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

Bakalářská práce

Implementace algoritmu matching pursuit pomocí genetických algoritmů

Plzeň, 2012 Vít Bábel

Prohlášení

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

V Plzni dne .....

Vít Bábel

Abstract

Implementation of matching pursuit algorithm using genetic algorithm

Matching pursuit is an algorithm that iteratively decomposes an input signal into a set of simple functions called atoms. The atoms are taken from a redundant dictionary to approximate the signal as well as possible. In each iteration, the selected atom is subtracted from the signal and the residuum becomes the input of following iteration. After specified number of iterations it completes a linear expansion of atoms, which can approximately reconstruct the input signal. The biggest difficulty of matching pursuit is computational cost of seeking the best fitting atom in the dictionary. The task of this bachelor thesis is to prove possibility of using Genetic Algorithms to this approach. This thesis also deals with using the fast Fourier transform to reduce the searching space. This considerably decreases computational cost to seek though it.

Poděkování

Velice rád bych poděkoval svému vedoucímu bakalářské práce Ing. Pavlu Mautnerovi, Ph. D. za jeho pomoc a velkou trpělivost při vedení mé práce.

Dále bych poděkoval svému bratrovi Bc. Ladislavu Bábelovi za téměř každodenní konzultace ohledně zpracování signálu a další cenné informace.

Vít Bábel

Obsah

1 Úvod

Matching pursuit je algoritmus, který iterativně rozkládá vstupní signál na jednoduché funkce neboli atomy. Ty vybírá z redundantního slovníku tak, aby vybraný atom co nejlépe aproximoval signál. Vybraný atom se od signálu odečte a zbytek je vstupem další iterace. Tak se po určitém počtu iterací získá lineární rozvoj, kterým se dá signál více či méně přesně rekonstruovat.

Toho se dá využít ke kompresi vstupního signálu, nebo k vyhlazení a odstranění šumu. Na oddělení neuroinformatiky[1] experimentují s využitím matching pursuit při detekci tzv. evokovaných potenciálů v EEG signálu.

Největším problémem tohoto algoritmu je náročnost prohledání často rozsáhlého slovníku atomů a nalezení atomu, který by nejlépe odpovídal vstupnímu signálu.

Úkolem této bakalářské práce je ověřit možnost využití genetického algoritmu k tomuto účelu. Genetický algoritmus je adaptivní metoda inspirovaná vývojem druhů v přírodě. Je založená na genetických procesech a principu přežití silnějšího. Napodobováním přírodního výběru, křížení a reprodukce je schopna efektivně prohledat prostor řešení a „vyvinout“ dobré řešení problému. Nemusí to být vždy optimální řešení, ale téměř vždy řešení uspokojivé.

Část této práce je věnovaná vývoji algoritmu, který k nalezení nejlepšího atomu využívá kromě genetického algoritmu ještě Fourierovu transformaci. Jejich kombinací by se měl výběr optimálního atomu ze slovníku zrychlit.

Dalším úkolem bakalářské práce je různé verze genetického algoritmu matching pursuit implementovat a porovnat jejich vlastnosti s klasickým algoritmem matching pursuit.

[1] Katedry informatiky a výpočetní techniky Fakulty aplikovaných věd ZČU v Plzni

2 Matching pursuit (MP)

Matching pursuit je metoda, kterou jako první prezentovali Mallat a Zhang[5]. Tento hladový (greedy) algoritmus dekomponuje předložený signál do lineárního rozvoje jednoduchých funkcí z *redundantního slovníku*. Tyto funkce (dále též *atomy*) jsou iterativně vybírány podle nejlepší shody se vstupním signálem. V každé iteraci se vybraný atom od aktuálního vstupního signálu odečte a zbytek je vstupem další iterace. Tak se od původního signálu stále „ukrajuje“ a předpokládá se, že zbytek konverguje k nulové funkci. Dobře je to vidět na obrázku A.1 v přílohách (str. 42).

V následující kapitole si podrobněji objasníme zákonitosti hlavní iterativní části algoritmu. Dále se stručně podíváme na slovníky funkcí. Kritickou částí algoritmu, na které závisí rychlost MP především, je ale *výběr optimálního atomu* ze slovníku. K tomuto tématu se váže oddíl 2.3.

2.1 Teorie

Mějme redundantní slovník $D = \{g_\gamma\}$ jako množinu vektorů[1], které mají jednotkovou normu. Tato vlastnost je velmi důležitá, protože vyžadujeme, aby skalární součin vstupního signálu f a libovolného vektoru g_n ze slovníku D byl délkou jejího průmětu do směru vektoru.

Délka tohoto průmětu je pak zároveň *modulem* (váhou atomu) a_n v lineární kombinaci. Takto můžeme signál pomocí všech N vektorů a jejich vah rekonstruovat.

$$f = \sum_{n=0}^{N} a_n g_n \tag{2.1}$$

Ale vraťme se zpět na začátek. V algoritmu matching pursuit budeme muset často provádět projekci vstupního signálu na vektor a počítat residuum Rf.

$$f = \langle f | g_{\gamma 0} \rangle g_{\gamma 0} + Rf \tag{2.2}$$

Takže Rf je zbytkový vektor po aproximaci. A protože Rf je *ortogonální* k

[1] Pro vysvětlení, proč je řeč o vektorech, viz začátek oddílu 2.2.

$g_{\gamma 0}$, podle Pythagorovy věty platí:

$$\|f\|^2 = |\langle f|g_{\gamma 0}\rangle|^2 + \|Rf\|^2 \tag{2.3}$$

Dále jednoduše převedeme na levou stranu prvek, který potřebujeme minimalizovat.

$$\|Rf\|^2 = \|f\|^2 - |\langle f|g_{\gamma 0}\rangle|^2 \tag{2.4}$$

Vidíme, že prvek $g_{\gamma 0} \in D$ musíme vybrat takový, aby maximalizoval $|\langle f|g_{\gamma 0}\rangle|$. Někdy ale není možné kvůli časové náročnosti výpočtu nalézt optimální vektor. Proto se musíme spokojit s nedokonalým odhadem optimálního řešení:

$$|\langle f|g_{\gamma 0}\rangle| \geq \alpha \sup_{\gamma} |\langle f|g_{\gamma}\rangle|, \tag{2.5}$$

kde $\alpha \in (0, 1\rangle$ je míra optimalizace, která je 1 pro nejlepší možnou aproximaci. V našem případě bude míru optimalizace zajišťovat *genetický algoritmus* a bude záviset na jeho parametrech.

Matching pursuit je iterativní algoritmus, který dekomponuje zbytek $R^n f$ tak, že ho projektuje na nejvhodnější vektor slovníku D. Pokud dosadíme: $R^0 f = f$, dostaneme finální iterační formuli:

$$R^n f = \langle R^n f|g_{\gamma n}\rangle g_{\gamma n} + R^{n+1} f \tag{2.6}$$

To se dá opakovat do nekonečna a je-li slovník D navržen správně, bude nám zbytek konvergovat: $lim_{M\to\infty} R^M f = 0$. Takto rozložený signál by se pak rozhodně dal opět složit (vztah 2.1) beze ztráty informace, ale taková krajnost nebývá předmětem pro matching pursuit. Nám bude obecně stačit rozložení nanejvýš na desítky atomů.

2.2 Slovník funkcí

Vstupní signál $f(t)$ je v praktických úlohách vzorkován. Hodnota t nabývá hodnot $t = 0, 1, \ldots N-1$, kde N je počet vzorků. Signál f reprezentujeme vektorem $f = (f_0, f_1, \ldots f_{N-1})$. Funkce slovníku pak musejí být reprezentovány obdobným způsobem a musí mít stejný počet vzorků. Jsou též prvky Hilbertova $\boldsymbol{L^2}$ prostoru a proto pro ně platí:

$$\|g\| = \sqrt{\sum_{i=0}^{N-1} g_i^2} < +\infty \tag{2.7}$$

Nyní jsme zároveň definovali Eukleidovskou normu, kterou využijeme i dále.

Pro aplikace MP se používají různé druhy slovníků. První MP, který představili Mallat a Zhang [5], pracuje s jedno–dimenzionálním časově–frekvenčním (time–frequency) slovníkem.

My budeme používat slovník *Gaborových atomů*, který se hodí pro aproximaci EEG signálu a identifikaci ERP komponent [6]. Atom Gaborova slovníku tvarem odpovídá Gaussově funkci modulované kosinovou funkcí. Obecné Gaborovy atomy mají následující předpis:

$$g^*_{(s,u,v,w)}(t) = e^{-\pi(\frac{t-u}{s})^2} cos(vt + w) \tag{2.8}$$

kde parametry znamenají: s – měřítko, u – posunutí, v – frekvence[2] a w – fáze atomu. Parametry Gaborova atomu se pro zjednodušení zapisují jako $\gamma = (s, u, v, w)$.

Aby metoda matching pursuit fungovala správně, musíme zaručit *normované Gaborovy atomy*. Důvod je vysvětlen na začátku oddílu 2.1.

$$g_\gamma = \frac{g^*_\gamma}{\|g^*_\gamma\|}$$

$$\|g_\gamma\| = 1$$

2.3 Způsob výběru optimálního atomu

Přestože již známe iterační formuli algoritmu matching pursuit a víme, jaké atomy budeme ve slovníku formovat, stále ještě neumíme nejlépe odpovídající atom ze slovníku vybrat. Tato část je ve skutečnosti velmi důležitá, protože určuje algoritmickou složitost i přesnost aproximace.

Musíme tedy prohledat prostor slovníku a nalézt ten atom, který bude nejlépe aproximovat zadaný signál. Prohledávaný prostor je dán parametry Gaborových atomů. Rozsah každého z těchto parametrů v praxi nějak závisí na délce vstupního signálu. A protože máme čtyři parametry, velikost prostoru můžeme řádově odhadnout jako N^4 atomů. Je to velmi přibližné a

[2]Ve skutečnosti se jedná o normovanou úhlovou rychlost. Nicméně v publikacích o algoritmu matching pursuit se už označení frekvence vžilo.

většina metod si snaží nějakým způsobem ulehčit práci, takže tento odhad nemusí platit vždy.

Použijeme-li k řešení metodu *brutální síly*, prostě postupně vytvoříme průběhy všech atomů (dosazením všech kombinací parametrů) a spočítáme jejich skalární součiny se vstupním signálem. Vybrán je atom s největším skalárním součinem. Ale tato metoda vede k algoritmické složitosti $O(N^5)$, což ji činí naprosto nepoužitelnou.

Mnohem lepším řešením je metoda, kterou publikovali S. E. Ferrando, L. A. Kolasa a N. Kovačević. Já čerpám informace z implementace a diplomové práce Ing. Tomáše Řondíka [8]. Tento přístup omezuje počet dosazených parametrů měřítka a posunutí a pro získání frekvence a fáze je využita rychlá Fourierova transformace. Složitost tak snižuje na $O(Nlog^2N)$ a aproximace je kvalitní.

To byly dvě deterministické metody, které nám vždy najdou nejlepší dostupný atom ze slovníku. Když se ale vzdáme jistoty optimálního řešení, můžeme zvážit použití některé optimalizační metody, jako jsou *horolezecký algoritmus* (hill climbing), *náhodné hledání* (random search), *simulované žíhání* (simulated annealing) nebo *genetické algoritmy* (genetic algorithms). Implementaci MP za pomoci genetického algoritmu už se věnovali například Dan Stefanoiu a Florin Ionescu [2]. Jejich práce byla inspirací i pro mou implementaci algoritmu matching pursuit.

3 Fourierova transformace (FT)

Každá transformace vytváří obraz originální funkce. V případě Fourierovi transformace je obrazem *frekvenční spektrum* $S(\omega)$ originálu $s(t)$. Originál je funkce času, zatímco obraz je funkce úhlové rychlosti. Fourierova transformace je definována následujícím vzorcem.

$$S(\omega) = \int_{-\infty}^{\infty} s(t)e^{-i\omega t}dt \tag{3.1}$$

Protože při digitálním zpracování nemůžeme použít spojitou Fourierovu transformaci, musíme si vystačit s její diskrétní variantou. *Diskrétní Fourierova transformace* (DFT) převádí navzorkovaný diskrétní signál na diskrétní frekvenční spektrum.

$$\mathbf{X}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i\frac{2\pi}{N}kn} \tag{3.2}$$

V tomto vzorci je $x(n)$ diskrétní vstupní signál. Index k je přirozené číslo, které určuje frekvenci jako násobek základní frekvence. To je taková, která má délku periody shodnou s délkou navzorkovaného signálu (první harmonická). Nultá harmonická frekvence ($k = 0$) je stejnosměrná složka (vertikální posunutí signálu). Každá další frekvence je k-násobek první harmonické. $X(k)$ je komplexní číslo, jehož absolutní hodnota představuje amplitudu a úhel fázi dané frekvence.

Oproti spojité variantě musíme počítat s jistými omezeními. Jednak v navzorkovaném signálu nejsou frekvence vyšší, než je polovina vzorkovací frekvence. Dále pak DFT dává správný výsledek jen u frekvencí, které v délce navzorkovaného signálu mají celý počet period (proto $k \in \mathbb{N}$). Z toho vyplývá že nemůžeme transformovat frekvence, jejichž perioda je delší než vstupní signál. Tím je pak jasně určeno jaké frekvence dostaneme na výstupu DFT.

Celé spektrum je pak sloupcová matice komplexních čísel $\mathbf{X}$. Pro každé k, vznikne ze vzorce dva jeden řádek ve čtvercové transformační matici $\mathbf{A}$. Sloupcová matice x je vstupní signál.

$$\mathbf{X} = \mathbf{A}x \tag{3.3}$$

$$\mathbf{A} = [W^{nk}] = \begin{bmatrix} W^{0} & \dots & W^{(N-1)\cdot 0} \\ W^{0\cdot 1} & \dots & W^{(N-1)} \\ \vdots & \ddots & \vdots \\ W^{0\cdot(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix} \tag{3.4}$$

$$W = e^{-i\frac{2\pi}{N}} \tag{3.5}$$

Vztah (3.3) ukazuje rovnici transformace. Vstupní signál x o délce N násobíme transformační maticí $\mathbf{A}$. Výstupem je sloupcová matice $\mathbf{X}$. Její výška odpovídá délce vstupního signálu. Druhý vztah (3.4) ukazuje transformační matici $\mathbf{A}$. Index k je zde číslo řádku. Index n je číslo sloupce v matici $\mathbf{A}$ a zároveň číslo vzorku v x. Oba indexy mají maximální hodnotu $N - 1$. W je základní frekvence vyjádřená komplexním číslem, které je pro dané N konstantní.

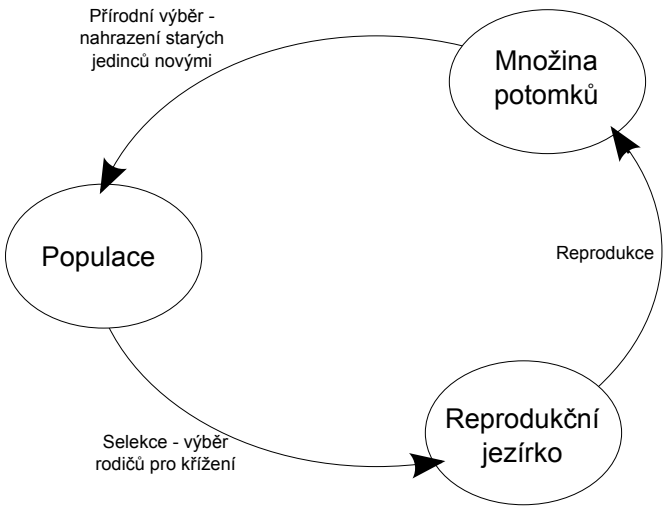
Počet frekvencí, které DFT transformuje, je $N/2$, a proto je spodní polovina matic A a X zbytečná. Dále při výpočtu této matice dochází k redundantním výpočtům. Proto má algoritmickou složitost $O(N^2)$. Pro jejich odstranění a urychlení výpočtu byl vytvořen algoritmus *rychlá Fourierova transformace* (FFT), který má složitost jen $O(Nlog_2N)$. Více informací o FFT naleznete ve skriptech [3].

