů z třídy EndPointMirror. Obsahuje navíc kontrolu chování originálního a zobrazovaného signálu na začátku a na konci. Z toho důvodu byly přidány atributy *Extremes originalExtremes* a *double[] translatedSignal*. Ty je možné naplnit voláním přísluš-

ných setterů. Další odlišností je zrcadlení extrému od krajního nalezeného extrému na rozdíl od krajního bodu signálu. Vzdálenost odhadovaných bodů je potom určena ze vzdálenosti mezi následujícími extrémy. To zamezuje případům, kdy je odhadovaný bod velmi daleko od intervalu signálu.

6.4.3 Počítání obálky

Po nalezení extrémů je možné spočítat dolní a horní obálku signálu. K tomu byl vytvořen modul Enveloper. Obsahuje atributy *Extremes extremes*, *double[] minsEnvelope*, *double[] maxsEnvelope* a *double[] meanCurve*. Atribut extremes se naplní při volání kontruktoru. Tento atribut obsahuje informace o všech nalezených extrémech. Extrémy tvoří množinu bodů, které se použijí k interpolaci funkce. Voláním metody *calculate()* je provedena interpolace a atributy minsEnvelope a maxsEnvelope se naplní hodnotami vypočítaných funkcí. K interpolaci modul používá třídy z knihovny apache.commons.math.analysis - UnivariateRealInterpolator a UnivariateRealFunction.

Posledním krokem je zprůměrování horní a dolní obálky. To je automaticky provedeno po jejich spočítání a výsledná funkce se uloží do atributu meanCurve. Ten je možné z instance získat voláním metody *double[] getEnvelopesMeanCurve()*.

6.5 Zastavovací podmínka siftingu

Pro zastavení procesu siftingu je v knihovně EEGHHT několik implementovaných řešení. Všechny jsou ale určeny k zastavení siftingu EMD a nefungují pro MEMD. Implementoval jsem tedy několik řešení pro MEMD, která dokáží pracovat s vícekanálovým signálem a kontrolovat podmínky IMF ve všech kanálech zároveň.

6.5.1 Rozhraní MultivariateStopSiftingCriteria

Rozhraní MultivariateStopSiftingCriteria zajišťuje, že každá implementující třída bude obsahovat požadované metody. Mezi ně patří metoda *calculateAll(double[][] hVals)*, která je volána sifterem na konci každé iterace a slouží

k určení IMF. Nevrací výsledek, ale uloží hodnocení do atributu třídy. Výsledek je možné potom získat voláním metody *boolean isImf()*. Instanci je možné dostat do výchozího stavu metodou *reset()*.

Rozhraní také obsahuje trojici metod, které nemusí být ve všech případech implementované, protože slouží k nepovinným mezivýpočtům. Během implementace samotných podmínek zastavení siftingu se totiž ukázalo, že je vhodné takové metody zabudovat do sifteru.

První z těchto metod je *addValuesFromChannel(Enveloper enveloper)*. Ta je volána při výpočtu každé obálky signálu přicházejícího z jednoho kanálu. Je tedy volána nejčastěji. Parametrem je Enveloper obsahující všechny informace o kanálu. Druhou z nich je metoda *addValuesFromDirection(double[][][] meanValues)*. Je volána pro každý směr jednou a jako parametr dostává soubor středních hodnot z každého kanálu. Poslední je metoda *processValues(double[][][] meanValues, int directions)*, která je volána pouze jednou na konci iterace. Jako parametr dostává výsledné střední hodnoty z každého kanálu a počet směrů, kterými byl signál zobrazen.

6.5.2 Třída MultivariateCauchyConvergence

Jedna z možností zastavení siftingu je podle popisu v kapitole 5.2.2 použitím podmínek z EMD na každý kanál signálu zvlášť. Toto řešení jsem implementoval třídou MultivariateCauchyConvergence. Třída aplikuje postupně podmínku Cauchyovské konvergence (kapitola 4.1.3) na každý kanál a pouze v případě kdy jsou všechny splněny, vyhodnotí funkce jako IMF. Třída obsahuje instanční atribut *double treshold*, který určuje společnou hranici klasifikace pro všechny zastavovací podmínky.

