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

Bakalářská práce

Python v neuroinformatice

Plzeň, 2013 Pavel Čurda

Prohlášení

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

V Plzni dne 8. května 2013

Pavel Čurda

Abstract

This work deals with possible use and integration of programming language Python in neuroinformatics experiments. It describes both advantages and drawbacks of Java, MATLAB and Python solutions. It covers an overview of existing neuroinformatics algorithms implemented in Python and cooperation between Python and other languages.

The work shows possible benefits of using Python instead of Java and Matlab on a specific example. It is implementation of chosen algorithm used in EEG/ERP experiments in Python and it's performance comparison against existing Java and C++ implementations.

Abstrakt

Tato práce pojednává o možném využití a následné integraci jazyka Python v neuroinformatických experimentech. Popisuje výhody i nevýhody použití jazyků Java, MATLAB a Python v neuroinformatické doméně. Tato práce uvádí přehled existujících implementací algoritmů vhodných pro použití v neuroinformatice v jazyce Python a spolupráci jazyka Python a dalších programovacích jazyků.

Tato práce ukáže možné výhody nasazení jazyka Python místo jazyků MATLAB a Java na konkrétním příkladu: implementací vybraného algoritmu používaném v EEG/ERP experimentech v jazyce Python a jeho výkonnostní porovnání s existujícími Java a C++ implementacemi.

Obsah

1 Úvod

Cílem této práce je seznámit čtenáře s možností využití programovacího jazyka Python a jeho knihoven v oblasti neuroinformatiky a případné začlenění tohoto jazyka do projektů neuroinformatické skupiny KIV na ZČU. Jazyk Python se začíná prosazovat ve všech oblastech IT především díky své jednoduchosti a intuitivnosti. Právě díky těmto vlastnostem se jazyk Python hodí především pro agilní vývoj aplikací a na rychlý vývoj prototypů.

Tato práce shrne a porovná dostupné možnosti implementace Python programů v doméně EEG/ERP experimentů a porovná je s dostupnými řešeními Javy a C++ z hlediska výkonnosti a rychlosti implementace. Zároveň uvede i možné postupy případné spolupráce Pythonu a ostatních programovacích jazyků.

V praktické části práce bude implementován jeden vybraný algoritmus v jazyce Python a následně bude porovnán z hlediska výkonnosti vůči stávajícím řešením v jazycích Java a C++.

2 Programovací jazyky v neuroinformatice

Výběr programovacího jazyka se při vývoji neuroinformatických aplikací odrazí především v rychlosti vývoje daného softwaru i jeho výkonnosti. Pro vybraný programovací jazyk musí existovat dostupné matematické knihovny implementující základní i pokročilé matematické funkce, zejména zpracování navzorkovaného signálu a jeho transformace.

Další kritérium je snadná vizualizace výsledků a těsná integrace s již existujícím softwarem.

2.1 Java

Java je programovací jazyk původně vytvořený společností Sun Microsystems, nyní je spravován a vyvíjen společností Oracle. Katedra KIV upřednostňuje tento jazyk především díky jeho rozšířenosti a přenositelnosti. Java je robustní interpretovaný objektově orientovaný jazyk, který obsahuje již v základu mnoho knihoven, a tím velmi urychluje vývoj prototypů aplikací. Nejrozšířenější interpret jazyka Java je Java Virtual Machine (JVM), který běží na všech nejrozšířenějších platformách.

Hlavní nevýhodou tohoto jazyka je především jeho výkonnost a paměťová náročnost ve srovnání s nízkoúrovňovými jazyky typu C++.

2.2 MATLAB

Prostředí MATLAB od společnosti MathWorks je robustní program určený pro jednoduché i složité matematické výpočty a snadné zobrazení výsledků například pomocí 3D grafů. Tento nástroj je velmi oblíbený především u matematiků a strojařů, nicméně svoje uplatnění najde i v neuroinformatice, kde nabízí velké množství již implementovaných algoritmů potřebných pro práci s analogovým signálem. Zároveň dokáže vhodně spolupracovat s jazykem Java, který dokáže MATLAB nativně spouštět ve svém jádře.

Jazyk MATLAB vychází z nízkoúrovňového programovacího jazyku Fortran, a proto je jeho rychlost výpočtu srovnatelná s implementacemi v jazyce C++. Jeho největší výhoda je vysoká optimalizace matematických operací nad maticemi a vektory čísel s plovoucí desetinnou čárkou.

Dále obsahuje již v základu vysoké množství pokročilých matematických funkcí a obsahuje širokou škálu nástrojů pro snadnou vizualizaci výsledků a jejich následnou analýzu.

Jeho největší nevýhoda je jeho vysoká pořizovací cena.

2.3 Python

Python je skriptovací programovací jazyk podporující objektově orientovaný návrh. V posledních letech se těší zvětšující se přízni programátorů i expertů z jiných profesí především díky jeho snaze usnadnit psaní kódu[1].

Nesnaží se přebírat konvence z klasických programovacích jazyků a naopak zavádí své vlastní konvence. Je navržen tak, aby předcházel chybám programátora a jeho notace spíše připomíná matematický zápis, který je velmi blízký vědcům bez programátorského vzdělání.

V jazyce Python lze programy psát s ohledem na objektově orientovaný návrh (podobně jako jazyk Java), ale i strukturovaně (podobně jako v jazyce C), odstraňuje potřebu psát ukončovací znak za každým příkazem a úplně odstraňuje typovou kontrolu proměnných. Python je uvolněn pod svobodnou licencí kompatibilní s GPL licencí - lze jej tedy využít i v komerčních projektech.

Právě díky těmto vlastnostem vznikají nové projekty, které si berou za úkol poskytnout prostředí podobné MATLABU založené na tomto programovacím jazyce. Jmenovitě se jedná například o projekty IPython[01], Numpy[02], SciPy[03] a ScientificPython[04].

1 Například ukončovací znak za příkazy (středník) byl nahrazen samotným ukončením řádku a zanoření kódu je řízeno odsazením (klasicky 4 mezery) místo používání závorek.

2.4 Integrace jazyků

Pokud má být provedena jakákoliv migrace stávajících řešení na katedře KIV, musí existovat možnost kombinace Javy, Matlabu a Pythonu, nebo plnohodnotná náhrada již existujících implementací.

Konkrétní nalezená řešení a odkazy na oficiální stránky projektů lze nalézt v příloze B.

2.4.1 Java a MATLAB

Prostředí MATLAB nativně podporuje Javu, protože některé jeho části ji využívají. Je možné volat nativní metody přímo z Javy, spouštět přeložené Java Class soubory nebo spouštět vlastní programy vytvořené v jazyce Java.

Společnost MathWorks nabízí komerční řešení, které přeloží libovolný MATLAB program do byte kódu pro JVM[2]. Existují i nekomerční řešení, které předávají zprávy (výsledky) z prostředí MATLAB do JVM.

2.4.2 Java a Python

Java dokáže spouštět nativně Python kód, pokud jej necháme přeložit JPython interpretem. Bohužel tato možnost znemožňuje použití většiny Python knihoven, a proto je nevhodná pro neuroinformatické experimenty.

Jazyk Python nedokáže spouštět kód jazyka Java a ani neexistují dokončené projekty zajišťující tuto funkci.

2.4.3 Python a MATLAB

Python a MATLAB jsou velmi rozdílné programovací nástroje. Existují projekty, které se snaží nahradit kompletně MATLAB jazykem Python a dodatečnými knihovnami, ale žádný, který by chtěl sjednotit Python a MATLAB dohromady.

2 Java Virtual Machine – překládá byte kód pro konkrétní platformu a spouští daný kód.

3 Projekty v neuroinformatice

Bylo provedeno hledání projektů z domény neuroinformatiky, jehož cílem bylo zjistit, zda a v jaké míře se vyskytují programy a projekty v oblasti neuroinformatiky používající jazyk Python.

3.1 Oblasti hledání

Hledání probíhalo primárně na stránkách www.incf.org, které obsahují katalog dostupného softwaru nejen pro EEG[3]/ERP[4] experimenty. Sekundárně pak na internetových vyhledávačích (google.com), repozitářích softwaru (github.com, sourceforge.net) i v katalozích zahraničních univerzit (eeg.pl, bci.tugraz.at).