4 Genetické algoritmy (GA)

Informace o genetických algoritmech jsem čerpal z [1] a [7]. Genetické algoritmy jsou adaptivní metody, které se používají pro vyhledávání a optimalizaci řešení problémů. Jsou založeny na genetických procesech živých organismů a principech přirozeného výběru a přežití silnějších.

Genetické algoritmy simulují život *populace řešení*, která evolučním cyklem přechází z generace do generace a pomalu sílí. Obdobou boje o přežití jsou v GA selekce na konci každého evolučního cyklu, při kterých jsou špatná řešení (slabí jedinci) odstraněni a jejich místa v populaci jsou zaplněna novými řešeními. Generací bývá zpravidla několik desítek až set a z té poslední se vybírá nejlépe „přizpůsobený jedinec“, kterého považujeme za řešení problému. Řešení nebývá zcela optimální, ale většinou je přijatelné.

GA obsahuje i obdobu soupeření o možnost zplození potomků. Přitom platí, že ta nejlepší řešení mají relativně velký počet potomků, zatímco ta horší mají potomků málo, nebo vůbec žádné. Tento fakt vede k tomu, že geny dobrých možností řešení se šíří populací a zvyšují svůj počet v každé následující generaci.



Obrázek 4.1: Evoluční cyklus

Přirozeně pak křížením dvou dobrých rodičovských chromozomů může

vzniknout „vynikající“ potomek, jenž svou silou převyšuje oba své rodiče. To je důvod, proč GA tak dobře fungují. Dobré geny se neztratí, ale přežijí v kombinaci s dobrými geny od jiných jedinců. Jedinci populace jsou ohodnoceni takzvanou „silou jedince“ (fitness score), což je číselná hodnota, která ohodnocuje kvalitu řešení problému (míru přizpůsobení).

4.1 Základní princip

Abychom mohli problém úspěšně řešit genetickým algoritmem, musíme zvolit vhodné kódování a reprezentaci řešení. Pracujeme také s funkcí síly (fitness function), která nám ohodnotí každé řešení. V každém cyklu algoritmu provádíme výběr rodičů pro reprodukci, abychom získali nové potomky a přírodní výběr do další populace. Následuje tradiční algoritmus podle [1].

```
BEGIN /* genetický algoritmus */
  generování počáteční populace
  výpočet síly všech jedinců

  WHILE NOT hotovo DO
  BEGIN /* evoluční cyklus - tvorba další generace */
    FOR velikost_populace / 2 DO
    BEGIN /* reprodukce */
      výběr dvou jedinců staré generace
        /* preferují se ty silnější */
      rekombinace dvou rodičů nám dává dva potomky
      výpočet síly potomků
      vložení potomků do populace
    END

    výběr jedinců do další generace

    IF ukončující_podmínka_splněna THEN
      hotovo := TRUE

  END
END
```

4.1.1 Kódování chromozomu a reprezentace řešení

Reprezentací jednoho možného řešení úlohy bývá nejčastěji množina parametrů. Parametry řešení považujeme za *geny*, které spojené do řetězce tvoří *chromozom*. Gen je základním nositelem genetické informace.

V genetice je soubor všech genetických informací označován jako *genotyp*. Genotyp obsahuje informace potřebné pro sestavení organismu. Skutečným projevem genotypu po sestavení je *fenotyp*. Fenotyp je soubor všech pozorovatelných vlastností organismu. Síla jedince pak úzce souvisí s vlastnostmi fenotypu.

S porušením (mutací) kódu jednoho genu se přenášená informace pravděpodobně výrazně změní. Dopad této změny na informaci přenášenou celým chromozomem a tedy i na fenotyp bude mít zejména v závislosti na funkci tohoto genu. Například změna parametru posunutí (na ose x) u periodické funkce, nebude mít zdaleka takový dopad jako posun funkce neperiodické.

4.1.2 Fitness funkce

Funkce síly (Fitness function) musí být definována pro všechny relevantní genotypy. Ve výsledku nám ohodnotí zvolený chromozom reálným číslem hodnotou síly (fitness), která nám poslouží při porovnávání kvality různých řešení.

Kdyby úlohou GA bylo najít globální maximum nějaké funkce, fitness bychom získali prostým dosazením hodnot genů za vstupní parametry funkce. Pro takovou úlohu ale existují mnohem spolehlivější a jednodušší metody, proto pro komplexnější úlohy musíme často zvolit vhodné otestování výsledného fenotypu. Pokud to jde, vyzkoušíme ho přímo v praxi, nebo co nejpřesvědčivější simulaci. Přitom jednoduše platí, že výstupní hodnota fitness funkce je vyjádření vlastnosti (nebo vlastností) které budeme maximalizovat.

4.1.3 Přírodní výběr

Přírodní výběr je základním nástrojem GA pro vytvoření tlaku na populaci. Přísnější výběr nechává přežít jen ty opravdu nejsilnější, na kterých je, aby svými potomky zaplnily místo v populaci, které museli slabší jedinci uvolnit.

To má pochopitelně vliv na poměr prohledávání prostoru a zlepšování řešení. Oba tyto aspekty jsou pro GA důležité a při volbě tlaku na populaci bychom měli mít na paměti, že jejich nevyváženost povede k podpoře neduhů GA, jako jsou předčasná konvergence a pomalý závěr (oddíl 4.2).

Rozlišujeme dva základní druhy přírodních výběrů – před a po aplikaci genetických operátorů. Některé konkrétní způsoby výběru se používají v obou případech, některé jsou specifické pro jeden z nich.

První druh (před genetickými operátory) je určen k vybrání jedinců do takzvaného *rozmnožovacího jezírka* a ten představuje vliv konkurence mezi členy populace. Z něho se vybírají jedinci pro *křížení*, dokud není jezírko prázdné. Obvykle se do jezírka vytvářejí kopie jedinců, jejichž počet nějak závisí na jejich síle.

Otázkou ale je podle jakého pravidla. Má mít čtyřikrát silnější jedinec čtyřikrát více potomků nebo má počet příležitostí k rozmnožení záviset pouze na pořadí? V každém případě se ukázalo, že značná převaha silnějších vede k předčasné konvergenci.

Druhou podskupinou jsou selektory aplikované na „přemnoženou“ populaci po sloučení původní generace rodičů s množinou potomků. Tyto selektory představují vliv prostředí. Selekcí by se měla velikost populace vrátit na stanovenou velikost.

Výběr nejlepších

Nejjednodušší způsob je asi selekce nejlepších. Jedinci se seřadí podle síly a určité procento (nebo přesně tolik, kolik přebývá) se z populace vyřadí. Je to možná až příliš striktní selekce, která ignoruje vliv náhody na přírodní výběr.

Výběr ruletou

Dalším způsobem je selekce ruletou. Po seřazení se fitness hodnoty všech jedinců přetransformují tak, aby jejich součet dával jedna. Náhodně vygenerované číslo od nuly do jedné pak určí vybraného jedince do další generace.

Výběr soutěží

Třetí způsob, výběr soutěží, vybírá zcela náhodně několik jedinců do soutěže. Jejich počet bývá dva, nebo o něco více. Nejlepší z nich postupuje do další generace. Volbou větších turnajů zvyšujeme tlak na přizpůsobení jedinců a tím klademe důraz zlepšování na úkor prohledávání prostoru. Můžeme ale také vítěze turnaje vybrat jen s určitou pravděpodobností (typicky kolem 0,9) a tak naopak dát šanci slabším jedincům.

Vzhledem k tomu, že poslední zmiňovaný nevyžaduje pro svou funkci řazení, bude patřit k těm rychlejším implementacím přírodního výběru.

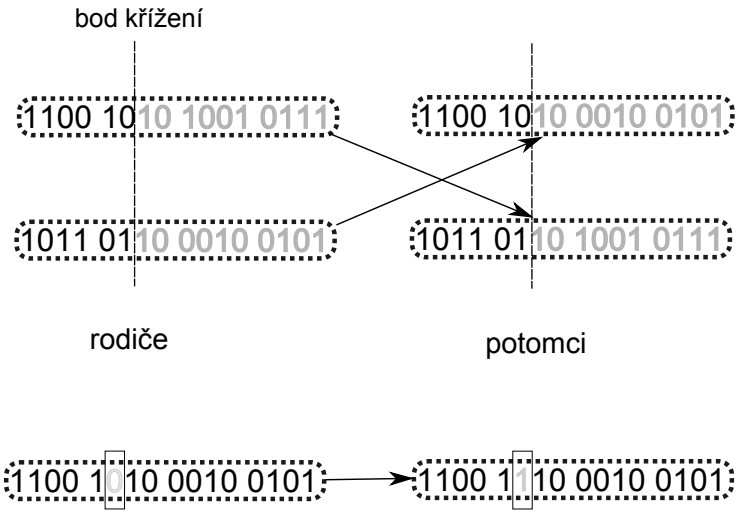
4.1.4 Genetické operátory

Během fáze reprodukce jsou vybírány dvojice a jejich chromozomy jsou zkombinovány za účelem získání potomků, kteří budou mít možnost stát se součástí další generace. Rodiče jsou vybráni náhodně, ale většinou existuje určité zvýhodnění těch s vyšší silou. Dva chromozomy vybraných jedinců poté kříží me. Nejtypičtější konfigurace reprodukce je dvojice genetických operátorů crossover a mutace.

Křížení (crossover)

Křížení je většinou hlavní genetický operátor. Při běžném křížení se náhodně zvolí takzvaný bod křížení. Ten udává začátek sekvence v chromozomu, kterou si chromozomy vymění. Tím vzniknou nové chromozomy dvou potomků a každý z nich má části genetické informace obou rodičů. Tomuto způsobu se říká v originále single-point crossover. Většinou se aplikuje na všechny dvojice, ale lze ho použít i s určitou pravděpodobností. Tak může docházet k tomu, že se rodiče jen replikují bez porušení jejich genotypů a ke zkřížení vlastně nedojde.

Bodů křížení můžeme použít i více. Druhý bod pak označuje místo konce vyměňované sekvence. Třetí označuje začátek další výměny a tak to stále pokračuje.



Obrázek 4.2: Grafické znázornění křížení (nahoře) a mutace (dole) binárního genu

Mutace

Mutace se provádí obvykle ihned po křížení, ale jen u malého procenta jedinců (typicky 0,001). Je jen doplňkovým mechanismem prohledávání prostoru. Mutace jen zaručuje, že žádná část prostoru nemá nulovou šanci na prohledání a částečně brání přílišné unifikaci jedinců populace.

4.2 Neduhy genetických algoritmů

Na začátku každého GA pracujeme s náhodně vygenerovanou počáteční populací. I jejich fitness hodnoty jsou více méně náhodné a jejich rozsah poměrně veliký. Po několika generacích začne populace konvergovat (ze začátku pravděpodobně k několika vhodným místům v prostoru). Jak populace konverguje, rozdíl mezi maximální a průměrnou fitness hodnotou se zmenšuje a to je znakem dvou častých nedokonalostí GA.

4.2.1 Předčasná konvergence

Předčasná konvergence je poměrně běžný problém, se kterým je třeba při návrhu počítat. V populaci vznikne skupina jedinců poměrně silnějších než většina ostatních, která přirozeně dostane více příležitostí k rozmnožení a začne v populaci jednoznačně dominovat. To zní jako normální princip GA, ale tento proces nesmí nastávat příliš rychle.

Dominantní skupina totiž může ovládnout populaci a tak ji stáhnou do lokálního maxima, ze kterého se těžko dostává ven. Tento jev není nepodobný tomu, který nastává u horolezeckého algoritmu. Má-li se populace dostat z lokálního maxima, musí zapůsobit mutace, protože křížení předčasnou konvergenci způsobuje. Pokud předčasná konvergence nastává často, GA degraduje náhodné prohledávání.

Několik zajímavých návrhů jak předčasné konvergenci předcházet naleznete ve článku [1] od strany 10.

4.2.2 Pomalé dokončování

Pomalé dokončování je do jisté míry opačným jevem k předčasné konvergenci. V případě, že GA je těsně před nalezením globálního maxima, musí mutace pravděpodobně generovat pouze horší řešení, která nejsou k ničemu. Algoritmu pak trvá dlouho nalézt to opravdu nejlepší řešení, přesto že ho má na dosah.

5 Genetický matching pursuit (GMPA)

Základní princip GMPA je shodný s algoritmem matching pursuit. GMPA, stejně jako MP, v každé iteraci provádí výběr atomu ze slovníku funkcí. Vybrané atomy si s jejich vahami ukládá a odečítá od vstupního signálu, dokud neprovede určený počet iterací.

Jediný rozdíl je v tom, jaký algoritmus GMPA využívá k výběru nejvhodnějšího atomu ze slovníku. K tomuto účelu používá genetický algoritmus, který pracuje s *populací chromozomů*, kde každý chromozom reprezentuje jeden atom. My budeme používat slovník Gaborových atomů. Populace GA se na začátku naplní náhodně vygenerovanými chromozomy, několika prvotními nástřely aproximace vstupního signálu Gaborovým atomem. Atomy (resp. jejich chromozomy) jsou ohodnoceny podle míry shody se signálem a následně populace projde několika *evolučními cykly*. Evoluční cyklus se skládá ze tří základních procesů (výběr, reprodukce a nahrazení) a jimi populaci vyvíjí. Když přejde populace do poslední generace, vybere se z ní nejlépe ohodnocený chromozom, který považujeme za řešení úlohy Gaboruv atom, který bude vybrán ze slovníku.

Nyní se budu věnovat částem GA, které musí korespondovat s řešenou úlohou. Jedná se o reprezentaci chromozomu a fitness funkci. Ukážeme si, jak by se měly řešit v kontextu GMPA, protože na nich závisí, jestli bude GA řešit úlohu výběru atomu správně a efektivně.

5.1 Reprezentace chromozomu

Jak bylo řečeno, populace je tvořena chromozomy, které reprezentují Gaborovy atomy. Gaboruv atom je určen čtyřmi parametry. Tedy i chromozom atomu bude tvořen čtyřmi geny, z nichž každý bude obsahovat číselnou hodnotu pro určení měřítka, pozice, frekvence nebo fáze atomu. Takto vypadá chromozom pro reprezentaci Gaborova atomu:

$$\theta = (\text{měřítko}|\text{pozice}|\text{frekvence}|\text{fáze})$$

V následující části se budeme zabývat tím, z jakých typů genů se dá takový chromozom složit. Vhodná reprezentace chromozomu zaručuje dostatečnou přesnost aproximace a také rychlost řešení dané úlohy.

5.1.1 Reprezentace genů

Parametry atomu mají přirozeně vymezené rozsahy hodnot, které závisí na délce vstupního signálu, takže od reprezentace genů budeme vyžadovat možnost snadného nastavení rozsahu. Dalším kritériem je velikost prostoru, kterou může reprezentace genů ovlivnit. Parametry atomu jsou reálné hodnoty a my máme tři možnosti reprezentace genů: gen reprezentovaný proměnnou typu double binární gen gen reprezentovaný proměnnou typu integer

Gen reprezentovaný proměnnou typu double

Takovýto gen by mohl jako jediný přímo nést hodnoty parametrů a to s obrovskou přesností. Dají se mu také snadno nastavit omezující hranice. Nevýhodou takových genů ale je, že vytváří obrovský prohledávaný prostor. Naším cílem je, mít prostor co nejmenší, aby bylo jeho prohledání snadné a rychlé.