6.5.3 Třída MultivariateEvaluatingFunction

Dalším způsobem zastavení siftingu v MEMD popsaným v kapitole 5.2.2 je použití ověřovací funkce. Pro implementaci této metody jsem musel pro výpočet funkce $a(t)$ použít všechny metody pro mezivýpočty z rozhraní MultivariateStopSiftingCriteria. Důvodem byla potřeba průměrovat hodnoty $a(t)$ (rovnice 3.2) podle počtu směrů na konci výpočtu. Třída MultivariateStopSiftingCriteria obsahuje atribut *double treshold*, který určuje hraniční hodnotu výsledku funkce $f(t)$.

6.6 Použití MEMD

Použití MEMD v jiné aplikaci je možné jednoduše realizovat následujícími kroky:

1. Vytvoření instance MEMD voláním kontruktoru. Je možné použít Bean konfiguraci voláním metody *getMemd(String cfgPath)*.
2. Zavolání metody *run(...)* nad vytvořenou instancí a vložení signálu do argumentů metody.
3. Výsledky MEMD je možné získat voláním metody *getImfs()*.

Také je možné vytvořit instanci třídy HilbertHuangTransform a jako parametr konstruktoru přidat instanci třídy MEMD. Zavoláním metody *calculateMultivariate(double[] xVals, double[][] yVals)* se provede MEMD nad signálem uloženým v parametrech xVals a yVals. Součástí této metody je i Hilbertova transformace nad nalezenými IMF. Výsledky MEMD je možné získat voláním metody *getMultivariateImfs()*.

6.6.1 JUnit testy

Pro regresní testování funkčnosti vytvořených modulů jsem implementoval několik jednotkových testů pomocí frameworku JUnit. Jedná se o třídy MultivariateSifterTest, CustomMirrorTest, StoppingCriteriaTest a HammersleyVectorGeneratorTest. Všechny testy jsou prováděny na uměle vytvořených datech.

Pro testování siftingu jsem použil signál vygenerovaný třídou SinusGenerator. Vygenerovaný signál je nakopírován do třech kanálů a předán jako argument do sifteru. Sifter by měl poznat, že se jedná o funkci sinus a označit jí jako IMF s nulovým residuem.

Ostatní testy obsahují předem vytvořená testovací data a očekávné výsledky. Výsledek vypočítaný testovaným modulem se nakonec porovná s očekávanými hodnotami.

7 Testování

Celý modul MEMD byl testovaný proti implemetanci MEMD_Supplement [15]. Při testování jsem objevil několik úprav algoritmu, které jsem následně implementoval do svých tříd. Jedná se o úpravy zastavující podmínky ověřovací funkcí a vytváření vektorů z vygenerované Hammersleyho posloupnosti. Při testování bylo zjištěno několik odlišností způsobených hlavně použitím jiného interpolátoru pro vytvoření obálek. Tyto odlišnosti nejsou kritické pro funkci modulu a výsledky ovlivňují jenom nepatrně.

7.1 Testovací data

Modul MEMD byl otestován jak na uměle generovaných datech, tak na reálných EEG datech naměřených v laboratoři. Popis měření těchto dat je v kapitole 2.1. Data byla pořízena se vzorkovací frekvencí 1 kHz. Z naměřených dat byly vyexportovány epochy a provedena baseline korekce. Ke každé epoše byl uložen testovací stimul.

Pro čtení EEG dat ze souborů jsem použil knihovnu EEGLoader[1]. EEG data se skládají ze tří souborů, které mají stejné jméno, ale jsou rozlišeny koncovkou:

- Header file .vhdr obsahuje informace o formátu uložení, vzorkovací frekvenci a informace o kanálech
- Marker file .vmrk obsahuje informace o stimulech
- EEG file .egg obsahuje samotná EEG data ze všech kanálů

7.1.1 Testovací konfigurace

Pro jednoduché spouštění je do knihovny přidáno několik spouštěcích konfigurací MEMD. Konfigurace jsou uloženy v XML souboru a k jejich aplikaci

[1]Dostupné na https://github.com/stebjan/eegloader

je využívána technologie JavaBean. Nahrávání konfigurací je provedeno metodou *getMemd(String cfgPath)*. Každá konfigurace je pojmenovaná podle parametrů siftingu. Příklady konfiguračních souborů:

```
testMemdCfgCC1.xml
testMemdCfgEF1_256.xml
```

Konfigurace se liší nastavením zastavovací podmínky a určením počtu generovaných vektorů, kterými se bude signál promítat. Byly implementovány dvě zastavovací podmínky popsané v kapitole 5.2.2. Konfigurace s ověřovací funkcí mají ve jméně písmena EF a konfigurace s Cauchyovskou konvergencí mají ve jméně písmena CC. Zastavovací hranice se ze jména určit nedá, ale každý index v názvu odpovídá jiné hodnotě (index 1 odpovídá hodnotě 0.75 pro EF). Poslední číslo v názvu určuje počet generovaných vektorů. Pokud číslo chybí, pak je konfigurace nastavena na defaultní hodnotu 64 vektorů.