3.2 Metodika hledání

V první části byl vytvořen seznam všech projektů a programů, které alespoň rámcově odpovídaly zaměření do libovolné oblasti neuroinformatiky. V druhé části průzkumu byly jednotlivé projekty posuzovány, zda jsou relevantní vůči výzkumu EEG/ERP probíhajícím na ZČU.

Byly **vyřazeny** všechny projekty, které:

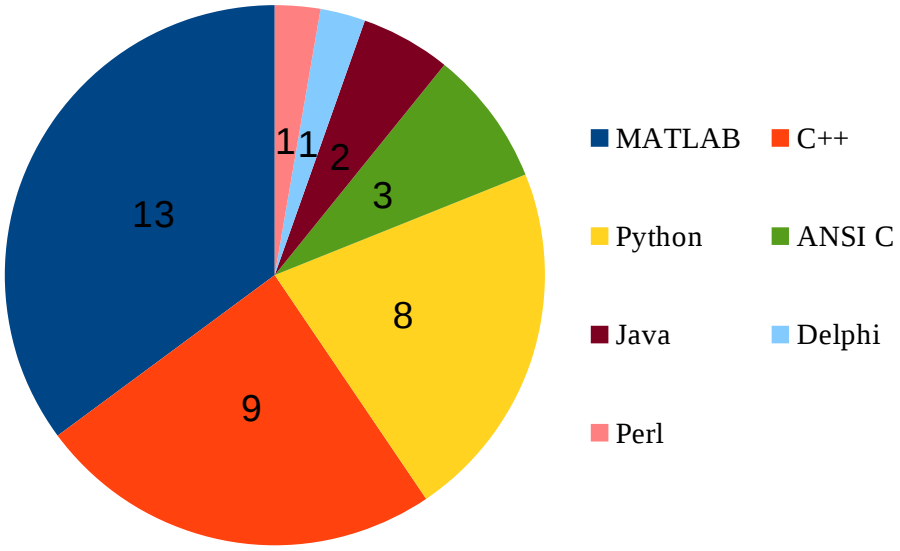
- Neměly nic společného s informatikou.
- Prováděly experimenty na neživých subjektech.
- Prováděly invazivní měření.
- Používaly vybavení nedostupné na ZČU (například magnetická rezonance).
- Nebyly dokončené.

3 Elektroencefalogram (EEG) je záznam časové změny elektrického potenciálu způsobeného mozkovou aktivitou.

4 Event-related potential (ERP) je naměřená mozková reakce (pomocí EEG) vyvolaná podnětem například pomocí smyslových vjemů.

3.3 Výsledek průzkumu

Detailní výsledky (seznam nalezených projektů s odkazem a krátkým popisem) je k dispozici v příloze A. Zjištěné četnosti jsou pro větší přehlednost vyobrazené v obrázku 3.1.



3.1. Obrázek: Graf četností použití programovacích jazyků v neuroinformatických projektech (zdroj: vlastní průzkum)

Ze zjištěných četností vyplývá, že Java se prakticky (kromě prostředí ZČU) v doméně neuroinformatiky nepoužívá. Nejpoužívanější bylo očekávaně prostředí MATLAB, a na druhém místě se umístil velmi rychlý a složitý programovací jazyk C++. Python se umístil na třetím místě. Nicméně počet Python projektů se zvýší, neboť několik Python projektů bylo v době průzkumu teprve ve fázi plánování, čili nebyly do výsledků zahrnuty.

Tento průzkum ukázal, že má smysl zabývat se otázkou migrace k jazyku Python, alespoň v doméně neuroinformatiky. Již teď existuje několik funkčních řešení dostupných v Pythonu, která nejsou bohužel kompatibilní ani s Javou, ani s prostředím MATLAB.

3.4 Zajímavé nalezené projekty

Při průzkumu byly objeveny nové projekty týkající se výzkumu neuroinformatiky a mohou být využity při EEG/ERP experimentech.

3.4.1 OpenBCI

OpenBCI je systém napsaný v jazyce Python. Byl navržen v rámci projektu pomoci pacientům, kteří nemohou komunikovat se svým okolím z vážných zdravotních důvodů. OpenBCI obsahuje kompletní systém BCI[5] pro EEG.

http://bci.fuw.edu.pl/

3.4.2 NeuroTools

NeuroTools je balík knihoven napsaných v jazyce Python určený především pro modelování a následnou simulaci složitých neuronových sítí včetně srozumitelné vizualizace pro interpretaci výsledků.

http://neuralensemble.org/NeuroTools/

3.4.3 BrainVoyager QX

BrainVoyager QX je několik komerčních programů napsaných v jazyce C++ věnujících se analýze dat především z magnetické rezonance.

http://www.brainvoyager.com/products/brainvoyagerqx.html

3.4.4 NeuroDebian

NeuroDebian je upravená distribuce Linuxu Debian. Obsahuje řadu předinstalovaných programů pro doménu neuroinformatiky. Také nabízí repozitáře pro jiné distribuce založené na Linuxu Debian obsahující přeložené balíčky nejpoužívanějších programů v rámci neuroinformatiky.

http://neuro.debian.net/

5 Brain–computer interface: Ovládání počítače pomocí myšlenek.

4 Existující implementace algoritmů

Konkrétní implementace zadaných algoritmů v jazycích Java a MATLAB jsou již zpracovány v práci Tomáše Prokopa [05]. Následuje výčet nalezených implementací algoritmů používaných v doméně neuroinformatiky v jazyce Python.

4.1 Lineární diskriminační analýza (LDA)

4.1.1 PyChem

PyChem je modul knihoven pro jazyk Python. Obsahuje nástroje pro analýzu obecných vícerozměrných dat (například signálů). Obsahuje algoritmy pro analýzu rozptylu, spektrální zpracování signálu, PCA[6], shlukovou analýzu a PLS[7].

http://pychem.sourceforge.net/

4.1.2 ALGLIB

ALGLIB je multiplatformní knihovna pro numerickou analýzu dat. Je dostupná i pro jazyk Python. Obsahuje nástroje pro statistiku, lineární algebru, řešení rovnic, interpolaci a matematickou optimalizaci.

http://www.alglib.net/dataanalysis/lineardiscriminantanalysis.php

4.1.3 MDP

Modular toolkit for **D**ata **P**rocessing je sada knihoven pro jazyk Python. Obsahuje především implementaci algoritmů pro zpracování signálů, jmenovitě například: PCA, ICA[8] a klasifikátory.

http://mdp-toolkit.sourceforge.net/

6 Principal components analysis (PCA)
7 Partial least squares (PLS) regression
8 Independent Component Analysis (ICA)

4.1.4 scikits-learn

Je balík knihoven pro jazyk Python. Obsahuje především implementované algoritmy strojového učení s učitelem i bez: GPML[9], Naive Bayes, Gaussian mixture models, shluková analýza.

http://scikit-learn.org/dev/auto_examples/plot_lda_qda.html

4.2 Fourierova transformace (FFT)

FFT je nativně obsažena v balíku SciPy. SciPy je knihovna poskytující základní i pokročilé matematické algoritmy pro jazyk Python.

http://docs.scipy.org/doc/scipy/reference/fftpack.html

http://docs.scipy.org/doc/numpy/reference/routines.fft.html

4.3 Waveletová transformace (WT)

4.3.1 PyWavelets

Knihovna pro práci s WT, napsáno v Pythonu/Cythonu s využitím knihovny NumPy.

http://www.pybytes.com/pywavelets/

4.4 Principal Component Analysis (PCA)

4.4.1 MDP

Modular toolkit for **D**ata **P**rocessing je sada knihoven pro jazyk Python. Obsahuje především implementaci algoritmů pro zpracování signálů.

http://mdp-toolkit.sourceforge.net/

9 Gaussian Processes for Machine Learning (GPML)

4.4.2 scikits-learn

Je balík knihoven postavených nad SciPy. Obsahuje především implementované algoritmy strojového učení.

http://scikit-learn.org/0.6/modules/generated/scikits.learn.pca.PCA.html

4.5 Hilbert-Huangova transformace (HHT)

4.5.1 PyHHT

Implementace HHT pomocí balíku SciPy a NumPy.

https://github.com/jaidevd/pyhht

4.6 Umělá neuronová síť

Je uveden pouze seznam vybraných implementací. Všechny tyto projekty využívají knihovny SciPy a NumPy:

- NeuroLab http://code.google.com/p/neurolab/
- PyNeurGen http://pyneurgen.sourceforge.net/
- PyBrain http://pybrain.org/

4.7 Independent Component Analysis (ICA)

4.7.1 MDP

Modular toolkit for **D**ata **P**rocessing je sada knihoven pro jazyk Python. Obsahuje především implementaci algoritmů pro zpracování signálů.

http://pypi.python.org/pypi/MDP/2.5

4.8 Finite impulse response (FIR)

4.8.1 FIR filter design

Konkrétní implementace pomocí SciPy je popsána na:

http://mpastell.com/2010/01/18/fir-with-scipy/

4.8.2 Nitime

Knihovna nabízející řadu algoritmů pro zpracování signálů z neuroinformatických experimentů například: filtrování, normalizaci a analýzu fMRI[10].

http://nipy.sourceforge.net/nitime/

10 Event-related functional magnetic resonance imaging (fMRI) .

5 Matching pursuit (MP)

Tento algoritmus byl vybrán supervizorem k porovnávání výkonnosti jazyků Java, C++ a Python. MP se používá v široké škále informatických disciplín [06]. Slouží především jako most mezi algoritmy pro zpracování obrazu, zvuku nebo videa (obecně libovolného spojitého analogového signálu) a samotným analogovým signálem. Proto existuje velké množství jeho implementací.

5.1 Obecný popis MP

Algoritmus MP obecně nahrazuje navzorkovaný analogový signál množinou funkcí, které v součtu aproximují daný signál s minimální chybou [07][08]. Dané funkce se vybírají buď z předem připraveného slovníku (suboptimální řešení), nebo se generují přímo při běhu samotného algoritmu (optimální řešení). Původní navzorkovaný signál *y(x)* se tedy dá vyjádřit jako lineární kombinace vybraných funkcí g_n ze slovníku:

$$y(x) \approx \sum_{n=1}^{N} (a_n \cdot g_n) \tag{5.1}$$

O výsledku MP lze také hovořit jako o optimální ztrátové kompresi analogového signálu. Například MP při použití Gaborových atomů dokáže zpracovat 256 vzorků (představující 256 desetinných čísel) a uložit je jako 10 Gaborových atomů (40 desetinných čísel), což představuje 84% kompresi při zachování průběhu funkce v rámci přijatelné chyby.

5.1.1 Slovník funkcí

Pokud se slovník generuje přímo během výpočtu, algoritmus dosahuje obecně lepších výsledků za cenu prodloužení doby výpočtu. Statický, předem zvolený slovník dosahuje horších výsledků v kratší době a lze pro něj MP algoritmus optimalizovat v mnohem větší míře.

Nelze zvolit obecně slovník tak, aby vyhovoval všem druhům signálů. Proto se musí zvolit funkce, ze kterých se vypočítávají konkrétní funkce (dosazením parametrů) – atomy, které tvoří samotný slovník. Pokud jsou tyto funkce navzájem ortogonální, pak se koeficient a_n vypočítá jako skalární součin daného atomu a signálu.

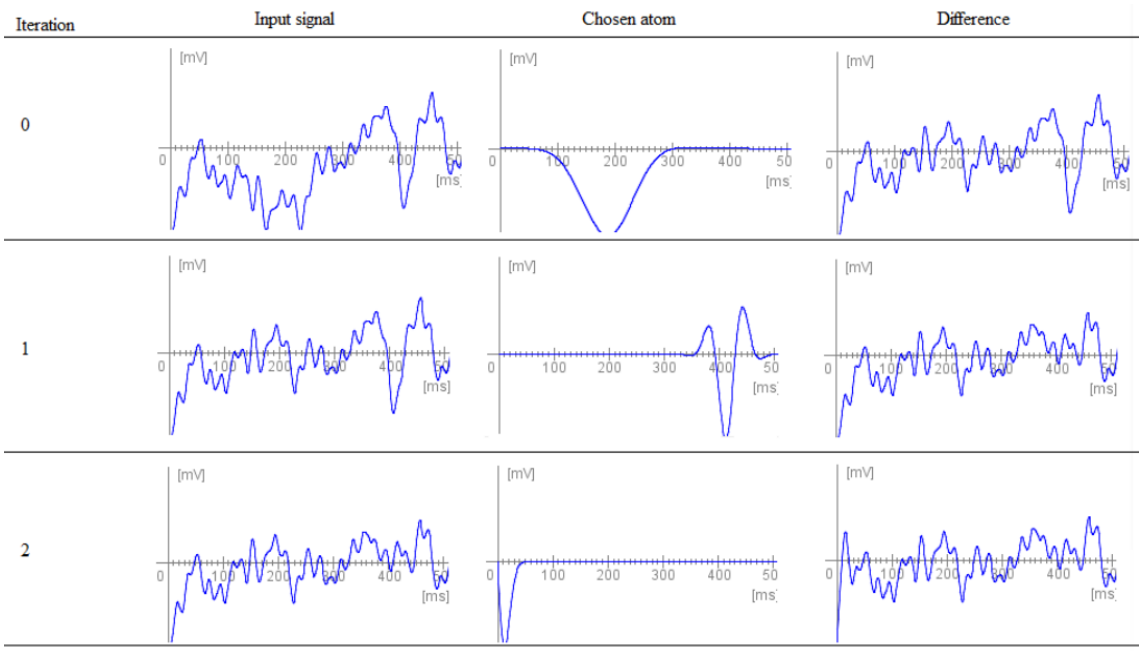
Zvolená generující funkce může generovat například Gaborovy atomy, Waveletové funkce, nebo jakékoliv jiné funkce. Slovník se vždy volí podle typu úlohy.

5.1.2 Iterace

Jakmile algoritmus zvolí konkrétní atom, musí se vypočítat rozdíl mezi signálem a zvoleným atomem. Naivní implementace algoritmu by vyzkoušela všechny kombinace atomů až do počtu *N* iterací, a pak by zvolila optimální kombinaci atomů, které by v součtu nahradily signál s minimální chybou:

$$\varepsilon = \left\| y - \sum_{n=1}^{N} (a_n \cdot g_n) \right\| = min \tag{5.2}$$

Bohužel, běh této naivní implementace by trval neúměrně dlouho. Proto se obecně používá iterační přístup, který v každé iteraci vybere nejlepší možný atom, ten odečte od signálu a pokračuje další iterací. Tento typ strategie se nazývá greedy algoritmus a získá suboptimální řešení za kratší dobu výpočtu.



5.1. Obrázek: Znázornění jednotlivých iterací MP (zdroj: [07, str.: 36])

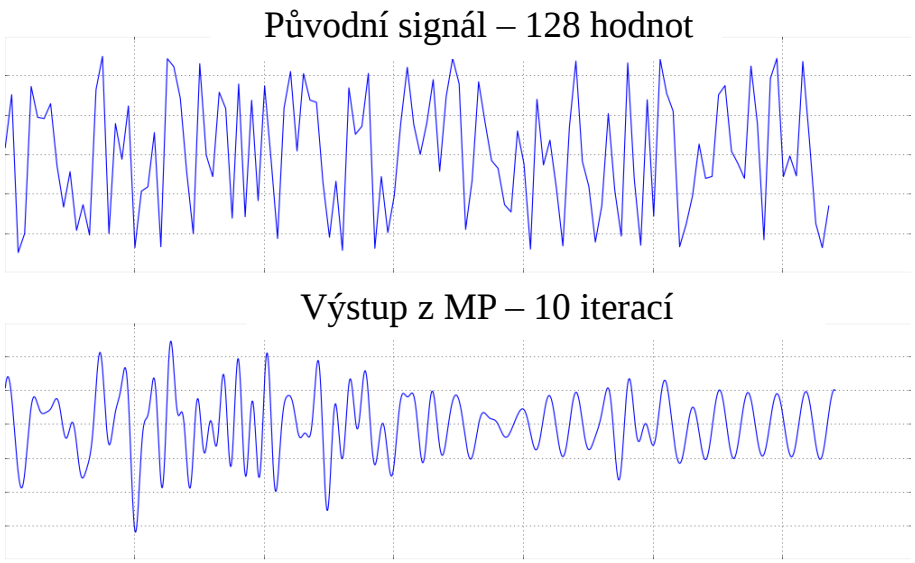
Na obrázku 5.1 je zobrazen průběh prvních tří iterací algoritmu MP: Vlevo je obraz původní funkce, uprostřed je znázorněn vybraný atom ze slovníku a vpravo je výsledný průběh nové funkce. S každou další iterací se zmenšuje chyba aproximace, a tím se zvyšuje přesnost výsledku.