Binární gen

Binární gen je nejtradičnější reprezentací genu a na rozdíl od obou zbývajících typů umožňuje provádění operace křížení, která nejvíce připomíná křížení v přírodě. Bod křížení může padnout i doprostřed genu. To má výhodu i nevýhodu zároveň. Zasažený gen je „roztržen“ na dvě části a tak vlastně dojde k porušení informace, kterou gen nesl. Výhodou je, že výsledek v narušeném genu je „tak napůl“ náhodné číslo, což samozřejmě přispívá k prohledávání prostoru. Binární gen by ale byl pro GMPA použitelný jen s omezeními. U binárního genu se totiž nedají snadno nastavit omezující hranice. Ze spodu je vždy omezen nulou a shora se dá nastavit maximální hodnota jen na $2^b - 1$, kde můžeme volit toliko b – počet využitých bitů.

Použitelnost binárního genu také předurčuje jeho vhodná implementace. Část knihoven pro implementaci GA binární gen pojímají jako řetězec 32 intergerů, což musí být extrémně neúsporné jak datově, tak časově.

Gen reprezentovaný proměnnou typu integer

Gen reprezentovaný proměnnou typu integer také není zcela ideální, ale převažují u něj dobré vlastnosti. Možnost nastavení rozsahu genu omezena pouze rozsahem integeru. Nevýhodou je, že stejně jako gen reprezentovaný doublem umožňuje pouze omezené křížení. Tím myslím, že bod křížení může být umístěn jen mezi geny. Schopnost prohledávání prostoru je tak operátoru křížení omezena. Tento nedostatek se dá ale vyvážit zvýšením četnosti mutace.

Pro reprezentaci parametrů Gaborova atomu bych tedy preferoval gen reprezentovaný proměnnou typu integer. Ještě by se dal použít binární gen ve správné implementaci, tj. jedním integerem. Binární gen by nám ale znepříjemňoval nastavování rozsahu genu. Řešením pro tento problém by mohla být podmínka, že délka vstupního signálu musí být vždy mocnina dvou.

5.1.2 Kódování genů a velikost prostoru

Kódováním genu je myšleno, jakým způsobem přepočteme číslo uložené (zakódované) v genu na parametr, který má představovat. Rozhodl jsem se použít pro použití genu reprezentovaného proměnnou typu integer. Je také vhodné, předtím než budeme pokračovat, podívat se znovu na vzorec Gaborova atomu (vztah (2.8)).

Frekvence a fáze

Tyto dva parametry ovlivňují vlastnosti kosinové modulace. Ale parametr, který se obecně v kontextu MP nazývá frekvence, ve skutečnosti frekvencí není. Jedná se o normovanou úhlovou rychlost (ω_N).

$$f = \frac{\omega_N f_{vz}}{2\pi} \ ,$$

kde f_{vz} je vzorkovací frekvence. Protože na následujících řádcích budu nucen používat i pojem frekvence f, je vhodné vrátit se na okamžik k obecně uznávaným názvům jednotek a předejít tak nedorozumění.

Mějme vstupní signál $x = (x_0, x_1, \ldots x_{N-1})$, který má N vzorků. Nejmenší možnou frekvencí f_{min} ve vstupním signálu je frekvence, jejíž počet vzorků na periodu je N. Tak můžeme zjistit i minimální ω_N ($\omega_{N,min}$), což je změna

úhlu na jeden vzorek signálu pro f_{min}. Dá se spočítat jako $\omega_{N,min} = 2\pi/N$ a je konstantní pro danou délku signálu. Minimální úhlovou změnou a číslem harmonické frekvence $k \in \mathbb{N}$ pak můžeme určit libovolnou ω_N podle vztahu:

$$\omega_N(k) = k \cdot \omega_{N,min}$$

Normovaná úhlová rychlost (pro danou harmonickou k) je změna úhlu na vzorek. Když je ω_N vysoká, je perioda signálu krátká. Pro nízkou ω_N je delší.

Číslo k mi tedy jednoznačně určuje ω_N a tím i frekvenci. Je to proto vhodná veličina k umístění do jednoho z genů chromozomu. Zbývá určit horní hranici genu pro danou délku signálu. Nejvyšší pozorovatelná frekvence v signálu má dva vzorky na periodu, tj. ω_N rovnou π. Hodnota k pro takovou frekvenci je rovna $N/2$. Tak jsme definovali rozsah a kódování genu. Ponese číslo harmonické, které může nabývat hodnot od 1 do $N/2$.

Fáze (φ) může nabývat hodnot od 0 do 2π, proto využijeme minimální úhlové změny a podobného vzorce jako pro normovanou úhlovou rychlost:

$$\varphi(l) = l \cdot \omega_{N,min} \quad ,$$

kde $l = 0, 1, \ldots N$. Docílili jsme tak potřebného rozsahu fáze a vytvořili proměnou l, která bude kódována dalším genem.

Měřítko a posunutí

Měřítko (s) a posunutí (p) jsou parametry, které ovlivňují Gaussovu funkci. Čím je měřítko větší, tím je atom „roztaženější“ do stran. Zvětšující se posunutí pohybuje atomem doprava. Maximální hodnota obou těchto vlastností je obvykle N. Spodní hranice posunutí je nula. Nulové posunutí znamená, že je střed atomu úplně na začátku. Pro měřítko se volí minimální hodnota 2. Atom s tak malým měřítkem už je opravdu úzký.

Měřítko a posunutí jsou sice ve skutečnosti reálné hodnoty, přesnost na celá čísla ale úplně stačí. Můžeme dokonce zavést krok těchto parametrů a tím zmenšit prostor na úkor přesnosti.

$$s(m) = m \cdot \Delta s$$

$$p(n) = n \cdot \Delta p$$

Když nastavíme kroky obou například na 4, prohledávaný prostor se zmenší 16krát a to bez znatelného dopadu na přesnost aproximace. Krok větší než

10 už by se asi na kvalitě mohl podepsat. Proměnné m a n budou tedy kódovány zbylými geny, ale jejich rozsahy se musí změnit podle velikosti kroků. Proměnná m začíná od jedné, aby nejnižší možné měřítko bylo co nejblíže 2 a vždy větší než nula.

$$m = 1, 2, \ldots \frac{N}{\Delta s}$$

$$n = 0, 1, 2 \ldots \frac{N}{\Delta p}$$

Pokud N je číslo nedělitelné Δs či Δp beze zbytku, zaokrouhlujeme dolů.

5.1.3 Pořadí genů v chromozomu

Pořadí genů v chromozomu nemá vliv na rekonstruovaný atom, může ale ovlivnit efektivitu vývoje jedinců. Platí, že geny, které mají jistou souvislost, by měly být umístěny blízko sebe, aby dobré kombinace genů měly větší šanci přejít do další generace pohromadě.

Chromozom Gaborova atomu je tvořen dvěma logickými dvojicemi genů. Dvojice měřítko a posunutí nastavují Gaussovu funkci a frekvence s fází určují průběh kosinové funkce. Žádné další logické vztahy nejsou zřejmé, a proto chromozom reprezentující atom může vypadat takto:

$$\theta = (m, n, k, l)$$

5.2 Fitness funkce

Fitness funkce je nástroj GA, který ohodnocuje chromozom hodnotou síly (fitness). V kontextu GMPA je posuzovanou vlastností míra podobnosti Gaborova atomu se vstupním signálem. Z iterační formule MP (vzorec (2.6)) víme, že úkolem GA bude vyřešit následující maximalizační problém:

$$\max_{\gamma} \langle x | g_\gamma \rangle \ ,$$

kde x je vstupní signál a g_γ je normovaný Gaborův atom, který je určen vektorem parametrů γ. Ten zase závisí na chromozomu θ.

Fitness funkce bude založená na skalárním součinu. Jako skalární součin ale může vycházet i záporné číslo. Co to znamená, když algoritmus najde atom, jehož skalární součin je zcela minimální? To znamená, že algoritmus nalezl řešení, ale s přesně opačnou fází. Ve vztahu (2.1), je proměnná modulus (a_n) reálné číslo. Proto princip MP nezakazuje použití přesně opačných řešení, pokud bude vynásoben správným (záporným) modulem.

Takže ohodnocovat řešení genetického algoritmu můžeme pomocí skalárního součinu v absolutní hodnotě:

$$\max_{\gamma} |\langle x|g_\gamma\rangle|$$

Velice špatná řešení tedy od teď považujeme naopak za velice dobrá. Kromě toho křížení „záporných“ řešení s „kladnými“ genetickému algoritmu nijak neuškodí a to nás vede ještě k malému bonusu.

Jsou-li řešení se záporným a kladným skalárním součinem ekvivalentní, jeden z poloprostorů daných hodnotou fáze ($0 - \pi$ a $\pi - 2\pi$) nepotřebujeme. Dále už můžeme používat rozsah fáze jen od 0 do π, a tím se nám prohledávaný prostor ještě 2krát zmenší.

6 Využití Fourierovy transformace pro urychlení algoritmu matching pursuit

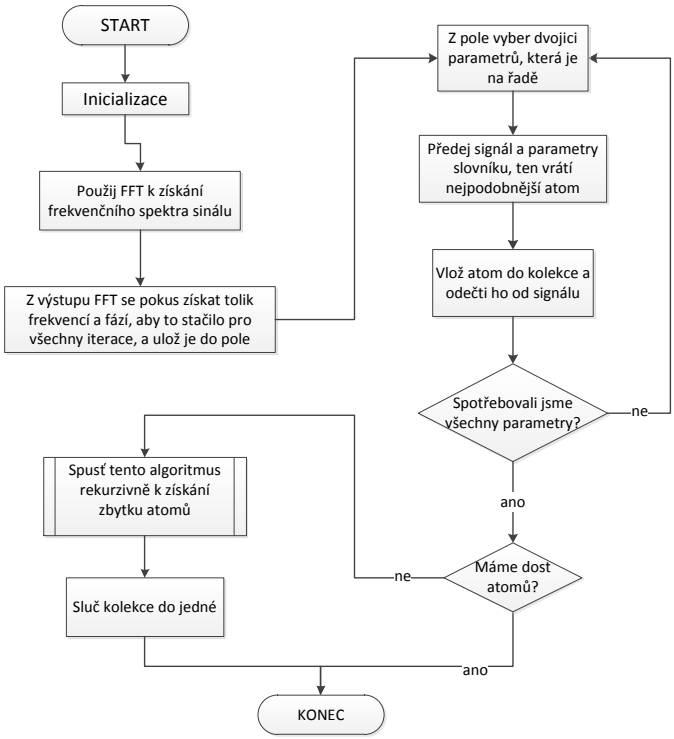
Velikost prohledávaného prostoru (a tím i rychlost prohledání) udávají parametry Gaborových atomů. Cílem je tedy vždy prostor co nejvíce zmenšit. Kdybychom měli metodu, která dokáže ze signálu rychle a přesně vypočítat některé parametry, předali bychom je GA jako konstanty a ten už by jen „doladil“ ty zbývající. Prostor by se tak radikálně zmenšil.

Vhodná metoda existuje, jmenuje se diskrétní Fourierova transformace (viz kapitolu 3). Já ale použiji její rychlejší variantu FFT. Fourierova transformace převádí signál z časové do frekvenční oblasti. To znamená, že v jejím výstupu se dají nalézt frekvence s vysokou amplitudou a k frekvencím také získat příslušné fáze. To jsou dva parametry ze čtyř, které bych získal rychlou deterministickou metodou a při prohledávání slovníku genetickým algoritmem bych je považoval za konstantní. GA by měl na jejich základě rycleji najít atom, který by velmi dobře aproximoval vstupní signál.

Na mě bylo zjistit, jestli takový algoritmus může fungovat, jestli je frekvence s vysokou amplitudou vhodná pro formování nejlepšího atomu a jestli to vůbec vede k urychlení algoritmu. Podařilo se mi vymyslet dva podobné algoritmy založené na tomto principu:

- GMPA s obecnou FFT akcelerací
- GMPA s FFT akcelerací „po jednom“.

Pro jejich fungování potřebujeme algoritmus, který umí z výstupu FFT získat určitý počet dominantních frekvencí (frekvencí s vysokou amplitudou). Tomuto algoritmu se věnuje oddíl 6.2.



Obrázek 6.1: Vývojový diagram genetického algoritmu matching pursuit s obecnou FFT akcelerací

6.1 Způsoby vylepšení

6.1.1 Obecná FFT akcelerace

Hlavní část tohoto vylepšení probíhá těsně před iteračním cyklem MP. Nejprve se spustí FFT, která přetransformuje vstupní signál na frekvenční spektrum. Dalším úkolem je ve spektru nalézt dominantní frekvence. MP potřebuje tolik frekvencí (a k nim příslušných fází), kolik má vytvořit atomů.

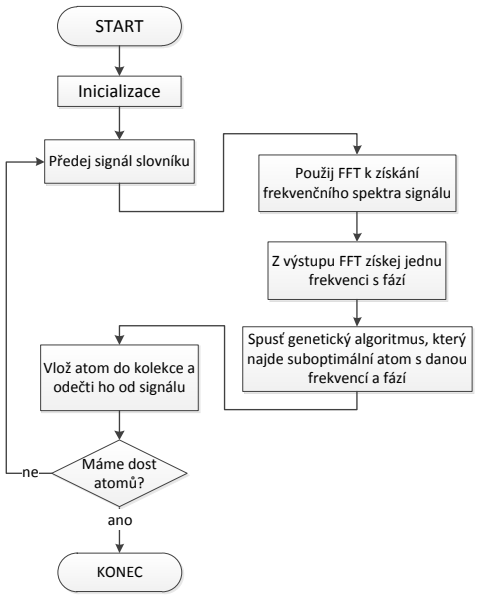
Když získá pole frekvencí a fází, začne je postupně předkládat algoritmu na prohledání prostoru, který těmto dvěma parametrům vždy dohledá měřítko a posunutí. Tím má značně ulehčenou práci a mělo by dojít k výraznému urychlení a dokonce i zpřesnění výsledků.

Co se stane v případě, že tento algoritmus ve výstupu FFT nenalezne dostatečný počet dominantních frekvencí? V tom případě použije ty, co má, a spustí tentýž algoritmus rekurzivně s residuem na vstupu pro vytvoření zbývajících atomů. Po jeho skončení výsledky sloučí.

Předpokládá se, že všechny frekvence jsou v signálu od začátku obsaženy. To je samozřejmě pravda, ale nedostatečná přesnost frekvence způsobuje, že při aproximaci určité části signálu atomem vzniká v signálu atom trochu jiné frekvence, na který už algoritmus není schopen zareagovat. Tento jev ukazuje obrázek A.2 v přílohách. A to je zdrojem chyb, které znatelně kazí celkovou aproximaci.

6.1.2 FFT akcelerace „po jednom“

V reakci na chyby GMPA s obecnou FFT akcelerací jsem navrhl ještě další metodu. Ta se od předchozí liší zejména tím, že výpočet frekvenčního spektra probíhá pro každý atom zvlášť.



Obrázek 6.2: Vývojový diagram genetického algoritmu matching pursuit s FFT akcelerací „po jednom“

MP běžným způsobem spustí iterační cyklus a přejde k hledání atomu ze slovníku. Před spuštěním GA se použije FFT k vytvoření diskrétního frekvenčního spektra a v něm se vyhledá pouze jedna nejdominantnější frekvence. Dopočítá se fáze a tyto dva parametry se použijí pro snadnější nalezení atomu genetickým algoritmem.

V další iteraci se vše opakuje. Hledáme-li k atomů, FFT a analýza frekvenčního spektra poběží k-krát, ale to není velká časová zátěž. Tento algoritmus vykazuje také dobré výsledky a k tomu ještě řeší problém svého předchůdce.