V konfiguracích existují variace 16, 64 a 256 generovných vektorů. Větší množství vektorů je možné použít, ale na přesnost výpočtu už přestává mít vyšší hodnota vliv, zatímco razantně prodlužuje dobu výpočtu. Pro zastavovací podmínky existují možnosti nastavení hraniční hodnoty na 1.5, 0.75, 0.25 a 0.1. Taková nastavení zaručují dostatečnou přesnost za krátkou dobu výpočtu.

7.2 Testování MEMD

Při testování jsem využíval všechny vytvořené konfigurace. Každá konfigurace generuje rozdílné IMF. Ukázalo se, že počet IMF je závislý hlavně na nastavení zastavovací podmínky. Se zmenšujícím se hraničním poměrem se počet nalezených IMF zvyšuje (tabulka 2). Tvar nalezených funkcí je potom závislý na počtu směrových vektorů. Při nastavení počtu vektorů na 8 a 32 jsem pozoroval nalezení většího množství IMF (tabulka 1). To je způsobeno jiným rozmístěním vektorů, které má za následek nalezení dalších IMF napříč kanály. To potvrzuje, že počet a rozmístění vektorů má na funkci MEMD nezanedbatelný vliv. Ukázka konfigurace s nejlepšími výsledky je vidět v příloze A Listing 1.

Nejdříve jsem otestoval funkci mnou implementovaného algoritmu oproti impelementaci popsané v [15, 9]. Ta je napsaná v Matlabu. Díky tomu jsem mohl použít knihovnu EEGLoader k nahrání a předzpracování dat stejným

počet vektorů	8	16	32	64	128	256
IMF	7	6	7	6	6	6

Tabulka 1: Výsledky testování vlivu počtu směrových vektorů na počet nalezených IMF

hranice	1.5	1.0	0.75	0.25	0.1
IMF	6	6	7	9	13

Tabulka 2: Výsledky testování vlivu nastavení hraniční hodnoty v zastavovací podmínce na počet nalezených IMF

způsobem jako v mojí implementaci. Výsledné funkce a jejich porovnání jsou vidět v příloze B na obrázku 2. Počet nalezených IMF při spuštění s indentickými parametry je stejný a výsledné IMF se liší pouze nepatrně ve svém průběhu. Extrahované frekvence, amplitudy a fáze jsou téměř stejné. Odlišnosti jsou způsobeny použitím jiného interpolátoru pro tvoření horních a dolních obálek signálu.

Dále jsem testoval odlišnosti v IMF nalezených algoritmem EMD oproti IMF nalezených algoritmem MEMD. Zde jsem narazil na problém, kdy není možné určit ekvivalentní konfigurace pro MEMD a EMD. Pro EMD jsem se rozhodl použít konfiguraci, při které bylo dosaženo nejlepších výsledků [1] a pro MEMD jsem použil konfiguraci z přílohy Listing 1. Testování proběhlo

	Index kanálu	EMD	MEMD		EMD	MEMD		EMD	MEMD
Signal 1	1	6	6	Signal 2	6	8	Signal 3	5	7
	2	5	6		6	8		7	7
	3	6	6		6	8		7	7

Tabulka 3: Výsledky porovnání EMD a MEMD. Hodnoty v šedých polích určují počet nalezených IMF

na třech trojkanálových EEG signálech. Výsledky testu jsou v tabulce 3. Z výsledků je vidět, že MEMD vždy najde stejné množství IMF pro všechny kanály. Také nachází více IMF než EMD. To ale může být způsobené neekvivalentní konfigurací.

V příloze B na obrázku 1 jsou zobrazené IMF extrahované z jednoho ze signálů. Na obrázku je vidět problém s extrahováním IMF pomocí EMD, kdy počet funkcí z prvního kanálu nesedí z ostatními kanály. Takto extrahované IMF si potom po indexech neodpovídají a nebo uplně chybí. IMF extrahované využitím MEMD obsahují vždy podobné harmonické komponenty a proto zlepšují přesnost výpočtu HT.