5.1.3 Výsledek MP

Algoritmus MP vypočte ze zadaného signálu maximálně *N* vektorů definující atomy, které byly použity při aproximaci signálu. Počet iterací udává maximální počet atomů, nikoliv minimum. Při každé iteraci se kontroluje, zda vnitřní energie nového signálu je nižší než předchozí. Pokud tomu tak není, nový atom zhoršil výsledek. Tato situace nastává typicky pro uměle vytvořené signály (lineární průběh) nebo pro vysoký počet iterací. V takovém případě se výpočet přeruší a algoritmus předá menší počet atomů jako výsledek.

Atom je definován jako vektor reálných čísel. Počet těchto čísel je určen podle zvoleného slovníku (počet parametrů nutných pro výpočet atomu).

Samotný výsledek se musí dále zpracovávat dalšími algoritmy, které díky MP mohou pracovat se signálem mnohem efektivněji. Pro ověření správnosti výpočtu se musí jednotlivé atomy sečíst a zobrazit ve stejném měřítku jako původní signál v grafu. Ty musí být vizuálně podobné jako na obrázku 5.2:



5.2. Obrázek: Ukázka vizualizace výstupu MP (zdroj: vlastní výpočty)

5.2 Implementace MP na ZČU

Pro věrohodné porovnání algoritmů musí být jejich implementace totožná. Proto implementace MP v Pythonu musí striktně vycházet z popisu implementace uvedené v tézi Tomáše Řondíka [07, str. 34-36] a článku [09].

5.2.1 Popis funkce MP

Vstupní data představují navzorkovaný analogový spojitý signál naměřený při EEG/ERP experimentu, který se musí nahradit lineární kombinací Gaborových atomů tak, aby jejich výsledná suma odpovídala pokud možno co nejvíce původnímu signálu.

Výstup algoritmu se dále zpracovává dalšími algoritmy, například Wigner-Villova transformace [07, str. 36], která umožní snadnou analýzu výstupu MP pouhým pozorováním grafu.

Algoritmus pracuje iteračně (popsáno v kapitole 5.1), slovník je neměnný a je tvořen Gaborovými atomy. V každém kroku vybere nejvíce významnou část signálu a tu se pokusí nahradit nejlepší možnou variantou ze slovníku Gaborových atomů.

5.2.2 Gaborovy atomy

Gaborovy atomy jsou definovány [09, str. 338] jako Gaussovo okno (Gaussian Window):

$$g(t) = \mathrm{e}^{-\pi t^2} \tag{5.3}$$

Modulováno přepisem:

$$g_{(s,u,v,w)}(t) = g\left(\frac{t-u}{s}\right)\cdot\cos(vt+w) \tag{5.4}$$

Kde parametry *s, u, v, w* představují měřítko, posun, frekvenci a fázový posun a tím jednoznačně definují Gaborův atom jako vektor (*s, u, v, w).*

5.2.3 Urychlení výpočtu

Pro urychlení výpočtu používá MP Rychlou Fourierovu transformaci (FFT). Ta urychluje výběr optimálního atomu pro jednotlivé iterace. Zároveň však klade dodatečný požadavek na délku vstupních dat: počet vzorků signálu musí být libovolná mocnina čísla dva. Díky tomu je asymptotická složitost algoritmu [09, str. 341]:

$$O\left(n \cdot \log_2(n)^2\right) \tag{5.5}$$

5.3 Nalezené MP implementace v Pythonu

Všechny nalezené implementace MP jsou velmi odlišné od ZČU implementace, a proto nejsou vhodné k porovnání. Z tohoto důvodu bude implementován algoritmus MP v Pythonu v rámci praktické části.

5.3.1 Py-pursuit

Tato implementace nepoužívá Gaborovy atomy. Jedná se spíše o zobecněný PCA algoritmus.

https://github.com/lmjohns3/py-pursuit

5.3.2 SciKit

SciKit je balík knihoven postavených nad SciPy a určený pro vědecké experimenty. Obsahuje nativně implementaci ortogonálního MP algoritmu. Ten je založený na K-SVD[11] Batch Orthogonal MP. Tento algoritmus si buduje vlastní slovník ze zadaných dat, a proto se jedná o jinou implementaci.

http://scikit-learn.org/dev/modules/generated/sklearn.linear_model.orthogonal_mp.html

11 K-SVD je efektivní algoritmus pro vytváření slovníku učením se ze zadaných signálů.

5.3.3 OMP blog.vene.ro

Na stránkách studenta Nicuale [10], který aktivně přispívá do balíku funkcí pro Python scikit-learn je popsána implementace algoritmu MP. Tato implementace opět používá K-SVD Batch Orthogonal MP, čili není vhodná pro porovnávání s ZČU verzí.

http://blog.vene.ro/2011/08/07/optimizing-orthogonal-matching-pursuit-code-in-numpy-part-1/

http://blog.vene.ro/2011/08/11/optimizing-orthogonal-matching-pursuit-code-in-numpy-part-2/

6 Implementace MP v Pythonu

Implementoval jsem algoritmus MP v jazyce Python v několika verzích pro porovnání výkonnostních parametrů. V rámci praktické části Bakalářské práce jsem vyzkoušel všechny smysluplné kombinace knihoven, interpretů a překladačů.

6.1 Python interpret

Jazyk Python je skriptovací jazyk. Pro jeho běh musí mít uživatel nainstalovaný konkrétní interpret pro daný operační systém. Výběr interpretu výrazně ovlivňuje možnost použití knihoven, výkonnost a přenositelnost programu mezi různými operačními systémy.

Zdrojové soubory mají koncovku *.py* a při prvním spuštění se z nich vytvoří bytecode *.pyc soubory*, které se spouštějí v interpretu. Ten se chová jako VM[12]. Díky tomuto návrhu jazyka Python vzniklo hned několik interpretů.

6.1.1 CPython

CPython je výchozí interpret jazyka Python. Jedná se o implementaci v jazyce C. Nachází se ve dvou verzích: dlouhodobě podporovaná verze 2.7 a poslední stabilní doporučovaná verze pro nové projekty 3.3.

CPython je doporučován především díky velkému počtu knihoven rozšiřujících jeho vlastnosti, v rámci MP především knihovny SciPy a NumPy. Vzhledem k jeho rozšířenosti a plné podpoře v mnoha neuroinformatických projektech bude použit při výkonnostním testu.

6.1.2 PyPy

Tento interpret je hlavní alternativou k CPythonu. Jedná se o JIT[13] interpret, který již při překladu určuje typy proměnných a tím výrazně urychluje dobu běhu programu. Zároveň umožňuje spouštět Python programy v stackless[14] módu, což umožňuje psát aplikace masivně využívající paralelní běh na více procesorech.

PyPy je ale stále ve vývoji a nepodporuje řadu matematických knihoven dostupných pro Python, a proto nebude zahrnut do testu.

12 Virtual Machine: Virtuální stroj, hlídá především alokaci paměti pro běh programů.
13 Just-in-time kompilace: Přednačítá instrukce bytecode a rovnou je překládá do instrukcí pro procesor.
14 Vlákna nemají v paměti žádný zásobník pro statické proměnné - šetří použitou paměť.

6.1.3 Psyco

Funguje jako JIT překladač pro CPython a podporuje i Python knihovny. Při spuštění překládá načtený kód v paměti do strojového kódu založeného na jazyce C. Bohužel podporuje pouze x86 architekturu a zastaralou verzi Pythonu 2.5. Tento projekt je navíc neudržovaný, a proto nebude zahrnut v testu.

6.1.4 Stackless Python

Jedná se o alternativní interpret, který je plně kompatibilní s CPython interpretem včetně knihoven[15]. Tato verze řeší výkonnostní problémy klasického interpretu ve vícevláknových programech.

Jelikož MP nelze nijak zásadně urychlit použitím vláken, nebude tento interpret použit.

6.1.5 Jython

Implementace jazyka Python v JVM[16]. Dokáže spouštět knihovny určené pro Javu, ale nenabízí žádnou přidanou hodnotu. Není kompatibilní s Python knihovnami. Mírně upravuje syntaxi původního jazyka Python.