6.2 Extrakce dominantních frekvencí

Výstup FFT je pole komplexních čísel o délce N (N je také délka vstupního signálu), ale stačí prozkoumat jen první polovinu, protože výstup se od poloviny opakuje zrcadlově obrácený. Z každého prvku pole můžeme získat všechny potřebné informace. Když máme $n = 1, 2, \ldots N/2$, pak pro komplexní číslo $c_n = (r_n, i_n)$ platí:

$$f_n = f(n) = n \cdot \frac{2\pi}{N}$$

$$\varphi_n = \varphi(c_n) = \begin{cases} NaN & \text{pokud } r_n = 0 \wedge i_n = 0 \\ \pi/2 & \text{pokud } r_n = 0 \wedge i_n > 0 \\ -\pi/2 & \text{pokud } r_n = 0 \wedge i_n < 0 \\ arctg\frac{i_n}{r_n} & \text{pokud } r_n > 0 \\ \pi + arctg\frac{i_n}{r_n} & \text{pokud } r_n < 0 \end{cases}$$

$$A_n = A(c_n) = \sqrt{r_n^2 + i_n^2}$$

Amplitudy, absolutní hodnoty komplexních čísel, tvoří průběh od A_1 do $A_{N/2}$ (na indexu nula je tzv. stejnosměrná složka). První část algoritmu zjistí indexy lokálních maxim v tomto průběhu. Ukazuje to obrázek 6.3 na straně 27 a následující pseudokód.

```
BEGIN /* najdi lokání maxima mezi amplitudami */
   INPUT c /* pole komplexních čísel, výstup FFT */
   count := 0

   current := c[1].abs()
   next := c[2].abs()

   IF current > next THEN
   BEGIN
      maxs[count] := 1
      count := count + 1
   END

   i := 2
   FOR i < N/2 DO
   BEGIN
      previous = current
      current = next
      next = c[i+1].abs()

      IF previous < current AND current > next THEN
      BEGIN
         maxs[count] := i
         count := count + 1
      END
      i := i + 1
   END
   OUTPUT maxs o délce count
END
```

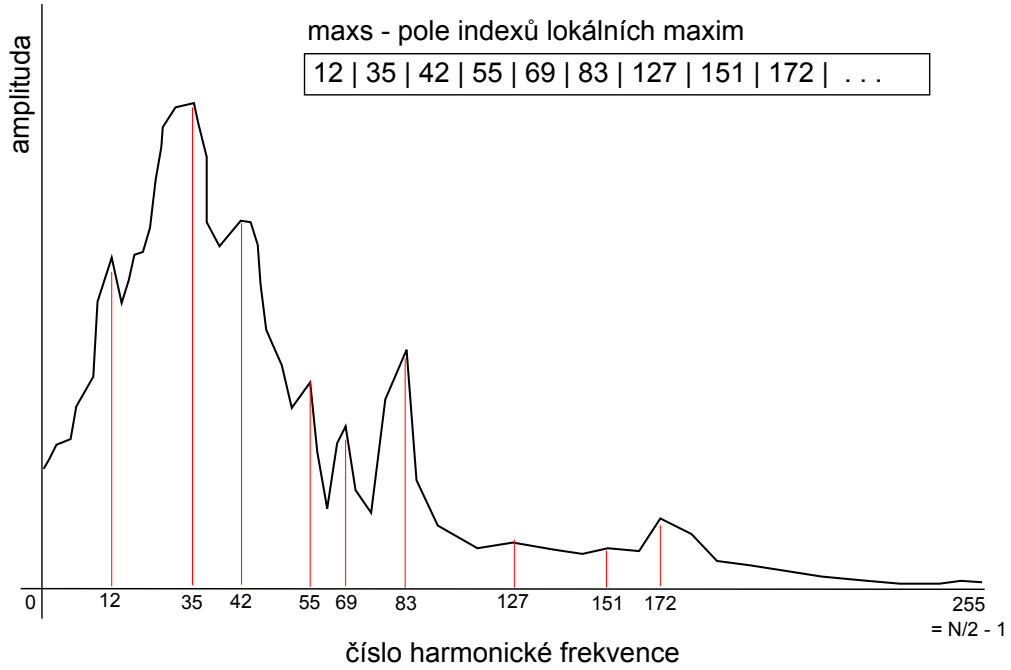
Získal jsem pole indexů lokálních maxim v průběhu amplitud. Nyní potřebuji nalézt určitý počet těch největších amplitud. K tomu mohu použít modifikaci algoritmu Selection sort[1]. Nechám si seřadit pouze k indexů podle velikosti amplitudy. Za k se může dosadit třeba počet atomů, které má MP vytvořit. Průběžně se pomocí vybraných indexů vytváří pole frekvencí a fází.

[1]Tento způsob řazení v každé iteraci vybere nejlepší prvek z aktuálního rozsahu a vymění ho s prvním prvkem rozsahu. Rozsah na začátku zahrnuje celé pole a po každé iteraci se zkracuje, jak se z leva zvětšuje seřazená část, až je pole seřazené celé.

```
BEGIN /* najdi k největších */
   INPUT k /* požadovaný počet frekvencí */
   j := 0

   IF k > maxs_length
   THEN
      count := maxs_length
   ELSE
      count := k

   FOR count DO
   BEGIN
      index := najdi v poli maxs (v rozmezí od j
         do konce) index, který označuje
         nejdominantnější frekvenci
      vyměň ho s j-tým v poli maxs
      /* aby byl příště mimo rozmezí */
      j := j + 1

      /* frekvenci ulož */
      dominants <= index * (2PI / N)
      /* ulož také fázi*/
      dominants <= vypočti fázi z c[index]
   END

   v poli dominants máme count frekvencí a fází
   /* většinou je count = k */
END
```

6.3 Nevýhoda FFT akcelerací

FFT akcelerace jsou velmi prostá vylepšení a z toho vyplývá nevýhoda, která se může občas projevit. Tyto metody analyzují signál pomocí FFT poměrně málokrát a vždy po celé délce. Algoritmy příbuzné Fourierově transformaci zjišťují, jakou amplitudu a fázi mají jednotlivé harmonické frekvence, ale ne v jakém místě. Výsledné amplitudy a fáze jsou vlastně průměrované.



Obrázek 6.3: Vyznačená lokální maxima v průběhu amplitud harmonických frekvencí; Jejich indexy (čísla harmonické) se uloží do pole.

Mějme situaci, kdy se ve vstupním signálu na různých místech nacházejí dva atomy stejné frekvence a rozdílné fáze. Transformace pravděpodobně odhalí výskyt frekvence, ale fázi určí chybně jako průměr obou. GA pak obdrží nepřesný parametr, a kvůli tomu se mu nepodaří spolehlivě aproximovat ani jeden z atomů vstupního signálu.

Tento případ je sice na naměřených datech dost nepravděpodobný, ale je nutno přiznat, že to riziko existuje.

7 Implementace GMPA

Při implementaci genetického matching pursuit jsem se z velké části opíral o informace z diplomové práce ing. Tomáše Řondíka [8], webových stránek JGAP (`http://jgap.sourceforge.net/`) a semestrální práce Petra Kellnhofera [4].

7.1 Knihovny použité k implementaci

K implementaci byly použity dvě knihovny, které my výrazně usnadnily návrh tříd a struktury. Většinu tříd jsem mohl oddědit a převzít zavedené principy.

7.1.1 EEGDSP

Následující odstavec je převzat ze stránky (`http://www.kiv.zcu.cz/en/research/downloads/product-detail-en.html?produkt_id=101`) a mírně upraven.

EEGDSP je knihovna pro zpracování signálů zaměřená na EEG/ERP signály. Poskytuje jednotné rozhraní pro práci s rychlou Fourierovou transformací, waveletovou transformací (diskrétní i spojitou variantou) a matching pursuit algoritmem, které obsahuje. Výstupy jednotlivých metod jsou ukládány do autory navržených kolekcí, které umožňují jejich efektivní zpracování.

Autorem většiny kódu je Ing. Tomáš Řondík. Kódy přepracoval do podoby univerzální snadno použitelné a rozšiřitelné knihovny Petr Kellnhofer, který také navrhl strukturu a rozhraní metod zpracování signálu. V současnosti je EEGDSP jedním z projektů Katedry informatiky a výpočetní techniky na Fakultě aplikovaných věd ZČU v Plzni, který je stále vyvíjen studenty a pracovníky KIV.

Pro mou práci jsem využil především zkušeností autorů se zpracováním signálu pomocí metody matching pursuit, kterou jsem rozšiřoval a modifikoval. Využil jsem také rychlou Fourierovu transformaci jako součást vylepšení

GMPA.

UML diagram MP algoritmu, části EEGDSP, naleznete v příloze B (Obrázek B.1).

7.1.2 JGAP

JGAP je nástroj na vytváření genetických algoritmů a genetického programování. Poskytuje nástroje k řešení úloh pomocí evolučních a genetických principů. Jeho základní užívání je v zásadě velmi jednoduché. Stačí splnit několik jednoduchých kroků:

1. Vytvořit instanci třídy `DefaultConfiguration`
2. Vytvořit vzorový chromozom (instanci `Chromosome`) a vložit do ní vzorové geny (instance `Double/Interger/BinaryGen` - existují i další typy a chromozom může obsahovat více typů najednou)
3. Implementovat vlastní fitness funkci (dědit od třídy `FitnessFunction`)
4. Nastavit velikost populace a konfiguraci předat instance fitness funkce a vzorového chromozomu konfiguraci
5. Založit novou populaci (typu `Genotype`) a spustit evoluci na určený počet generací

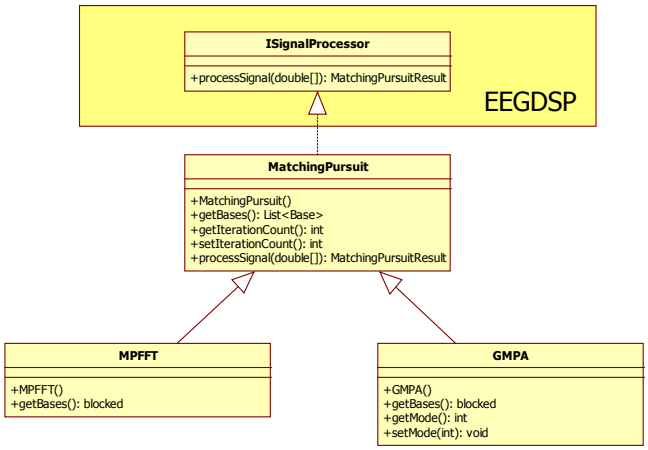
A předpřipravený genetický algoritmus řeší vaši úlohu.

Pokročilí uživatelé ocení, že je JGAP modulární. Je tedy možné složit si vlastní konfiguraci GA jako ze stavebnice. Pro ještě pokročilejší uživatele, kteří nejsou spokojeni s nabídkou modulů, je v mnoha případech snadné knihovnu modifikovat a doplnit o vlastní části. Popravdě, některé části se ale svému přepsání sveřepě brání. Zejména tím, že mají pro nezasvěceného uživatele příliš složitou metodu `clone()`, bez které nemůže nová implementace správně fungovat.

Nicméně, knihovna obsahuje mnoho příkladů, ze kterých se začátečník naučí JGAP používat a může se inspirovat pro vlastní projekty. UML diagram základních tříd knihovny naleznete v příloze B (Obrázek B.2).

7.2 Realizace

Metodu GMPA jsem naprogramoval v jazyce Java SE (verze 1.7) za pomoci vývojového prostředí NetBeans. Implementace algoritmu je součástí projektu ERP-Classifier.



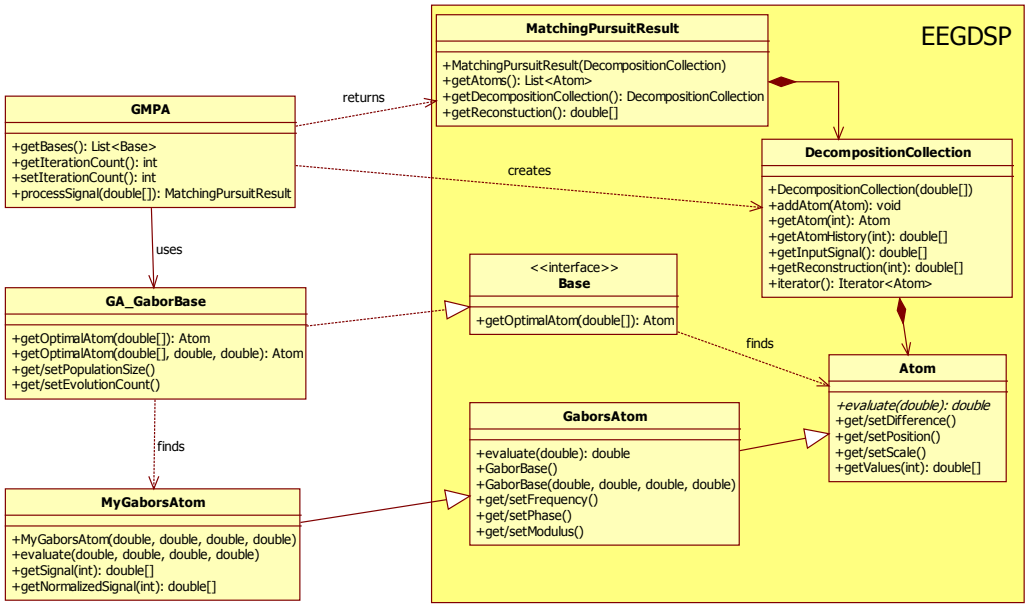
Obrázek 7.1: Nové schéma algoritmů matching pursuit

7.2.1 Nové schéma MP

Jako první jsem si musel přizpůsobit hlavní třídu `MatchingPursuit`, protože v knihovně EEGDSP je tato třída implementována podle návrhového vzoru *singleton*. Což se vylučuje s použitím metody ve více vláknech a také znemožňuje od této třídy dědit. Singleton totiž nemá veřejný konstruktor. Zkopíroval jsem si tedy celou třídu do svého projektu a vše upravil.

Z UML diagramu na obrázku B.1 je vidět, že třída `MPFFT` a obalující třída `MatchingPursuit` plní téměř stejný účel, a tak jsem je tedy spojil dohromady. Tato třída má obecný název, měla by to tedy být i třída obecného MP. `MPFFT` a `GMPA` jsou pak její potomci, konkrétní způsoby implementace. Podívejte se do schématu na obrázku 7.1.

Třída `GMPA` díky dědičnosti všechny důležité části, jako je základní algoritmus MP, získala od svého rodiče. Pro správné fungovaní stačí založit v



Obrázek 7.2: Schéma tříd genetického algoritmu matching pursuit

konstruktoru instanci nově implementované báze[1] a zrušit fungování funkce `getBases()`, která dovolovala modifikaci seznamu bází.