8 Závěr

V rámci bakalářské práce jsem se seznámil s aspekty měření EEG signálu a se základy ERP experimentů. Dále jsem prostudoval problematiku časově-frekvenční analýzy signálu a dvě časově-frekvenční metody - waveletovou transformaci a Matching Pursuit. Tyto metody tvoří základ časově frekvenční analýzy pro EEG signál.

V posledních kapitolách teoretické části práce jsem se zabýval Hilbert-Huangovou transformací, která je hlavním zaměřením mé práce. Blíže jsem se seznámil se všemi součástmi HHT a hlavně s problematikou EMD. Podařilo se mi nalézt modifikace EMD, které vyhovují zadání mé práce. Tyto modifikace jsem prozkoumal a vybral vícerozměrnou EMD jako vhodný algoritmus pro implementaci. MEMD nejlépe vyhovuje zadání díky tomu, že počet zpracovávaných kanálů touto metodou je libovolný. To je dobrá vlastnost pro zpracování EEG signálu, jelikož různé ERP experimenty vyžadují rozdílné množství datových kanálů, a proto bude možné MEMD používat pro všechny experimenty.

Knihovnu EEGHHT jsem rozšířil o implementaci MEMD společně se spouštěcími konfiguracemi a jednotkovými testy. Také jsem implementoval několik tříd zlepšujících funkci MEMD. Mezi ně patří vícekanálové zastavovací podmínky a vícekanálové estimátory koncových hodnot signálu. Všechny moduly jsem důkladně otestoval. Z testů byly zřejmé výhody MEMD oproti EMD. Počet IMF pro každý kanál je při použití MEMD vždy stejný a IMF stejného indexu mají podobné harmonické vlastnosti.

Cíle práce vytyčené na začátku byly splněny. V rámci dalšího vývoje by bylo dobré provést optimalizaci algoritmu, protože jeho časová náročnost je daleko vyšší než u EMD. Také by bylo vhodné otestovat vliv MEMD na klasifikaci ERP komponent.

Přehled zkratek

EEG - Elektroencefalogram
ERP - Event-related Potential
EMD - Empirical Mode Decomposition (empirická modální dekompozice)
IMF - Intrinsic Mode Function (vlastní modální funkce)
HT - Hilbert Transform
HHT - Hilbert-Huang Transform (Hilbert-Huangova transformace)
MEMD - Multivariate Empirical Mode Decomposition

Literatura

[1] Ing. Jindřich Ciniburk, *Hilbert-Huangova transformace pro detekci evokovaných potenciálů*, disertační práce, Západočeská univerzita v Plzni, 2011.

[2] Wikipedia The Free Encyklopedia, *10-20 System (EEG)*, [online], Dostupné na: https://en.wikipedia.org/wiki/10-20_system_(EEG), 2015.

[3] Wikipedia The Free Encyklopedia, *Wavelet*, [online], Dostupné na: https://en.wikipedia.org/wiki/Wavelet, 2015.

[4] M. O. Berger, *12th International Conference on Analysis and Optimization of Systems. Images, Wavelets and PDEs*, Paris, Červen 1996.

[5] C. Valens, *A Really Friendly Guide to Wavelets*, [online], Dostupné na http://agl.cs.unm.edu/∼williams/cs530/arfgtw.pdf, 1999.

[6] T. Řondík, *Použití Matching Pursuit algoritmu s vlastním slovníkem pro detekci ERP v EEG signálu*, Proceedings of the 10th Conference Kognice a umělý život, strana 329-332, Bezručovo náměstí 13, Opava, Czech Republic, 2010. Slezská univerzita v Opavě, Filozoficko-přírodovědecká fakulta v Opavě.

[7] Zdeněk Mžourek, *Rozklad signálu pomocí transformace typu EMD*, bakalářská práce, VUT Brno, 2012.

[8] MathWorks, *hilbert*, [online], Dostupné na: http://www.mathworks.com/help/signal/ref/hilbert.html, 2015.

[9] Kasabov, *Handbook of Bio-/Neuro-Informatics*, strana 745-755, Springer, ISBN: 978-3-642-30573-3, 2014.

[10] Josef Pelikán, *Náhodné rozmist'ování bodů v rovině*, Computer Graphics Group, KSVI MFF, Charles University, [online], Dostupné na: http://cgg.mff.cuni.cz/∼pepca/papers/placementPelikan2014.pdf.