MP je již implementován v Javě, a tudíž nebude použit tento interpret.

6.1.6 IronPython

Jedná se o implementaci jazyka Python v rodině jazyků .NET, především v jazyce C#. Tento interpret umožňuje provázat Python kód s ostatními jazyky .NET, ale není kompatibilní s Python knihovnami, a proto nebude použit v testu.

6.1.7 Cython

Tento překladač překládá Python kód přímo do jazyka C. Ten se pak překládá standardními překladači C do strojového kódu a tím zlepšuje výkon daného programu. Je plně kompatibilní s jazykem Python a jeho knihovnami. Zároveň volitelně zavádí dodatečnou typovou kontrolu a tím umožňuje vyšší optimalizaci kódu.

Tento překladač bude použit v testu.

15 Kromě knihoven používající knihovnu SIP – naprosté minimum.
16 Java Virtual Machine.

6.2 Matematické knihovny

Pro rychlý vývoj experimentálních aplikací v oblasti neuroinformatiky je nutný výskyt knihoven obsahující nejpoužívanější matematické funkce. Pro tyto účely vznikly projekty, které dohromady nahrazují funkčnost MATLABU.

V rámci implementace MP byla použita funkce FFT[17] z knihovny NumPy.

6.2.1 NumPy

Numpy je kompatibilní Python knihovna určená pro numerické výpočty a operace nad velkými poli čísel nebo matic. Numpy knihovna je napsaná v jazyce C a nabízí podstatně větší výkon oproti Pythonu v konkrétních situacích.

Numpy zavádí podobný přístup k matematickým operacím jako jazyk Fortran: Pracuje s velkou množinou dat a malým počtem operací, například nabízí základní matematické operace, které jsou schopné pracovat s poli a maticemi. Tento přístup obchází nutnost používat cykly v kódu a nabízí větší výpočetní výkon než konvenční cyklus.

Spolu s jednoduchostí jazyka Python se zdrojový kód podobá matematickému zápisu, což vysvětluje jeho rostoucí popularitu na úkor MATLABU nebo jazyka C.

6.2.2 SciPy

SciPy je knihovna vycházející z NumPy, která dodává pokročilejší matematické funkce. Dané funkce jsou optimalizovány v jazyce C a jsou plně kompatibilní s jazykem Python.

6.2.3 matplotlib

Matplotlib je knihovna poskytující pohodlné vykreslování grafů. Tato knihovna je velmi užitečná pro vykreslování grafů a následnou analýzu dat. Nabízí modul pylab, který umožňuje vykreslit graf pomocí dvojice příkazů:

```
plot(x,y)
show()
```

17 Fast Fourier transform: efektivní algoritmus pro výpočet diskrétní Fourierovy transformace (DFT).

6.3 Python Optimalizace

Byly implementovány dvě verze MP. První verze využívající výchozí Python interpret CPython a druhá verze optimalizovaná pro překladač Cython. Obě verze využívají matematickou knihovnu NumPy a byly optimalizovány.

6.3.1 Typová kontrola

Python jazyk nedeklaruje typ proměnných při jejich založení, ale určuje jejich typ až při samotném běhu programu.

Experimentálně bylo zjištěno, že přetypování je velmi drahá operace v Pythonu. Pokud nějaké číslo bylo v algoritmu využíváno jako float i int, nejlepším řešením bylo uložit dané číslo do dvou proměnných a ty udržovat odděleně.

Dalším problémem bylo celočíselné dělení: pokud během výpočtu libovolné reálné číslo bylo bez desetinných míst, uložilo se v paměti jako integer a pak vznikaly numerické chyby[18]. Proto se u každé operace dělení musí dané číslo označit jako float.

6.3.2 Moduly math a numpy

Modul numpy je optimalizován pro vektorové operace, ale funguje i na skaláry. Nicméně bylo experimentálně ověřeno, že pro operace se skaláry je mnohem výhodnější použít modul math (pokud danou operaci obsahuje), včetně matematických konstant.

Další výkonnostní omezení bylo zaznamenáno u vektorových funkcí numpy, pokud se správně nepoužijí všechny parametry konkrétní funkce:

```
numpy.multiply(x1, x2[, out])
```

Funkce multiply vynásobí libovolný skalár, vektor či matici. Ovšem při každém zavolání této funkce se v paměti alokuje nové pole (matice) k uložení výsledku. Proto je vhodné alokovat místo na výsledek předem (nejlépe před zanořením do cyklů) a ten předat jako třetí nepovinný parametr.

18 Tato vlastnost je v Pythonu 3 odstraněna: Celočíselné dělení se nově značí dvojitým lomítkem //.

6.3.3 Pole nebo seznamy

Python neobsahuje klasická pole a místo nich doporučuje použít seznamy. Bohužel tyto seznamy jsou pro matematické výpočty velmi nevhodné z důvodu vysoké paměťové náročnosti a absenci přístupu ke konkrétnímu prvku v čase O(1).

Tento problém řeší knihovna numpy, která implementuje chybějící pole. Pro alokaci paměti na nový vektor nebo matici se volají funkce **zeros**, **ones** a **empty** z modulu numpy. První parametr udává tvar nového pole. Druhý povinný parametr udává typ proměnné a číslo udává počet bitů pro uložení dané proměnné do paměti. Funkce zeros vyplní nové pole nulovou hodnotou, ones hodnotou jedna a empty nechá dané pole neinicializované[19].

Tento příkaz alokuje prázdnou matici 5 prvků širokou a 3 prvky vysokou, každý prvek je uložen jako 8 bitové číslo integer. Samotná matice může mít libovolný počet dimenzí.

```
prazdna_matice_5x3 = numpy.zeros((5, 3), numpy.int8)
```

Tento příkaz alokuje vektor o pěti prvcích, každý prvek má hodnotu 1.0 a je uložen jako 32 bitové float číslo.

```
vektor_5prvku = numpy.ones(5, numpy.float32)
```

Příkaz **array** slouží k převodu datových Python typů[20] na pole z důvodu rychlosti následujících výpočtů. Lze použít i pro inicializaci polí konkrétních hodnot. Tento příkaz alokuje místo pro vektor o 3 prvcích. Každý prvek je uložen jako imaginární číslo uložené jako dvojice float čísel s přesností na 64 bitů. Imaginární číslo se zapisuje jako suma reálné složky a imaginární složky označené písmenem *j*.

```
com = numpy.array([2 + 1j, 9.3 - 3j, 6j], numpy.complex128)
```

6.3.4 Inicializace proměnných

V mnoha programovacích jazycích nezáleží, na kterém místě v kódu programu se založí pomocné proměnné, jelikož jsou následně zpracovány překladači, které provedou optimalizaci. Bohužel v Pythonu obecně záleží na tom, zda se daná proměnná inicializuje v každé iteraci cyklu, nebo pouze jednou před započetím cyklu.

19 Jednotlivé hodnoty prvků jsou závislé na místě v paměti a obsahu dané paměti před tím, než byla alokována pro interpret jazyka Python – jsou náhodné.

20 Především slovníky a seznamy.

Proto lze výkon aplikace zlepšit díky včasné inicializaci všech proměnných a do nich dosadit datové typy tak, aby nedocházelo k přetypování dále v průběhu programu.

6.3.5 Použití polí místo tříd

Python podporuje objektově orientované paradigma, a proto se nabízí použít k uložení Gaborova atomu (4 reálné čísla) třídu. Bohužel tento přístup může výrazně zpomalit výpočet, protože při použití tříd musí interpret při každém přístupu nejprve řešit přístup k objektu (třídě) a pak teprve k hodnotě samotné.

Proto se pro uložení atomu využívá pole a přístup k jednotlivým prvkům Gaborova atomu přes indexy uložené jako globální konstanty.

6.4 Cython optimalizace

Cython překladač dokáže přeložit kód Pythonu bez zásahu programátora do kódu jazyka C. Ten se následně přeloží jako statická knihovna klasickým překladačem jazyka C. Výsledná knihovna se pak použije stejně jako například vestavěná knihovna numpy, která je také přeložena do strojového jazyka pro danou platformu.

Proto se vyplatí překládat překladačem Cython pouze moduly, ve kterých tráví program nejvíce času. V případě MP se jedná o moduly MP_original.py a gauss_window.py.