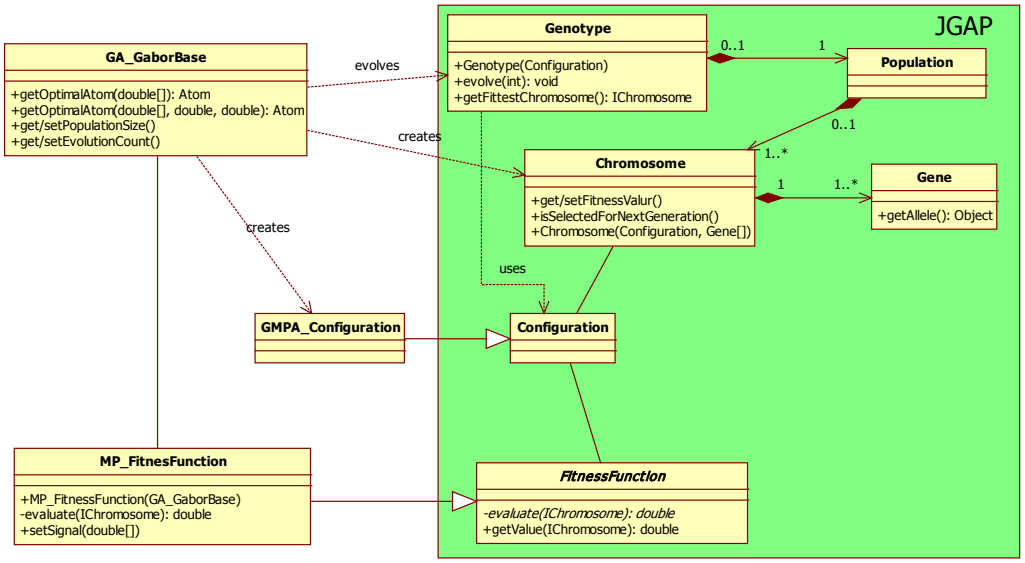
Vše ostatní funguje podle zaběhnutého schématu. Bylo ale třeba naprogramovat nové verze zmíněné báze (`GA_GaborBase`) a Gáborova atomu (`MyGaborsAtom`). Ten jsem obohatil hlavně o funkci, která vrací normovaný průběh atomu. Schéma GMPA naleznete na obrázku 7.2.

7.2.2 Nová verze báze

V této kapitole bych vysvětlil schéma nejdůležitější části, *báze s genetickým algoritmem*. Popíšu problémy, se kterými jsem se setkal, a pokusím se navrhnout možná řešení. Kapitola se věnuje třídě `GA_GaborBase`, ve které probíhá výběr Gáborova atomu pomocí genetického algoritmu.

`Configuration` je třída knihovny JGAP, do které se ukládají všechny důležité parametry a součásti GA. Do `Configugation` nastavujeme velikost populace, genetické operátory (`GeneticOperator`), vlastnosti přírod-

[1]Báze je v implementaci totéž, co slovník funkcí, který známe z teorie MP



Obrázek 7.3: Schéma báze, která využívá genetický algoritmus

ního výběru (`NaturalSelector`), algoritmus, podle kterého probíhá vývoj GA (`Breeder`), generátor náhodných čísel a mnoho dalších.

Konfigurace také před spuštěním potřebuje vzorový chromosom a připravenou fitness funkci. Chromosom vytváří třída `GA_GaborBase` a umístí do něho čtyři instance typu `IntegerGen`, kterým nastaví rozsahy podle pravidel z kapitoly 5.1.2. Jedinou instancí, která se dá používat opakovaně, je `MP_FitnessFunction`. Tu budeme při každém spuštění aktualizovat novým vstupním signálem, aby mohla počítat skalární součin signálu a průběhu atomu.

Průběhy atomů fitness funkci počítá `GA_GaborBase`, ačkoli to není dobré řešení. Mnohem srozumitelnější by bylo, kdyby chromosom sám předal fitness funkci svůj průběh (resp. průběh reprezentovaného atomu). Bylo by tedy vhodné realizovat potomka třídy. Zde jsem ale narazil na složitost implementace třídy `Chromosome`, kterou ostatní třídy během evoluce klonují, ale zatím pouze jako typ `Chromosome`. Fitness funkci jsou pak nové funkce nedostupné, protože špatně naklonované chromozomy je prostě nemají. To by se však dalo po hlubším prozkoumání knihovny vyřešit.

Třída `Genotype` obsahuje populaci a spouští evoluci. Tu je třeba na začátku naplnit náhodně vygenerovanými jedinci a může se přejít k nejzajíma-

vější části GA – evoluci. Tuto část zajišťuje třída `GABreeder`, kterou také při konstrukci ukládáme do konfigurace. Právě v této třídě stráví algoritmus drtivou většinu času. Tuto třídu by se tedy vyplatilo naprogramovat znovu a zvýšit tak rychlost algoritmu. Samotný průběh evoluce nabízený autory knihovny je poměrně neortodoxní. Například se v něm nepoužívá reprodukční jezírko a velikost populace mezi generacemi mírně kolísá. Prostor pro vylepšování jistě existuje, ale v rámci bakalářské práce na to nezbyl čas.

7.2.3 Konfigurace pro GMPA

V programu jsem vytvořil a použil vlastní konfiguraci GA. Třída této konfigurace se jmenuje `GMPA_Configuration` a většinu nastavení provede už při konstrukci. Následující nastavení je přizpůsobeno velikosti prohledávaného prostoru (závislé na a délce signálu).

- velikost populace = délka signálu / 10
- počet generací = délka signálu / 15
- výběr před rozmnožováním žádný, jezírko nepoužito
- křížení (1/6 populace)
- mutace (1/15 populace)
- výběr do další generace – soutěží (3 ch. v soutěži, ppst. 0,9)
- elitismus: 1 nejlepší jedinec vždy přežívá

Tato konfigurace, přesto že funguje docela dobře, by se dala ještě hodně vylepšit. Také by se měl změnit algoritmus třídy `GABreeder`, aby více odpovídal tradiční verzi a byl výkonnější.

8 Porovnání různých implementací MP

Tato kapitola se zabývá zejména ověřením funkčnosti všech implementovaných algoritmů. Implementoval jsem celkem tři algoritmy:

- Genetický matching pursuit (GMPA)
- GMPA a FFT akcelerací „po jednom“ (GMPA1)
- GMPA s obecnou FFT akcelerací (GMPA2)

Mohu prozradit, že všechny verze úspěšně řeší úlohu MP, tedy poměrně efektivně dekomponují signál na atomy, kterými se dá následně aproximovat vstupní signál.

8.1 Srovnání výkonu a přesnosti

Je načase ověřit předpoklady předchozích kapitol o chybách a přednostech vytvořených algoritmů a porovnat zcela nové algoritmy (GMPA1 a GMPA2) s klasickým algoritmem matching pursuit využívajícím FFT (MPFFT), který je součástí knihovny EEGDSP.

Testování probíhalo na reálných datech. Jedná se o EEG signál naměřený v laboratoři neuroinformatiky na KIV. Délka vstupního signálu se v průběhu testování měnila, aby byl naznačen vliv algoritmické složitosti na dobu výpočtu a vzrůstající náročnost aproximace delšího signálu. Zvolil jsem délky 128, 256, 512, 1024 a 2048 vzorků, protože algoritmy používající FFT umí bez úprav pracovat pouze se signály délky mocniny dvou. V ostatních případech si signál doplňují nulami na délku mocniny dvou, což by bylo znevýhodňující. V následujících tabulkách naleznete průměrnou dobu výpočtu v milisekundách a kvalitu aproximace signálu danou metodou. Doby výpočtů jsou v tabulce 8.3. Kvalitu aproximace ukazují tabulky 8.1 a 8.2. Jedná se o průměry z 93 měření (resp. z 50 pro 1024 vzorků nebo z 15 měření pro 2048 vzorků). Všechny testované algoritmy rozkládaly signál na 10 atomů.

N	MPFFT	GMPA	GMPA1	GMPA2
128	13,5421	26,1372	18,2401	25,5674
256	20,4431	25,4277	23,1090	32,2521
512	26,3324	28,0776	27,6450	35,1905
1024	28,0629	28,6171	28,9216	37,8862
2048	26,8849	26,8020	27,4793	39,6206

Tabulka 8.1: Průměrná chyba na vzorek (μV)

N	MPFFT	GMPA	GMPA1	GMPA2
128	0,9330	0,7677	0,8769	0,8078
256	0,8595	0,7735	0,8141	0,7730
512	0,8412	0,8100	0,8159	0,7814
1024	0,8226	0,8134	0,8145	0,7654
2048	0,7879	0,7896	0,7764	0,7209

Tabulka 8.2: Průměrná korelace

Pro měření přesnosti aproximace jsem použil korelaci a chybu na vzorek. Korelace nabývá hodnot od 0 do 1. Čím více se blíží jedničce, tím jsou si signály podobnější. Chyba na vzorek je v jednotkách měřeného signálu (v našem případě μV) a získáme jí pomocí následujícího vztahu:

$$c = \sum_{i=0}^{N-1} |s_i - a_i|$$

$$c^* = \frac{c}{N}$$

, kde c je celková chyba, c^* chyba na vzorek, a je vektor aproximace a s je vektor vzorového signálu. Tento způsob je dle mého názoru o něco průkaznější, ale hodnoty obou tabulek zjevně nejsou v rozporu.

Nejpřesnější metodou je jistě klasický MP z knihovny EEGDSP. Když se ale podíváme do tabulky doby výpočtu (tabulka 8.3), zjistíme, že ostatní metody sice nedosahují takové kvality, ale zato jsou mnohonásobně rychlejší. Nejvýraznější je to na krátkých signálech. S rostoucí délkou se rozdíl zmenšuje.

Test dále ukazuje, že problémy GMPA2 s přesností se projevují poměrně značně. Proto je nejméně přesnou metodou ze všech. Je pro mě překvapením, že ani v rychlosti nepřekoná příbuznou metodu GMPA1, přestože předpoklad byl opačný. GMPA dosahuje stabilně celkem přijatelných přesností a do délky 1024 je rychlejší než MPFFT. Od délky 2048 už je o dost pomalejší.

N	MPFFT	GMPA	GMPA1	GMPA2
128	99	13	3	2
256	401	81	19	19
512	1585	591	169	169
1024	6396	4705	1298	1300
2048	26658	37335	10406	10449

Tabulka 8.3: Průměrná doba na zpracování jedné epochy (ms)

GPMA1 (tj. s FFT akcelerací po jednom) dosahuje téměř stejných přesností jako GMPA a rychleji než GMPA. Na nejčastěji používaných délkách 512 a 1024 vzorků je jen o něco málo méně přesný než MPFFT, za to je ale 9krát resp. 5krát rychlejší. Pokud jede o poměr rychlosti a přesnosti, hodnotil bych pro tyto délky GMPA1 z porovnávaných algoritmů jako nejlepší.

8.2 Zjištění algoritmické složitosti

Z testu také vyplývá, že náskok GMPA1 nad metodou MPFFT se zmenšuje s rostoucí délkou signálu. To naznačuje, že by GMPA1 přes svou slušnou rychlost na malém počtu vzorků měl mít horší algoritmickou složitost než MPFFT. Klasický algoritmus matching pursuit má složitost $O(Nlog^2N)$. Rozhodl jsem se tedy pečlivěji prozkoumat výkon metody GMPA1 a odhadnout její složitost.

Na rozdíl od předchozího testu jsem měřil dobu výpočtů pro více různých délek signálu a zajímal jsem se pouze o GMPA1. Výsledky najdete v tabulce 8.4.

Protože GA je velmi složitý, rozhodl jsem se pro experimentální zjištění algoritmické složitosti. N je počet vzorků signálu a $T(N)$ je průměrná doba trvání zpracování jednoho signálu algoritmem. Složitost pak zjišťuji tak, že zkouším $T(N)$ dělit různými funkcemi $f(N)$. Vyjde-li průběh $T(N)/f(N)$ konstantní, $f(N)$ je pravděpodobně funkce algoritmické složitosti.

Tabulka 8.4 ukazuje, že určení složitosti GMPA1 bude jen přibližné. Složitost vychází určitě větší než $O(N^2log^2N)$ a menší než $O(N^3)$. Lépe to bude vidět z grafů (obrázek A.3 v přílohách). Spíše se ale přiklání k $O(N^3)$, protože od $N = 1024$ je průběh podílu téměř dokonale konstantní.

N	256	512	1024
T(N)	28	170	1289
$T(N)/N^2$	0,000427246	0,000648499	0,001229286
$T(N)/N^2log^2N$	7,36679E-05	8,83496E-05	0,000135654
$T(N)/N^3$	1,66893E-06	1,2666E-06	1,20047E-06
$T(N)/N^4$	6,51926E-09	2,47383E-09	1,17234E-09
N	2048	4096	8192
T(N)	10320	82382	657601
$T(N)/N^2$	0,00246048	0,00491035	0,009799
$T(N)/N^2log^2N$	0,000224396	0,000376297	0,00064
$T(N)/N^3$	1,20141E-06	1,19882E-06	1,2E-06
$T(N)/N^4$	5,86624E-10	2,9268E-10	1,46E-10

Tabulka 8.4: Hodnoty podílu $T(N)/f(N)$ pro různá N

Ani jedna z možností není dobrý výsledek. Je ale možné, že GA je pro delší signály předimenzovaný a zbytečně běží „na prázdno“, přestože už výsledek vylepšit nejde. Možná, kdyby se upravil způsob výpočtu počáteční populace a počtu generací, algoritmická složitost by se mohla snížit bez zásadního vlivu na přesnost.

9 Závěr

Ve své bakalářské práci jsem ukázal, že je možné využít genetického algoritmu pro implementaci matching pursuit. Po vhodné volbě fitness funkce a vyřešením reprezentace genů genetický algoritmus spolehlivě řeší úlohu výběru nejlepšího atomu ze slovníku.

Po prostudování literatury a zhodnocení faktů jsem naprogramoval celkem tři různé druhy genetického matching pursuit. Základní GMPA používá k optimalizaci všech čtyř parametrů Gaborových atomů pouze genetický algoritmus. Tento algoritmus sice úlohu řeší, ale docela pomalu. Proto jsem s pomocí mého vedoucího Ing. Pavla Mautnera, Ph.D. navrhl a naprogramoval další dva algoritmy, které usnadňují práci genetickému algoritmu tím, že pro nalezení optimální frekvence a fáze využívají rychlé Fourierovy transformace. První z metod, GMPA s obecnou FFT akcelerací, si vytvoří frontu frekvencí a fází ještě před iteračním cyklem, zatímco druhý způsob, GMPA s FFT akcelerací „po jednom“, hledá dvojici parametrů v každé iteraci z aktuálního signálu.

Testování algoritmů dopadlo nejednoznačně. Algoritmy, které jsem naprogramoval, sice nedosahují tak výborných aproximací jako běžný matching pursuit, ale zato byly rychlejší. Nejlepší z mých algoritmů, GMPA s FFT akcelerací „po jednom“, pracuje velice rychle a s dobrou přesností i v porovnání s klasickým matching pursuit. Jeho algoritmická složitost je ale $O(N^3)$. Přesto je pro běžně užívané délky signálu (512 – 1024 vzorků) několikrát rychlejší než klasický algoritmus. Mám už několik návrhů na jeho vylepšení, která by mohla nepříznivou složitost zmenšit. Například by se mohl změnit způsob výpočtu počáteční populace a počtu generací tak, aby tyto parametry rostly logaritmicky místo lineárně vzhledem k délce vstupního signálu.

Implementace samotného genetického algoritmu by potřebovala výrazně upravit evoluční cyklus, aby více odpovídal tradiční verzi a byl výkonnější. Dále bych vyzkoušel kódování parametrů atomu binárními geny a několik dalších úprav.

Pro další práci obdobného charakteru bych rozhodně doporučil využití FFT akcelerací, protože testy potvrdily, že urychlení výpočtu je za stejné přesnosti značné. Navrhoval bych vytvořit FFT s vyšší přesností frekvence a vyzkoušet akcelerace s dalšími optimalizačními metodami, jako je simulované žíhání nebo hillclimbing.