[11] Danilo P. Mandic, Naveed ur Rehman, Zhaohua Wu a Norden E. Huang, *Empirical Mode Decomposition-Based Time-Frequency Analysis of Multivariate Signals*, strana 74-86, IEEE SIGNAL PROCESSING MAGAZINE, Listopad 2013.

[12] Norden E. Huang a et. all, *The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences[454], 1998.

[13] N Huang a Nii O. Attoh-Okine, *The Hilbert-Huang Transform in Engineering*, CRC Press, 2005.

[14] Danilo P. Mandic a Naveed ur Rehman, *Multivariate empirical mode decomposition*, Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, UK, [online], Dostupné na: https://pdfs.semanticscholar.org, 2009.

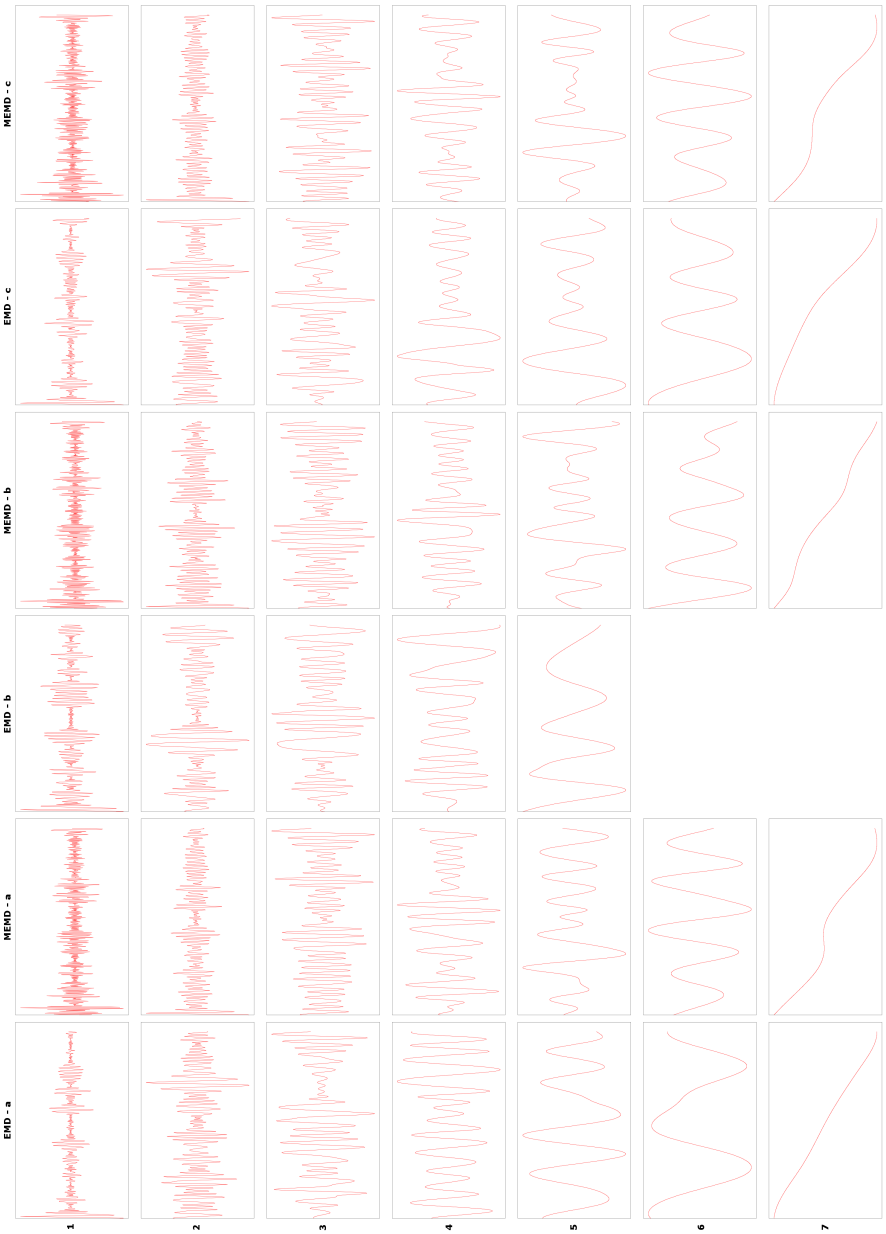
[15] Naveed ur Rehman a Danilo P. Mandic *Supplementary material to "Multivariate Empirical Mode Decomposition"*, Proceedings of the Royal Society A, 2010.