6.4.1 Překlad

Samotný překlad sestává ze dvou fází: překlad zdrojových Cython souborů pyx na zdrojový kód:

```
cython jmeno_souboru.pyx
```

A následně překlad překladačem C na sdílenou knihovnu pro danou platformu:

```
gcc jmeno_souboru.c -fPIC -fwrapv -O3 -I
/usr/include/python2.7/ -o knihovna.so
```

Nejdůležitější přepínač je fPIC, který generuje pozičně nezávislý kód. Pokud není uveden, daná knihovna bude funkční pouze na počítači, na kterém byla přeložena.

Pokud se neupraví původní kód z Pythonu, k jediné optimalizaci dojde při překladu překladačem gcc díky přepínači -O3. Ten zajistí zrychlení samotného kódu průměrně o 30% [11].

6.4.2 Zavedení typů proměnných

K značnému urychlení výpočtu stačí u všech proměnných uvést jejich typ zápisem:

```
cdef int cislo1 = 5

cdef numpy.float64_t cislo2 = 3.2
```

Díky této přidané notaci kód ztrácí kompatibilitu s klasickým jazykem Python.

6.4.3 Python a C funkce

Cython rozlišuje mezi dvěma druhy funkcí. Python funkce vrací libovolnou hodnotu a může mít nadefinovány typy vstupních parametrů:

```
def apply_gauss_window1(numpy.float64_t param):

def apply_gauss_window2(libovolny_param):
```

Python funkce může být volána jak z Python kódu, tak z C funkcí. Naproti tomu C funkce má definovaný typ návratové hodnoty a zároveň musí mít nadefinované i typy vstupních parametrů:

```
cdef int apply_gauss_window3(numpy.float64 param):
```

C funkce mohou být volány pouze z jiných C funkcí a z přeložených modulů Cythonem. Proto funkce apply_gauss_window3 nemůže být zavolána přímo interpretem Python. Díky přidané informaci o očekávaných typech proměnných generuje Cython mnohem výkonnější kód.

6.4.4 Použití systémových knihoven

Python je přenosný jazyk, a proto i jeho knihovny musí být univerzální. Bohužel za cenu sníženého výkonu. Stejně tak i numpy[21]. Ale jazyk C má k dispozici systémové knihovny pro některé matematické výpočty poskytované operačním systémem. Jmenovitě se jedná o knihovnu math obsaženou standardně v knihovně stdlibc. Tu můžete naimportovat do Cythonu příkazem:

```
cimport libc.math as math
```

Stejným postupem můžete importovat libovolné knihovny, které jsou dostupné z jazyka C. Pokusné měření ukázalo, že například funkce math.sin() je rychlejší z knihovny jazyka C, než z jazyka Python modulu math.

6.4.5 Jazyk Cython

Cython není zpětně kompatibilní s jazykem Python. Jeho zápis je velice chaotický, zmatený a většinu operací by mohl překladač zvládat automaticky. Projekt se nachází ve fázi beta. Kód jako takový nepřináší žádnou přidanou hodnotu: programátor se musí naučit další jazyk, kterým následně generuje jazyk C pro překladač. Mnohem jednodušší je napsat výkonný kód rovnou v C a ten pak pouze zavolat přes Python interpret.

Z těchto důvodů nebyla implementace všech optimalizací Cythonu dokončena. Optimalizovaný Cython kód vygeneruje C kód, který poběží stejně rychle jako implementace MP v jazyce C++.

21 Numpy využívá sofistikovaný software pro algebru ATLAS. Pokud se Numpy nezkompiluje při instalaci, pak nedosahuje maximální výkonnosti, který tento software nabízí.

7 Porovnání výkonu MP

Pro porovnání výkonnosti jednotlivých programovacích jazyků byly otestovány totožné implementace MP v jazycích C++, Java a Python. V rámci testu se měřila pouze čistá doba výpočtu bez I/O operací na systému:

Procesor Intel Core i3	(2,26GHz, 2 jádra)
4 GB RAM	(DDR3)
Linux MINT 13	(3.2.0-39-generic-pae, architektura i686)

MATLAB implementace MP nebyla zahrnuta, neboť je implementována pouze částečně a vyžaduje implementovat slovník hledaných atomů, což je mimo rozsah této práce.

7.1 Java

Algoritmus MP v jazyce Java byl implementován Tomášem Řondíkem[Chyba: zdroj odkazu nenalezen]. Byly testovány dvě verze JVM od společnosti Oracle především z důvodu jejich rozšířenosti.

- Oracle Java SE 6
- Oracle Java SE 7

Nejlepších výsledků dosáhla Oracle Java SE 6. Oracle Java SE 7 dosahuje horších výsledků v řádek jednotek procent. Testování bylo provedeno s ohledem na JIT kompilátor, proto samotná metoda MP byla volána v cyklu a tím se docílilo plné využití JIT kompilátoru. Bez tohoto postupu by byly naměřené hodnoty zkreslené[22].

22 Například MP pro 64 prvků by trval delší dobu, než pro 256 prvků.

7.2 C++

MP implementovaný v C++ obsahoval dvě verze MP. Testována byla verze využívající Gaborovy atomy. Dosahuje stejných výsledků a jedná se o stejný algoritmus jako u verze Javy. Zdrojový kód byl přeložen překladači:

- Linux: g++ 4.6.3
- Windows: MinGW 20120426

Výkon je totožný v rámci chyby měření na platformě Windows i Linux. Proto v tabulce výsledků 1 jsou zaznamenány pouze výsledky z platformy Linux.

7.3 CPython

CPython verze byla testována na dvou verzích klasického Python interpretu:

- Python 2.7.3
- Python 3.3.0

7.4 Cython

Moduly gauss_window a MP_original byly přeloženy do jazyka C pomocí překladače Cython verze 0.18 a následně zkompilovány překladačem GCC 4.6.3 na statické knihovny. Ty pak byly použity místo původních Python modulů při testu.

7.5 Výsledek měření

Časy v tabulce 1 jsou v milisekundách. Množství dat je odstupňováno podle mocnin čísla dva z důvodů uvedených v kapitole 5.2.3.

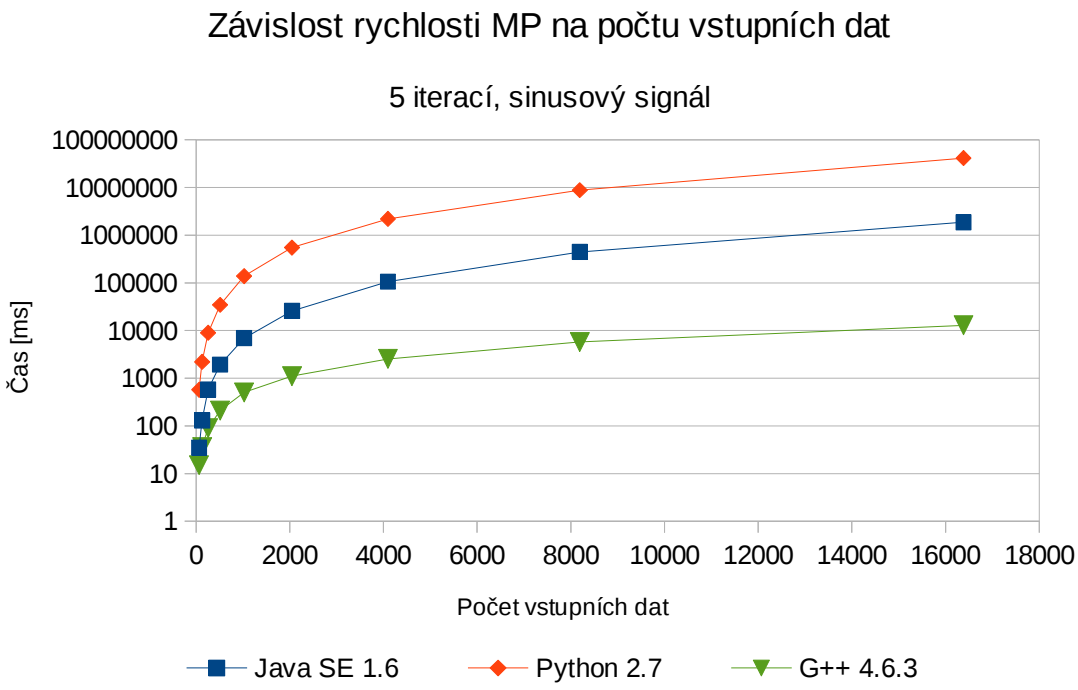
Počet vstupních dat	Java SE 1.6	Java SE 1.7	Python 2.7	Python 3.3	G++ 4.6.3	Cython 0.18
64	35	34	572	706	15	473
128	130	130	2 202	2 674	36	1 835
256	581	502	8 974	10 496	90	7 417
512	1 928	1 908	34 709	41 911	210	29 357
1 024	6 940	7 180	138 013	166 376	503	116 163
2 048	25 804	28 543	550 162	669 665	1 104	456 910
4 096	106 434	116 914	2 198 307	2 651 932	2 540	1 836 857
8 192	445 101	487 205	8 797 116	11 297 116	5 749	7 349 664
16 384	1 872 967	2 035 501	41 487 413	46 197 563	12 895	32 750 002

1. Tab.: Výkonnostní porovnání jednotlivých implementací (zdroj: vlastní měření), všechny časy jsou uvedeny v milisekundách [ms].