Přehled zkratek

EEG	Elektroencefalografie (z angl. electroencephalography)
ERP	Evokovaný potenciál (z angl. event-related potential)
FT	Fourierova transformace (z angl. Fourier transform)
DFT	Diskrétní Fourierova transformace (z angl. discrete Fourier transform)
FFT	Rychlá Fourierova transformace (z angl. fast Fourier transform)
MP	Matching pursuit
MPFFT	Matching pursuit využívající FFT
GMPA	Genetický algoritmus matching pursuit (z angl. Genetic matching pursuit algorithm)
EEGDSP	Knihovna pro zpracování signálů zaměřená na EEG/ERP signály
JGAP	Knihovna na vytváření genetických algoritmů a genetického programování (z angl. Java genetic algorithms package)

Obsah CD

- **build** – Adresář obsahuje spustitelnou verzi programu.
- **data** – Adresář obsahuje naměřená data.
- **doc** – Adresář obsahuje pdf s textem bakalářské práce, pdf s uživatelskou dokumentací programu ERP-Classifier, soubory pro vygenerování bakalářské práce programem `pdfcslatex.exe`, použité obrázky a vygenerovanou programátorskou dokumentaci JavaDoc.
- **src** – Adresář obsahuje zdrojové soubory pro překlad programu ve formě projektu pro vývojové prostředí NetBeans. Nachází se zde i soubor `build.xml` pro překlad programu z příkazové řádky. Pro vytvoření spustitelného jar souboru je možno použít příkaz ant jar.

Literatura

[1] BEASLEY, D. – BULL, D. R. – MARTIN, R. R. *An Overview of Genetic Algorithms: Part 1, Fundamentals*. University Computing, 1993.

[2] DAN, S. – FLORIN, I. *A genetic matching pursuit algorithm.*Signal Processing and Its Applications,2003,1: 577 580

[3] DAVÍDEK, V. – SOVKA, P. *Číslicové zpracování signálů a implementace*, Skripta. 1. vydání. Praha: Vydavatelství ČVUT, 1996. ISBN 80-0101-530-0.

[4] KELLNHOFER, P. *Knihovna pro zpracování signálů*, Semestrální práce KIV-AZS. Verze 1.0, Fakulta aplikovaných věd ZČU v Plzni, 2011.

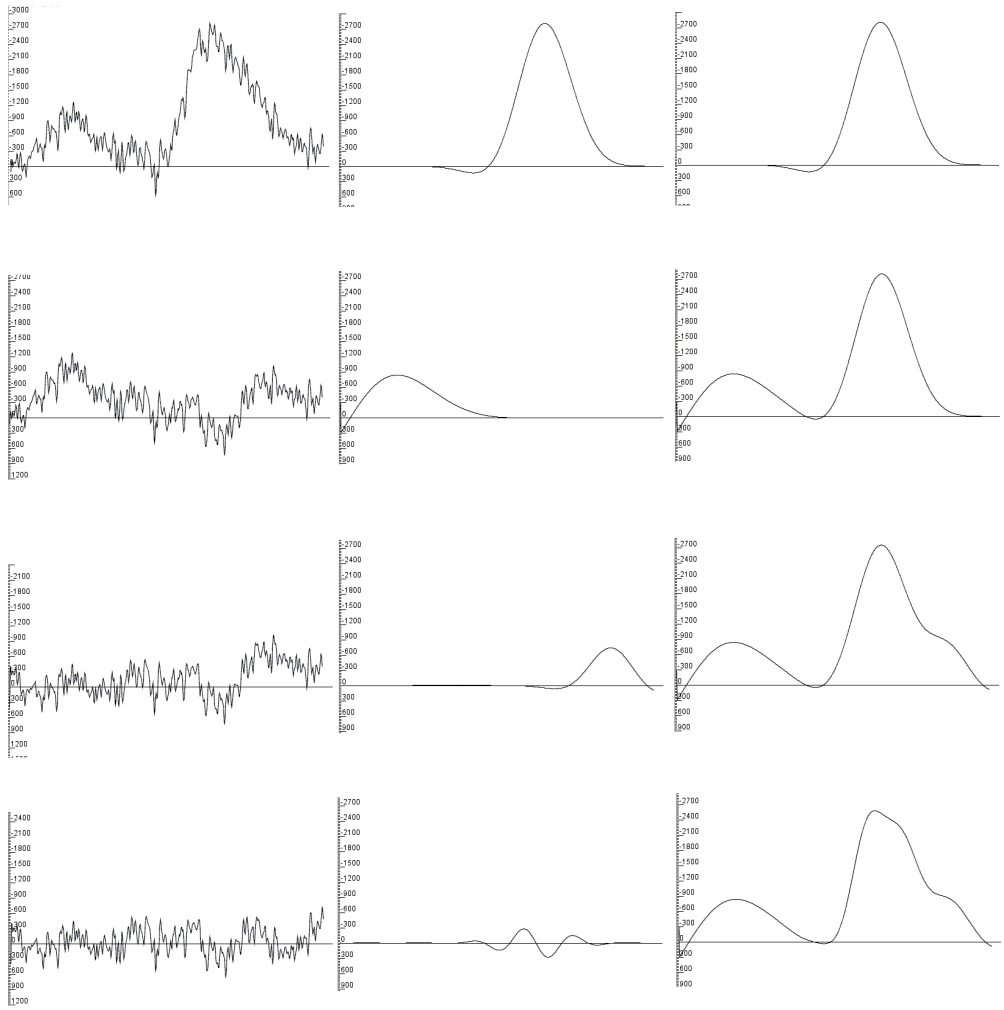
[5] MALLAT, S. G. – ZHANG, Z. *Matching Pursuit with Time-Frequency Dictionaries*. IEEE Trans. On Signal Proc., 1993. Vol. 41, No. 12

[6] MAUTNER, P. – MOUČEK, R. *Využití metody matching pursuit pro detekci ERP vln*, sborník IX. konference Kognice a umělý život. Stará Lesná, Slovensko: 2009. pp. 201-205.

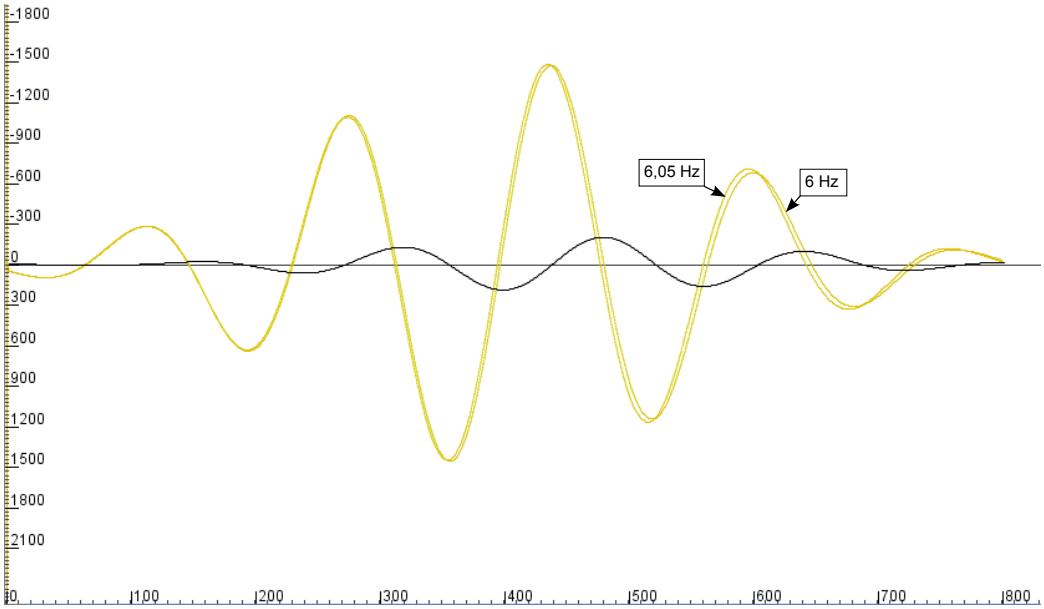
[7] MITCHELL, M. *An Intoduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts, 1995.

[8] ŘONDÍK, T. *Metody zpracování ERP signálů*, diplomová práce. Fakulta aplikovaných věd ZČU v Plzni, 2010.

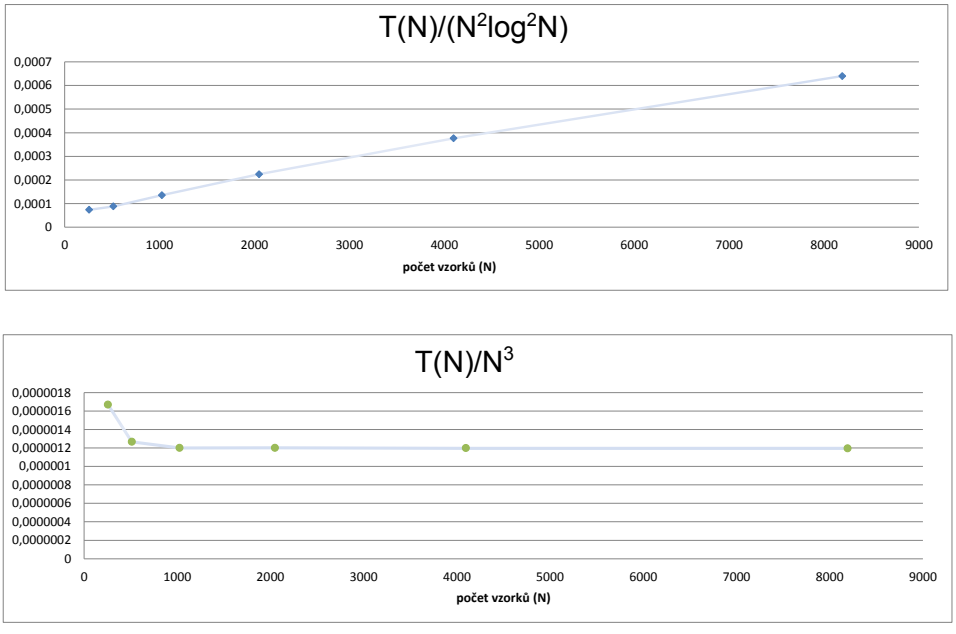
A Příloha: Obrázky



Obrázek A.1: Ukázka dekompozice vstupního signálu a residua (levý sloupec) na čtyři atomy (prostřední sloupec) a složení aproximace (pravý sloupec)

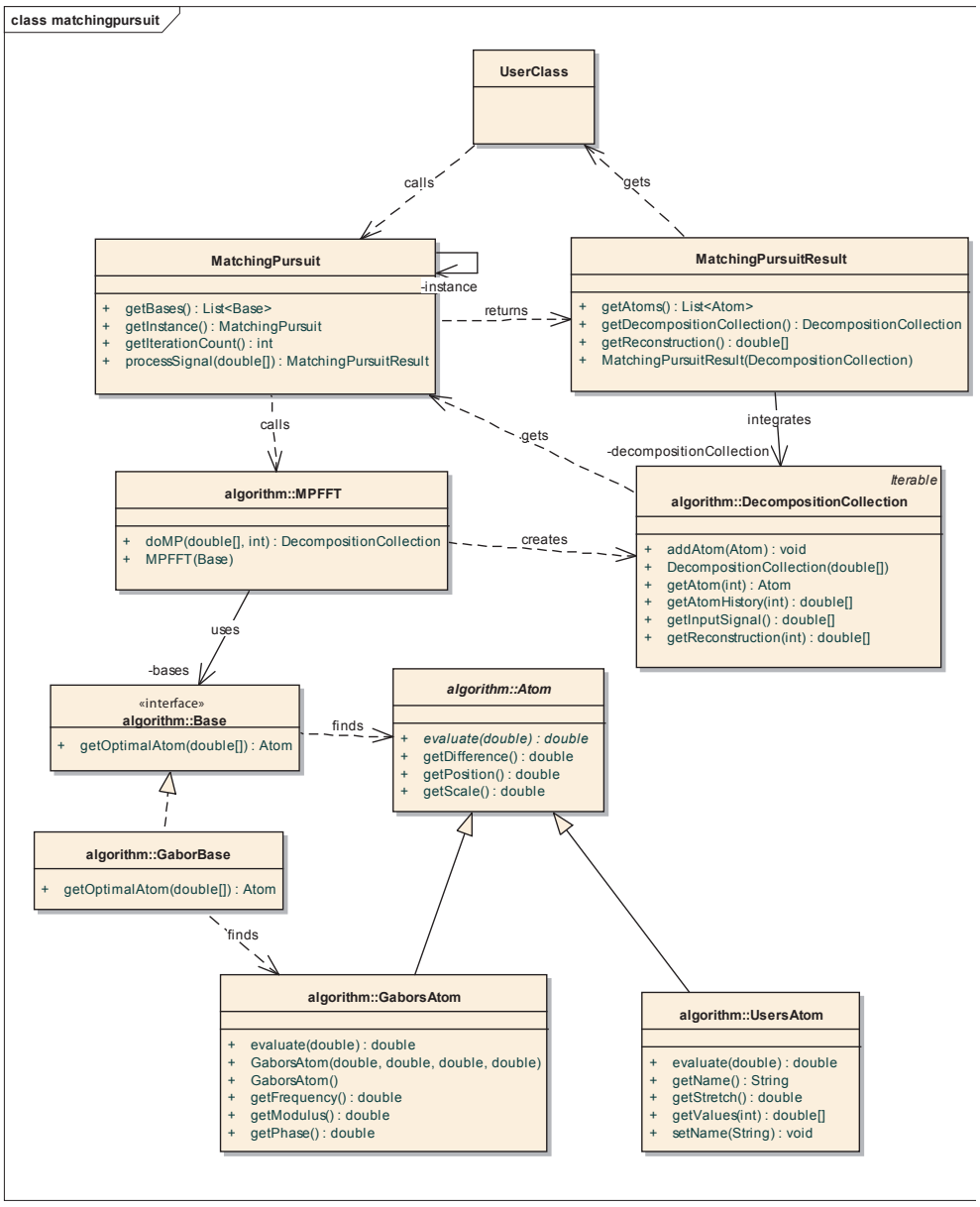


Obrázek A.2: Ukázka dopadů nedokonalé aproximace; tmavý signál je rozdíl dvou světlých signálů

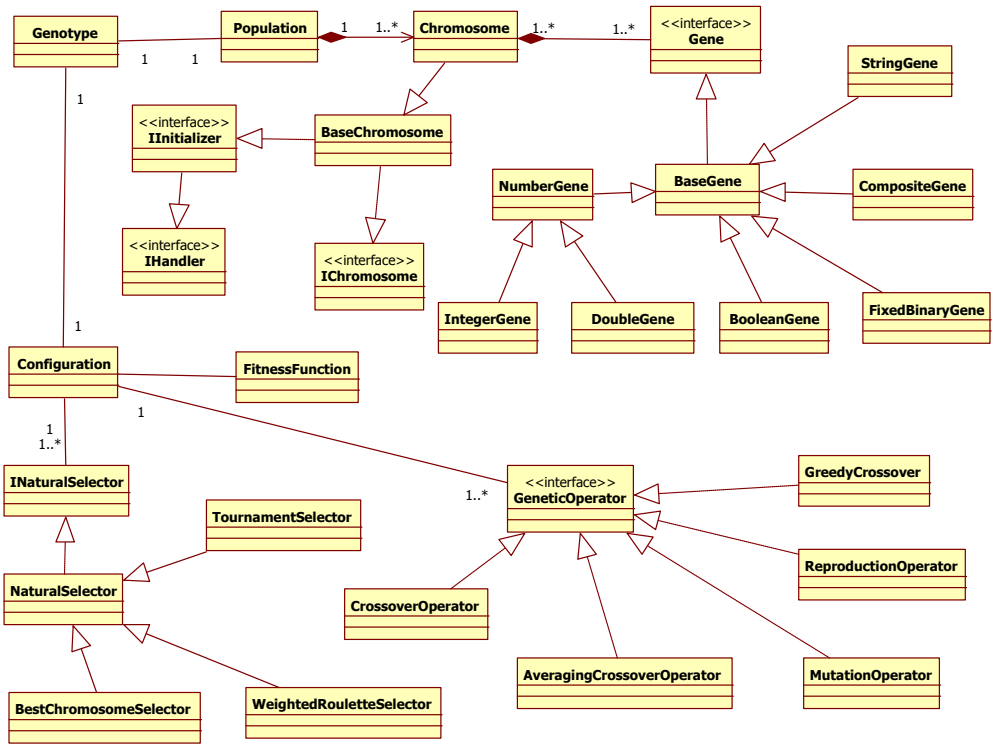


Obrázek A.3: Grafy průběhu T(N)/f(N)

B Příloha: UML diagramy



Obrázek B.1: Schéma algoritmu matching pursuit z knihovny EEGDSP, převzato z [4]



Obrázek B.2: Schéma základních tříd knihovny JGAP

C Příloha: Uživatelská dokumentace ERP-Classifier

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

ERP-CLASSIFIER
UŽIVATELSKÁ PŘÍRUČKA

Plzeň, 2012 Jan Strejc

Obsah

1 Program

1.1 Překlad

Pro překlad jsou k dispozici zdrojové soubory a Ant skript v souboru *build.xml*. Pro vytvoření spustitelného *.jar* souboru použijte cíl *jar*. Je také přiložena celá složka s projektem z NetBeans a tak je možné program snadno upravit a přeložit pomocí NetBeans.