Příloha A

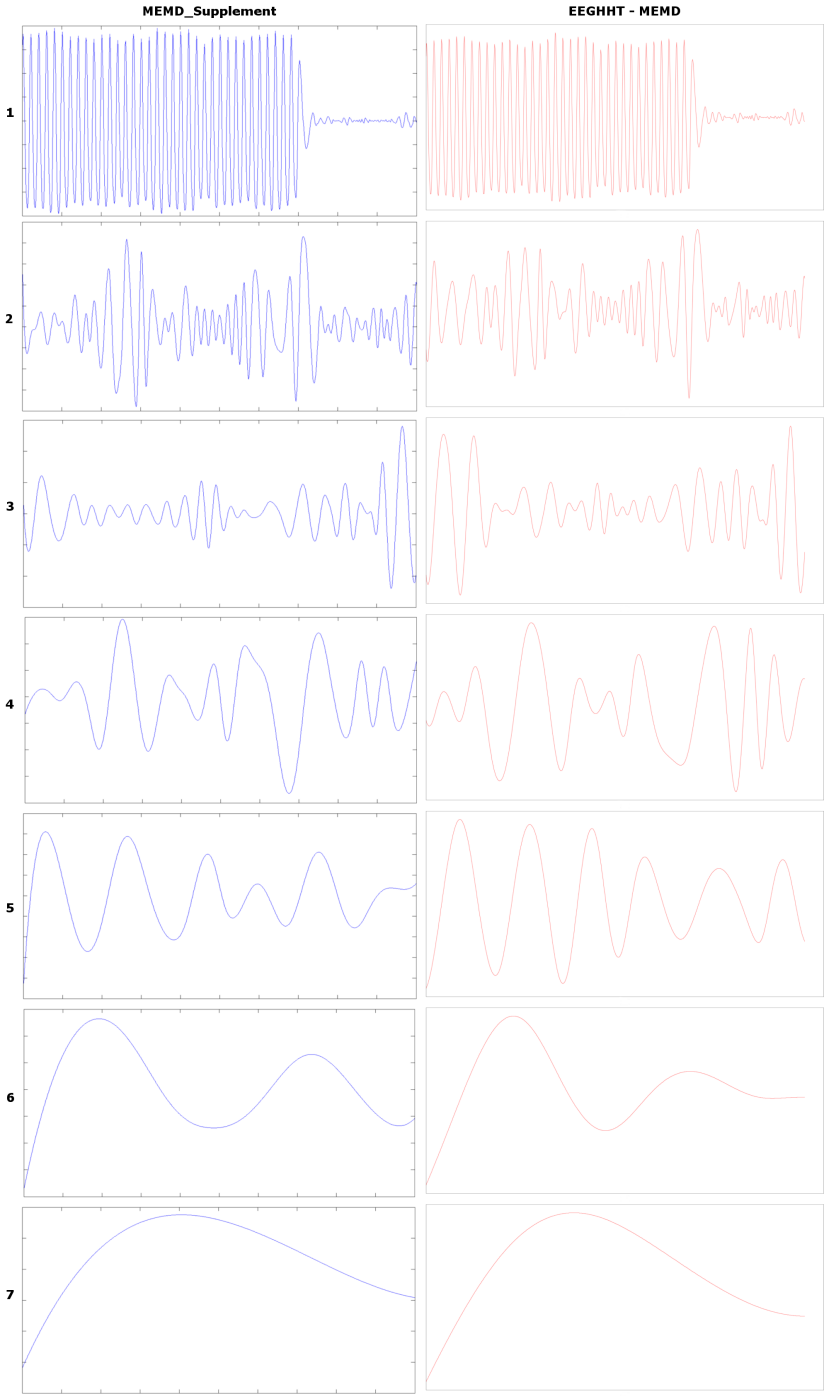
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
              http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd">

       <bean  id="memd" class="hht.memd.MultivariateEMD" >
              <constructor-arg ref="sifter"/>
              <constructor-arg ref="extremesLocator"/>
              <constructor-arg ref="vectorGenerator"/>
              <property name="counter" ref="counter"/>
       </bean>