Počet vstupních dat odpovídá počtu vzorků signálu. MP byl testován na sinusovém signálu, protože jsem experimentálně změřil, že typ signálu nemá vliv na výkonnost algoritmu.

Naměřené hodnoty jasně ukázaly, že implementace v nízkoúrovňových jazycích je bezkonkurenčně nejrychlejší oproti interpretovaným jazykům typu Python a Java.

V následujícím grafu jsou pro přehlednost zobrazeny pouze výsledky G++, Python 2.7 a Java SE 6. Měřítko je **logaritmické**.



7.1. Obrázek: Závislost rychlosti implementací MP (zdroj: vlastní výpočty)

7.6 Rychlost jazyků

7.6.1 C++

Nízkoúrovňové jazyky mají obrovskou výhodu díky překladu pro konkrétní platformu. Pokud se daný kód přeloží pouze pro **konkrétní instrukční sadu**, může využívat všechny instrukce, které na jiných platformách nemusí být dostupné. V konkrétním případu MP se jedná o matematický koprocesor, který se využívá pro většinu výpočtů v MP algoritmu. Díky tomu je implementace v jazyce C++ nejrychlejší.

7.6.2 Java

Java nemůže využívat speciální instrukce přímo, ale přes **nasimulované vstupy a výstupy**[23] JVM. Díky tomu nabírá značné zpoždění u každé aritmetické operace z důvodů vysoké režie.

7.6.3 Python

Rychlost jazyka Python je zásadně ovlivněna především díky jeho **dynamickému typování**. I při zpracovávání cyklů již nabírá Python zpoždění oproti Javě. V každém cyklu se musí zkontrolovat, zda stále platí podmínka pro výpočet následujícího cyklu. V jazyce Java i C++ se daná podmínka zkontroluje jedinou instrukcí procesoru typicky porovnáváním čísla integer a konstanty. V jazyce Python se musí navíc při každém porovnání zkontrolovat typ proměnných[24], následně zjistit jak je porovnat (pokud existuje komparátor dvou konkrétních typů) a pak teprve provést operaci porovnání.

Navíc testovaný interpret CPython neobsahuje **JIT kompilátor** ani AOT[25]. Tyto dvě technologie citelně zvyšují výkon interpretovaných jazyků, protože kompilují interpretovaný jazyk za běhu aplikace na strojový kód a tím se přibližují rychlosti nízkoúrovňových programovacích jazyků.

Díky těmto vlastnostem je jazyk Python velmi nevhodný pro matematické aplikace, kde záleží na výkonu samotné aplikace.

23 Bytecode Javy je multiplatformní – pokud daná platforma nepodporuje některé instrukce, musí je JVM implementovat softwarově.

24 Interpret nemůže vědět, jestli programátor nezměnil během cyklu typ některé proměnné.

25 Ahead-Of-Time – kompilátor čte instrukce napřed a díky tomu může optimalizovat průběh výpočtu.

8 Vizualizace výstupu MP

Výstupem algoritmu MP je matice, kde řádky představují jednotlivé atomy a sloupce jednotlivé parametry měřítka, posunu, frekvence a fázového posunu.

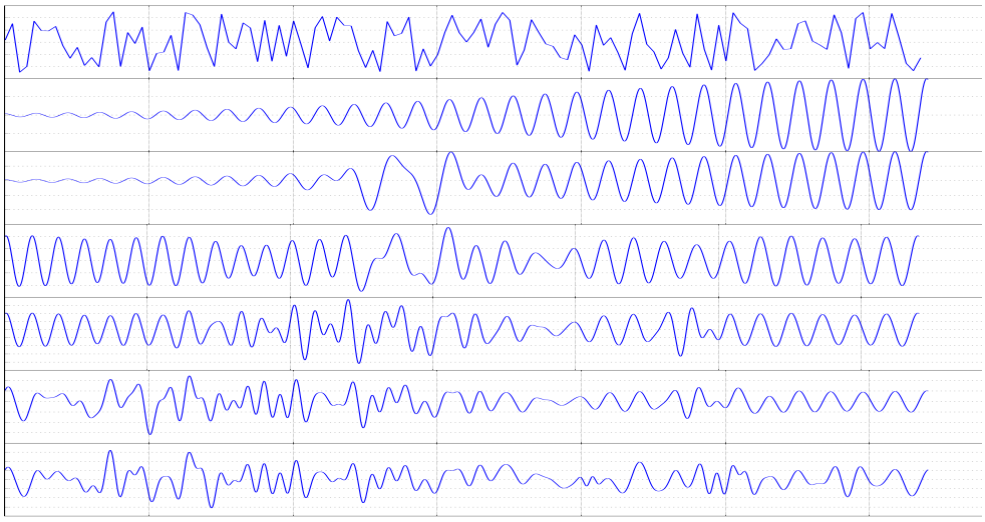
Pro ověření výsledků MP je tedy nutné zobrazit původní signál a vypočítat průběh součtu jeho atomů. Zobrazení původního signálu je triviální. Pro spočítání průběhu funkce Gaborových atomů se musí dosadit do vzorce Gaussova okna (vzorec 5.3 str.: 15) přepis pro Gaborovy atomy (vzorec 5.4 str.: 15):

$$g_{(s,u,v,w)}(t) = e^{-\pi\left(\frac{t-u}{s}\right)^2} \cdot \cos(vt + w) \tag{8.1}$$

Tento vzorec se použije pro vypočítání aktuálních hodnot pro jednotlivé body. Rozsah proměnné *t* musí být stejný jako u původního signálu.

Vizualizace byla implementována nejprve v jazyce Java pomocí knihovny JFreeChart pro kontrolu výsledků při následné implementace MP v Pythonu. Vizualizace v Pythonu využívá knihovnu matplotlib, která velmi těsně spolupracuje s knihovnou numpy a scipy. Toto trio knihoven dohromady vytvářejí konkurenci pro běžná výpočetní prostředí typu MATLAB.

Na následujícím obrázku 8.1 je vizualizace původního signálu a jednotlivé stupně iterace (první řádek je původní signál, další jsou iterace 1, 2, 3, 5, 10, 20).