1.2 Instalace

Program není nutné instalovat, je možné nakopírovat do vlastního počítače a spouštět jej odtamtud. Na přiloženém CD je k dispozici přeložená verze programu pro Javu 1.7. Kromě souboru *erp-classifier.jar* je nutné do stejné složky nakopírovat složku lib s knihovnami.

1.3 Požadavky

Program pro svůj běh vyžaduje nainstalovanou aktuální verzi Javy 1.7 (mělo by být možné spustit jej na starší verzi 1.6, ale bezchybná funkčnost není zaručena a bude nutné provést rekompilaci). Pro bezproblémový chod je doporučeno mít alespoň 1 GB volné paměti RAM.

Hlavní platformou pro použití je MS Windows, ale program je možné používat i na Linuxu, nebo Mac Os (testováno na Windows 7 a Ubuntu 12.04).

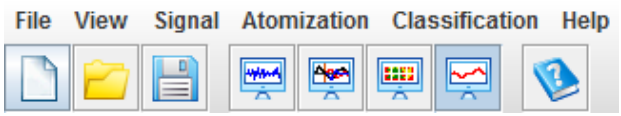
1.4 Spuštění

Pokud je správně nainstalována Java tak je možné spouštět program dvojklikem na soubor *erp-classifier.jar*. Jinak lze spuuštění provést z příkazové řádky příkazem *java -jar erp-classifier.jar*.

2 Uživatelské rozhraní

2.1 Menu

Menu slouží jako hlavní ovládací prvek aplikace a jsou z něj přístupné všechny funkce, ty nejčastější jsou umístěny na toolbaru pod menu (Obrázek 2.1). Aktuálně nepoužitelné položky menu jsou šedivé a nejde je použít.



Obrázek 2.1: Menu a toolbar

Seznam jednotlivých položek menu:

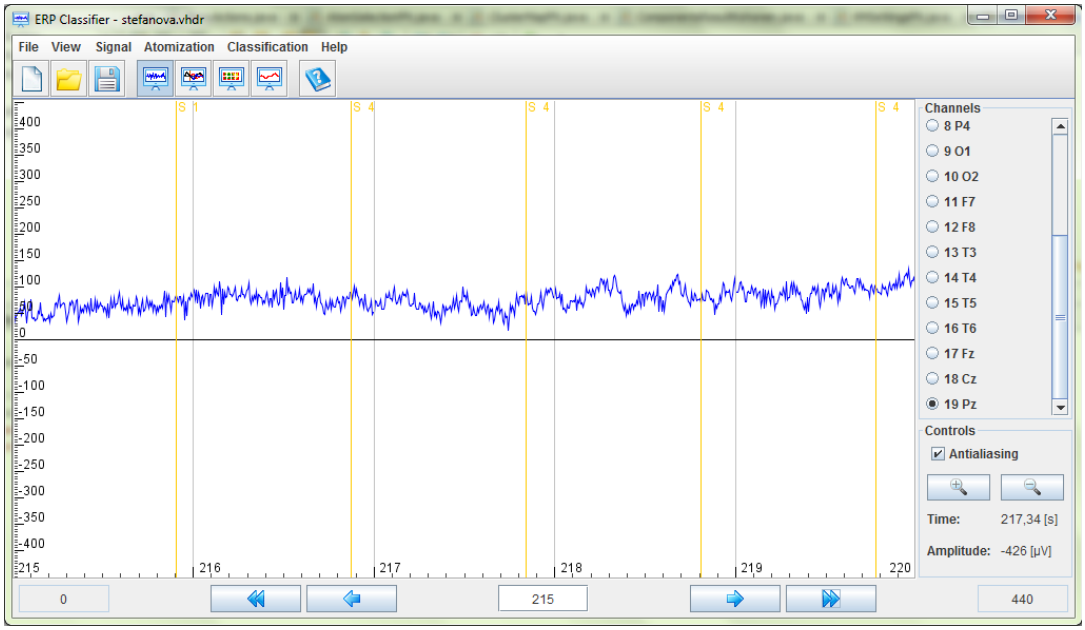
- **File**
  - **New Project** – vytvoří nový projekt z VHDR souboru.
  - **Open Project** – otevře uložený projekt z XML souboru.
  - **Save Project** – uloží rozpracovaný projekt do XML souboru.
  - **Exit** – ukončí aplikaci.
- **View**
  - **View Signal** – zobrazí načtený neupravený signál.
  - **View Epochs and Atoms** – zobrazí extrahované epochy a jejich atomy vytvořené matching pursuitem.
  - **View Clusters** – zobrazí neuronovou sítí vytvořené shluky atomů.
  - **View Components** – zobrazí okno s tvorbou komponent.
- **Signal**
  - **Epoch Extraction** – zobrazí panel pro extrakci epoch.
  - **Epoch Averaging** – zobrazí panel s filtry a přepne na záložku pro průměrování epoch.
  - **Treshold Fitrer** – zobrazí panel s filtry a přepne na záložku pro prahový fitr.
  - **FIR Fitrer** – zobrazí panel s filtry a přepne na záložku pro FIR filtr.
  - **Moving Window Fitrer** – zobrazí panel s filtry a přepne na záložku pro filtr pohybujícím se okénkem.

  - **Signal Averaging Fitrer** – zobrazí panel s filtry a přepne na záložku pro filtr na průměrování signálu.

- **Atomization**
  - **GMPA** – přepnutí modulu atomizace na Genetický matching pursuit.
  - **MPFFT** – přepnutí modulu atomizace na obyčejný matching pursuit z EEGDSP.
  - **Settings** – nastavení aktuálně vybraného modulu atomizace.
  - **Atomize** – spustí atomizaci aktuálně vybraným modulem.
- **Classification**
  - **Kohonen neural network** – přepnutí modulu klasifikace na Kohonenovu neuronovou síť.
  - **ART2 neural network** – přepnutí modulu klasifikace na ART2.
  - **Settings** – nastavení aktuálně vybraného modulu klasifikace.
  - **Classify** – spustí klasikaci aktuálně vybraným modulem.
  - **Atomize and Classify** – spustí atomizaci a pak klasikaci aktuálně vybranými moduly.
- **Help**
  - **Help** – zobrazí nápovědu k programu.

2.2 Načtení signálu

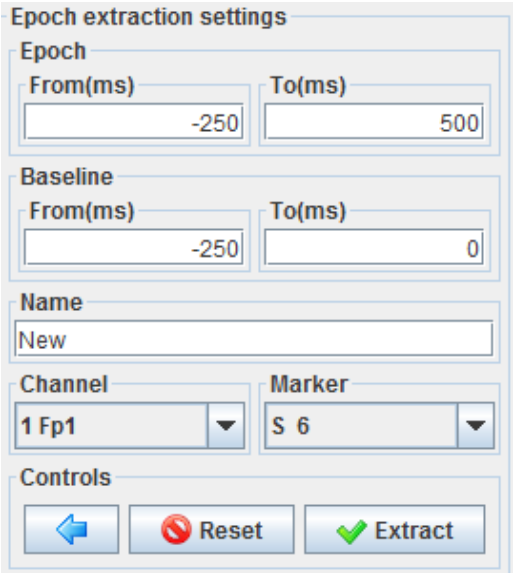
Pomocí *File* → *New Project* vybereme VHDR soubor, který chceme načíst. Poté se nám objeví okno s načteným signálem, později je možné signál zobrazit pomocí *View* → *Signal* (Obrázek 2.2). V pravé části je možno si vybrat kanál, který chceme zobrazit dole se můžeme pomocí šipek posouvat v signálu.



Obrázek 2.2: Zobrazení signálu

2.3 Extrakce epoch

Pro extrakci epoch musí být načtený signál. Pro nastavení extrakce klikneme na *Signal* → *Epoch Extraction* a zobrazí se nám následující panel (Obrázek 2.3).



Obrázek 2.3: Extrakce epoch

První dvě políčka určují kde bude epocha začínat a končit (vzhledem k markeru), druhá dvě určují úsek epochy, který bude použit pro korekci baseline. Skupinu je možné pojmenovat zadáním jména do políčka *Name*. Políčko *Channel* slouží pro výběr kanálu (elektrody), jehož signál se použije a políčko Marker pro výběr markeru (stimulu), od kterého chceme epochy vytvořit.

Extrakci spustíme tlačítkem *Extract*, tlačítko *Reset* vrátí hodnoty do původního stavu a šipka zpět panel schová.

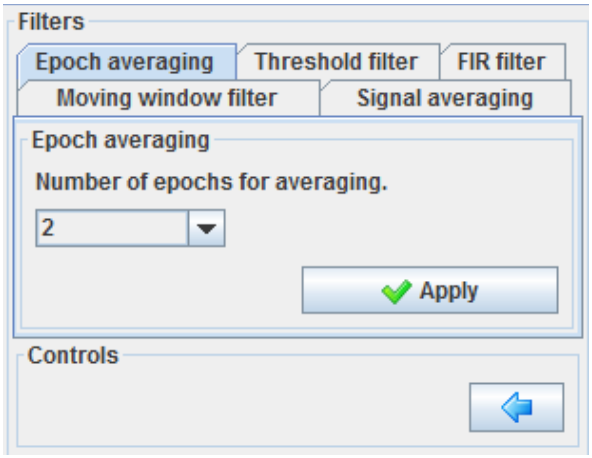
Po dokončení extrakce se zobrazí okno s epochami (Obrázek 2.4). V dolní části se nachází ovládání pro přepínání epoch a možnost odebrání epochy, která se nám nelíbí. V pravo je seznam zobrazených signálů (zde později přibudou atomy a signál epochy aproximovaný matching pursuitem) a pod ním je ovládání grafu (zoom, vyhlazování, možnost otočit osu Y a informace aktuální pozici v signálu). Svislá oranžová čára na grafu představuje marker, osa Y představuje napětí v μV a osa X čas v milisekundách.

Obrázek 2.4: Zobrazení epoch

2.4 Filtry a průměrování epoch

2.4.1 Průměrování epoch

Pro průměrování epoch vybereme *Signal* → *Epoch Averaging*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 2.5).

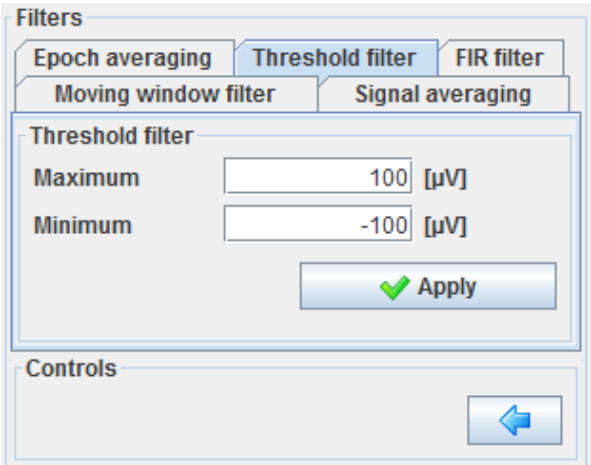


Obrázek 2.5: Průměrování epoch

Zde je možné nastavit kolik epoch má být průměrováno. Tlačítko *Apply* provede průměrování.

2.4.2 Prahový filtr

Pro prahový filtr vybereme *Signal* → *Treshold Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 2.6).

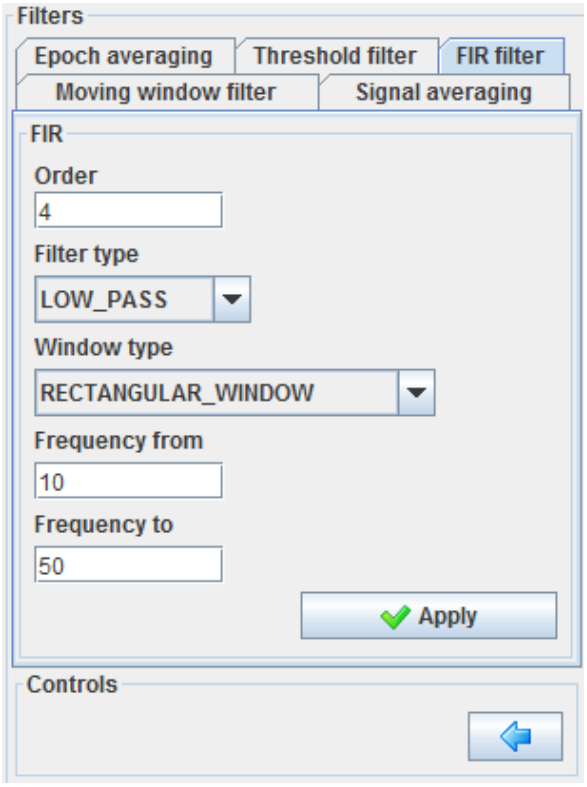


Obrázek 2.6: Prahový filtr

Zde je možno nastavit minimální a maximální hranici, která je přípustná pro signál v epochách, po spuštění tlačítkem *Apply* filtr odstraní ty epochy, které se do tohoto limitu nevejdou.

2.4.3 FIR filtr

Pro FIR filtr vybereme *Signal* → *FIR Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 2.7).

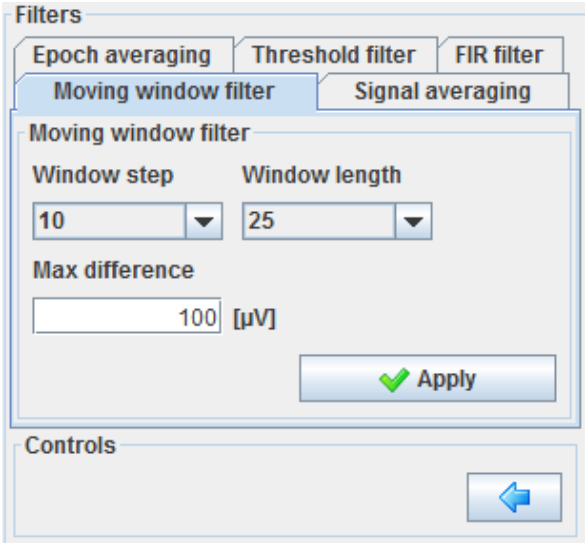


Obrázek 2.7: FIR filtr

Zde je možno nastavit řád filtru (vyšší řád znamená lepší filtraci, ale způsobí posunutí fáze signálu o řád filtru), typ filtru, typ okénka a dolní a horní frekvenci pro jednotlivé filtry. Tlačítkem *Apply* spustíme filtraci.

2.4.4 Filtr pohybujícím se okénkem

Pro filtr pohybujícím se okénkem vybereme *Signal* → *Moving Window Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 2.8).