       <bean id="counter" class="hht.emd.sifting.IterationCounter">
              <property name="max" value="10"/>
       </bean>

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

       <bean id="minLocator" class="hht.emd.sifting.extremes.locators.MinimumFinder">
       </bean>

       <bean id="maxLocator" class="hht.emd.sifting.extremes.locators.MaximumFinder">
       </bean>

       <bean id="sifter" class="hht.memd.sifting.MultivariateSifter">
              <constructor-arg ref="stoppingCriteria"/>
              <constructor-arg ref="zeroEstimator"/>
              <constructor-arg ref="extremesLocator"/>
              <constructor-arg ref="vectorGenerator"/>
       </bean>

       <bean id="zeroEstimator" class="hht.memd.sifting.extremes.estimators.CustomMirror"/>

       <bean id="stoppingCriteria" class="hht.memd.sifting.stoppingCriteria.MultivariateEvaluatingFunction">
              <property name="treshold" value="0.75"/>
              <property name="tolerance" value="0.075"/>
       </bean>

       <bean id="vectorGenerator" class="hht.memd.sifting.vectorGenerators.HammersleyVectorGenerator">
              <property name="count" value="64"/>
       </bean>
</beans>
```

Listing 1: XML konfigurační soubor pro MEMD

Příloha B

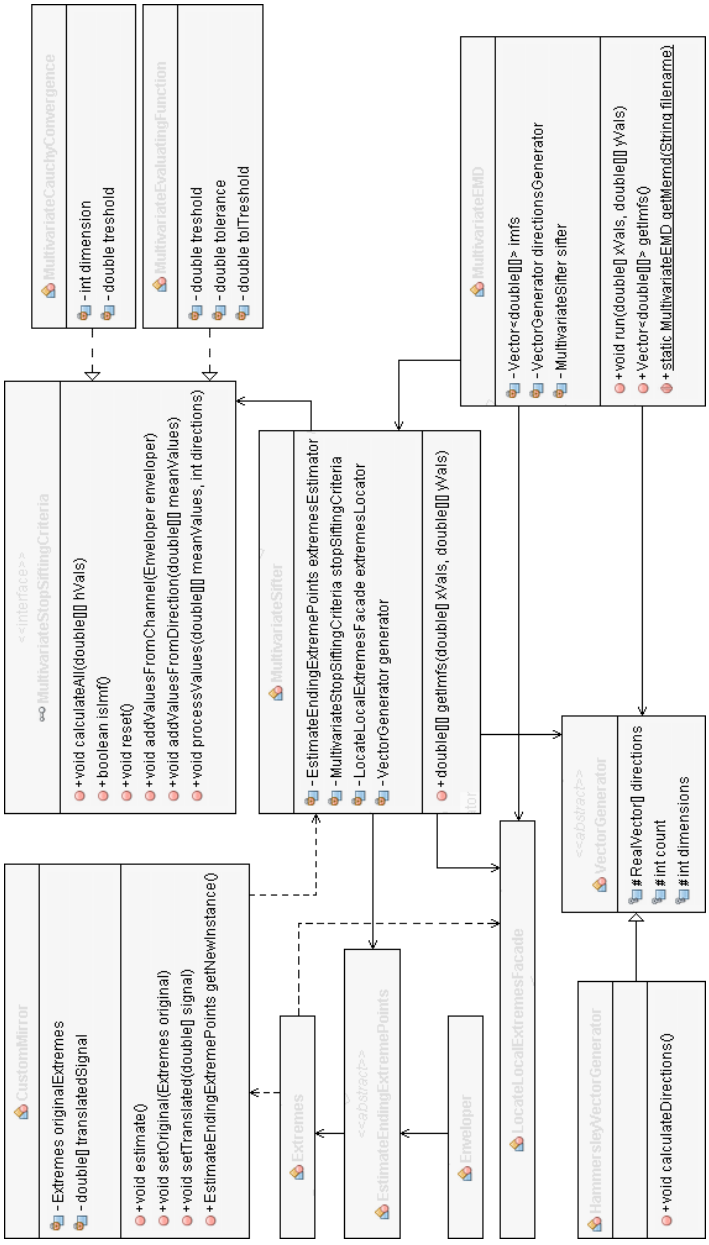


Obrázek 1: Porovnání EMD a MEMD při zpracování stejného signálu



Obrázek 2: Porovnání s implementací MEMD Supplement

Příloha C



Obrázek 1: UML diagram vytvořených tříd