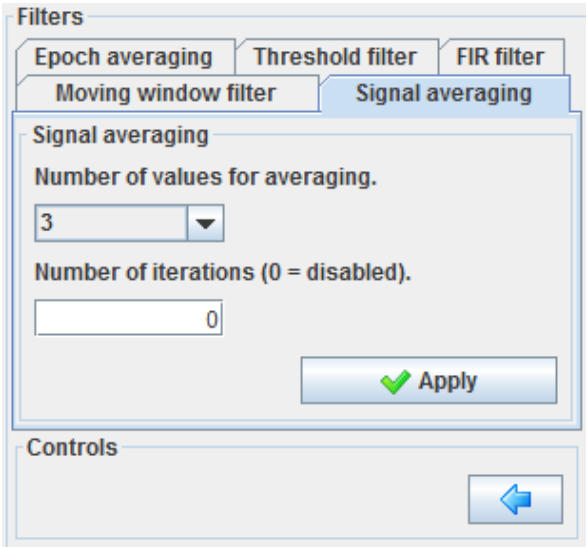


Obrázek 2.8: Filtr pohybujícím se okénkem

Zde můžeme nastavit krok po jakém se bude okénko posouvat, jeho délku a maximální rozdíl hodnot signálu v okénku. Po spuštění tlačítkem *Apply* filtr odstraní ty epochy, které se do tohoto limitu nevejdou.

2.4.5 Filtr pro průměrování signálu

Pro filtr průměrování signálu vybereme *Signal* → Signal Averaging *Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 2.9).

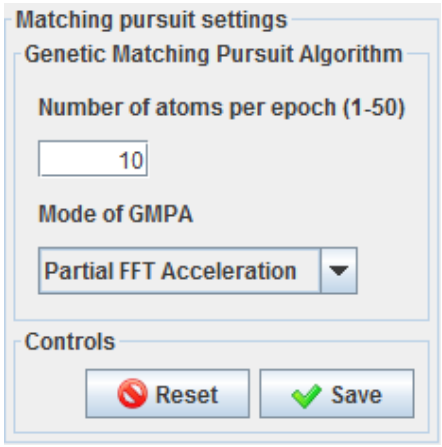


Obrázek 2.9: Filtr pro průměrování signálu

Zde si můžeme vybrat počet hodnot pro průměrování a počet iterací filtru. Nastavení příliš velkého počtu hodnot a iterací může ze signálu odstranit důležité detaily. Tlačítkem *Apply* spustíme filtraci.

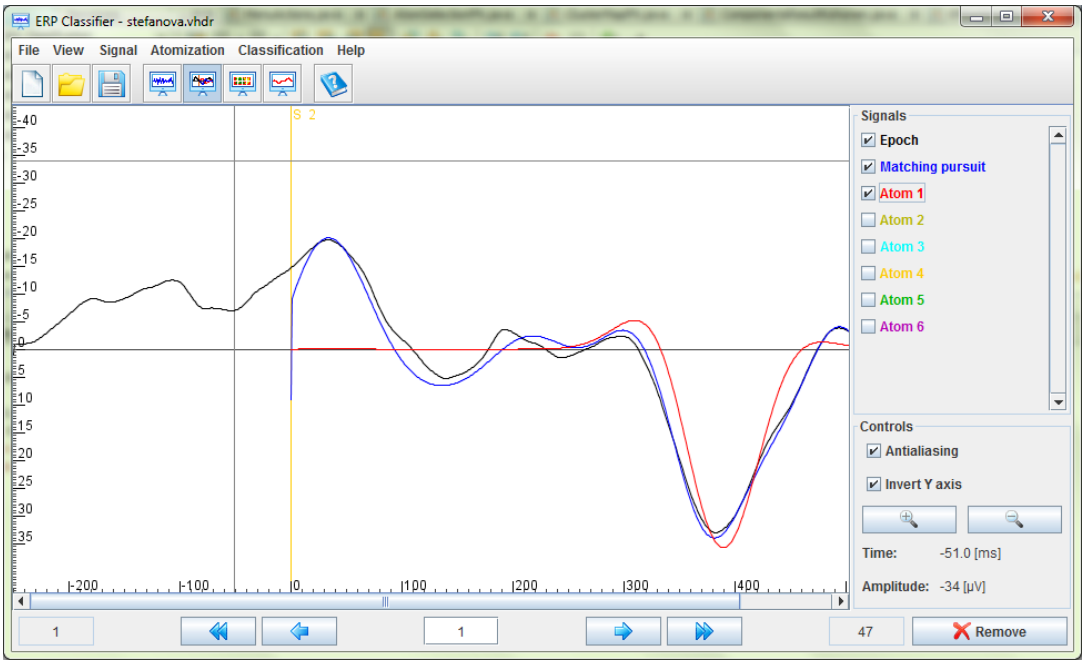
2.5 Atomizace

V menu *Atomization* můžeme vybrat, který modul pro atomizaci chceme použít. Pak si můžeme zobrazit jeho nastavení. Na obrázku 2.10 je nastavení pro Genetický matching pursuit, zde je k dispozici nastavení počtu atomů na epochu a mód matching pursuitu.



Obrázek 2.10: Nastavení matching pursuitu

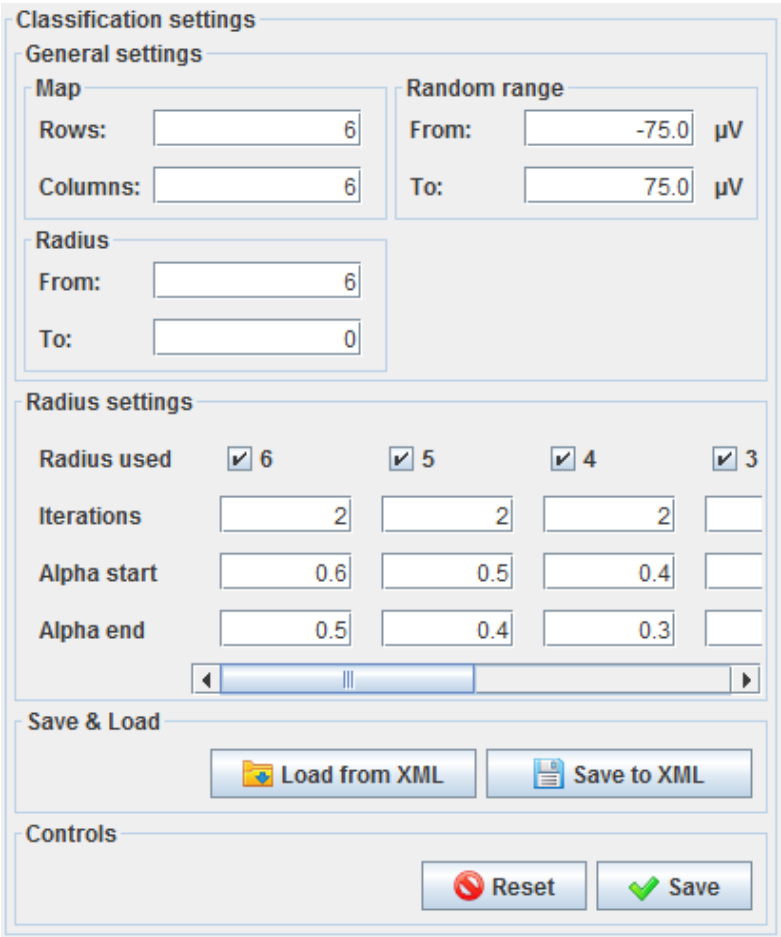
Tlačítkem *Save* uložíme nastavení. V menu pak nejdeme ještě položku *Atomize*, která spustí tvorbu atomů. Na obrázku 2.11 jsou vidět výsledné atomy, ovládání se shoduje s dříve uvedeným oknem pro zobrazení epoch.



Obrázek 2.11: Zobrazení výsledných atomů

2.6 Klasifikace

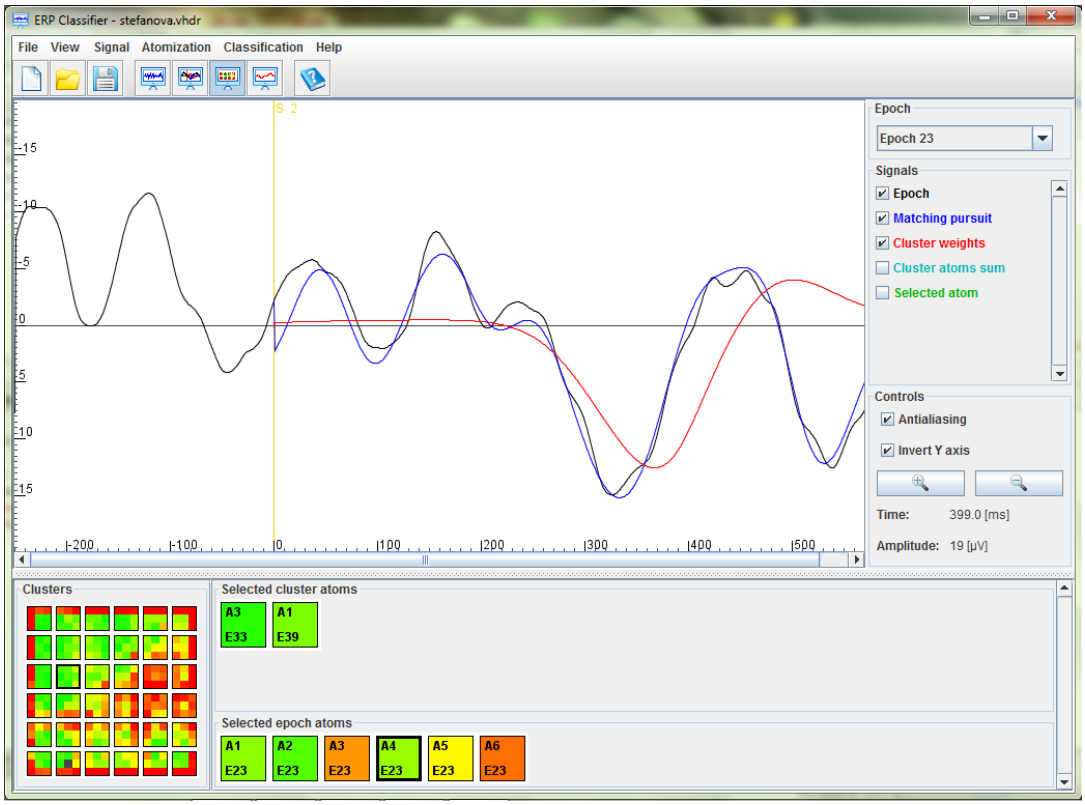
V menu *Classification* si můžeme vybrat jaký modul pro klasifikaci budeme používat. Pak si můžeme zobrazit jeho nastavení. Na obrázku 2.12 je vidět natavení pro Kohonenovu neuronovou síť. Ve vrchní části možno nastavit velikost mapy neuronů, rozsah náhodných hodnot pro váhy sítě a použité radiusy pro aktualizaci vah neuronů. Ve středu se nachází nastavení jednotlivých rasiusů (počet iterací, počáteční a koncov= učení). Dole je možno uložit a nahrát nastavení sítě do a z XML.



Obrázek 2.12: Nastavení Kohonenovi neuronové sítě

Stisknutím tlačítka Save uložíme nastavení. V menu pak nejdeme ještě položku *Classify* pro spuštění klasifikace a položku *Atomize and Classify* pro sekvenční spuštění obou výpočtů.

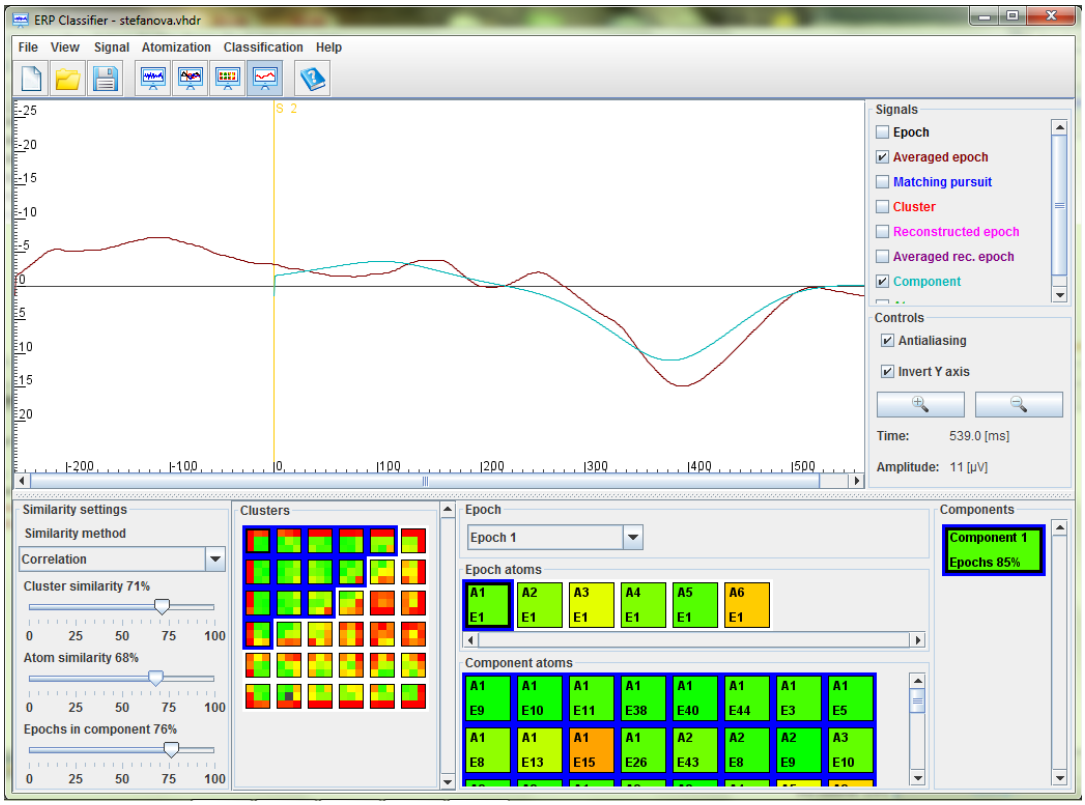
Na obrázku 2.13 je okno pro zobrazení výsledků klasifikace. V pravé části jsou k dispozici různé signály a výběr epochy. V levém dolním rohu je mapa neuronů po kliknutí na některý neuron (jeho shluk) se zobrazí signál jeho vah a průměr signálu shluku jeho atomů. Uprostřed dole jsou potom tlačítka pro zobrazení jednotlivých atomů, nahoře jsou ty z vybraného shluku a dole ty z vybrané epochy. U neuronů barvy zobrazují podobnost vah neuronů jejich sousedům (osm čtverců po obvodu tlačítka) a podobnost atomů jeho shluku s jeho vahami (prostřední čtverec), šedivá znamená že shluk neobsahuje žádné atomy.



Obrázek 2.13: Zobrazení výsledků klasifikace

2.7 Tvorba komponent

Po dokončení klasifikace je k dispozici i okno pro tvorbu komponent z *View* → *Components* (Obrázek 2.14). V pravé části se jsou k dispozici různé signály. V levém spodní rohu je nastavení kritérií pro tvorbu komponent (typ porovnání, podobnost shluků, podobnost atomů shluku a minimální zastoupení epoch v komponentě). Vedle se nachází mapa shluků, na které jsou podbarveny ty, co patří do nějaké komponenty. Vpravo dole je seznam nalezených komponent, po stisknutí se zobrazí uprostřed její atomy a na grafu signálů tvar této komponenty. Uprostřed je možno přepínat zobrazenou epochu a prohlížet atomy vybrané epochy a komponenty.



Obrázek 2.14: Tvorba komponent