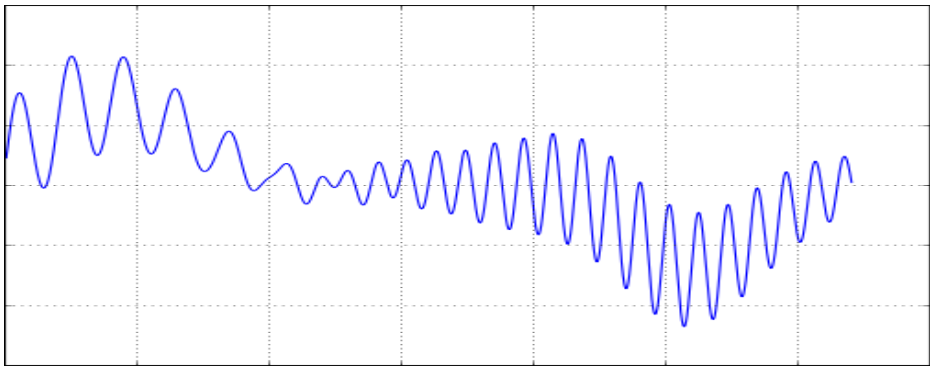


8.1. Obrázek: Ukázka vizualizace výstupu MP (zdroj: vlastní výpočty)

8.1 Výstupy jednotlivých implementací

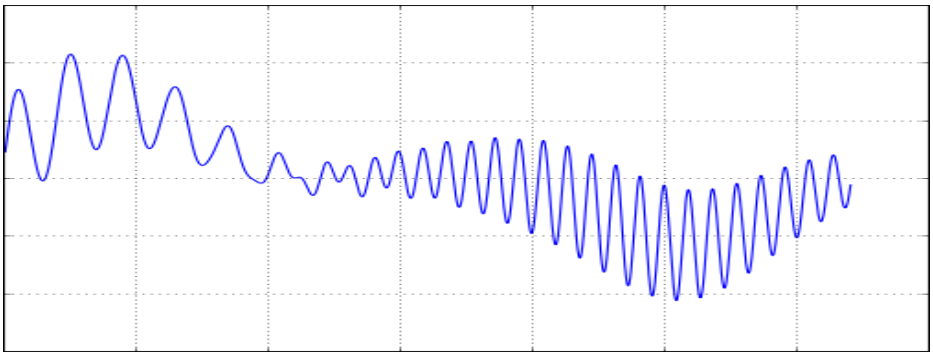
Bylo zjištěno, že největší odchylku mezi výsledky z jednotlivých implementací (Java, Python, C++) způsobují uměle vytvořené signály[26]. Proto byly jednotlivé algoritmy otestovány na sinusovém signálu.

Implementace Javy a Pythonu mají stejné výsledky na 13 platných číslic. Jejich výstup proto vypadá v grafu totožně:



8.2. Obrázek: Vizualizace MP Java a Python, sinusový signál, 5 iterací. (zdroj: vlastní výpočty)

Implementace C++ podává velmi podobné výsledky v běžných situacích. Rozdíl je patrný až na uměle vytvořených signálech:



8.3. Obrázek: Vizualizace MP C++, sinusový signál, 5 iterací. (zdroj: vlastní výpočty)

26 Tyto signály také způsobují různé artefakty při následné vizualizaci, neboť k tomuto účelu není slovník Gaborových atomů vhodný.

9 Závěr

Úspěšně jsem implementoval a změřil výkonnost algoritmu Matching Pursuit v jazyce Python. Měření ukázalo, že Python není vhodný pro rozsáhlé matematické výpočty.

Jazyk Python vzal zažité (a často i zastaralé a zbytečné) programovací konvence a zavedl z nich zápis programovacího jazyka, který se blíží matematickému zápisu. V poslední době vzniká velké množství projektů okolo Pythonu především díky rychlosti vývoje prototypů aplikací. Tento jazyk velmi usnadňuje programátorům jejich práci, bohužel na úkor výkonu samotné aplikace.

V současné době existují tři hlavní překladače: CPython, PyPy a Cython. Jakmile bude interpret PyPy dokončen, nic nebrání jazyku Python vyrovnat se Javě i svou výkonností. Překladač Cython je ve fázi vývoje a zatím se nevyplatí ho používat, především díky častým změnám v syntaxi jazyka samotného a kvůli chaotickému zápisu programového kódu.

Python v kombinaci s knihovnami numpy, scipy a matplotlib se vyrovnají i prostředí MATLAB. Tato kombinace obsahuje všechny prostředky nutné k rychlému vývoji prototypů aplikací v doméně neuroinformatiky.

Reference

[01] PEREZ, Fernando a Brian E. GRANGER. IPython: A System for Interactive Scientific Computing. Computing in Science. 2007, vol. 9, issue 3, s. 21-29. DOI: 10.1109/MCSE.2007.53. Dostupné z: http://ipython.org

[02] TRAVIS, E. Oliphant. Guide to NumPy. USA: Trelgol Publishing, 2006. Dostupné z: http://www.numpy.org/

[03] ERIC, Jones, Oliphant TRAVIS, Peterson PEARU. SciPy: Open source scientific tools for Python. 2001. Dostupné z: http://www.scipy.org/

[04] ScientificPython [online]. 2007 [cit. 2013-05-06]. Dostupné z: http://dirac.cnrs-orleans.fr/plone/software/scientificpython/

[05] PROKOP, Tomáš. ZČU. Metody vhodné pro vyhodnocování elektrofyziologických měření. 2012.

[06] Matching pursuit. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-23]. Dostupné z: http://en.wikipedia.org/wiki/Matching_pursuit#Applications

[07] ŘONDÍK, Tomáš. Methods for Detection Waveforms in BCI Systems: The State of the Art and the Concept of Ph.D. Thesis. s. 34-37.

[08] Matching pursuit. In: DURKA, Piotr Jerzy. Scholarpedia journal [online]. San Diego, CA: Scholarpedia.org, 2007 [cit. 2013-04-23]. Dostupné z: http://www.scholarpedia.org/article/Matching_pursuit

[09] FERRANDO, Sebastian E.; KOLASA, Lawrence A.; KOVAČEVIĆ, Natasha. Algorithm 820: a flexible implementation of matching pursuit for Gabor functions on the interval. ACM Transactions on Mathematical Software (TOMS), 2002, 28.3: 337-353.

[10] NICULAE, Vlad. My programming and machine learning blog [online]. 2011 [cit. 2013-05-07]. Dostupné z: http://blog.vene.ro/

[11] BEHNEL, Stefan, Robert BRADSHAW, Craig CITRO, Lisandro DALCIN, Dag Sverre SELJEBOTN a Kurt SMITH. Cython: The Best of Both Worlds. Computing in Science. 2011, roč. 13, č. 2, s. 31-39. ISSN 1521-9615. DOI: 10.1109/MCSE.2010.118. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5582062

Příloha A

Seznam nalezených projektů v doméně neuroinformatiky.

Nifti2Dicom
Převádí 3D snímky z CT[27]obrazy do 2D snímků ve formátu DICOM 2D.
C++
http://www.bio.dist.unige.it/projects/national/85

Neural Signal Processing Algorithms
Algoritmy především pro dynamické lokalizování polohy neuronových aktivit v mozku z EEG a MEG měření.
Matlab
http://www.neurostat.mit.edu/home

NDF Toolbox
Knihovna pro převod dat z neuroinformatických experimentů do formátu CARMEN.
Java, C++, Matlab
http://www.carmen.org.uk/software/descriptions/softwareproduct.2010-11-11.1935112106

BRAIN Motor Imagery Based BCI
Systém pro měření a analýzu EEG signálu. Určeno především pro subjekty představující si pohyb.
Matlab
http://eeg.pl/Members/jarekz/brainerds

DTF calculations package
Balík knihoven určených pro výpočet Directed transfer function (DFT) z EEG měření.
Matlab
http://eeg.pl/Members/maciek/MMultar

OpenBCI systém
Platforma pro Brain Computer Interface (ovládání počítače pomocí myšlenek).
Python
http://bci.fuw.edu.pl/wiki/OpenBCI_system

http://eeg.pl/Members/dlaszuk/graphic-udp
Modul k openBCI, rozblikává diody, evokuje akční potenciál k meření EEG

27 počítačová tomografie (CT) se nejčastěji využívá k diagnostice změn mozku.

TFStats package
Software pro hledání a analýzu ERD/ERS[28] v naměřeném EEG signálu.
Matlab
http://eeg.pl/Members/jarekz/tfstats-package

QtMapper
Nástroj pro mapování EEG signálu na 3D modelu mozku.
C++
http://eeg.pl/Members/dobi/qtmapper

M-eega
Nástroj pro zpracování neuroinformatických dat pro osoby bez znalosti vyšší matematiky.
Java
http://eeg.pl/Members/durka/m-eega

DDV
Software pro vizualizaci EEG dat naměřených ve spánku.
Borland Delphi
http://eeg.pl/Members/durka/ddv

SigViewer
Software pro vizualizaci EEG dat.
C++
http://bci.tugraz.at/downloads.html

TOBI interface & server
Server a klient pro vysílání a zpracovávání biosignálů (například EGG).
C++
http://bci.tugraz.at/downloads.html

BioSig
knihovna pro zpracování EEG signálů.
C++ / Matlab
http://biosig.sourceforge.net/

tools4BCI
Framework pro BCI.
C++
http://tools4bci.sourceforge.net/

28 Event-related desynchronization (ERD) and synchronization (ERS)

NEMO ERP Analysis Toolkit
Nástroje pro analýzu dat z EEG a MEG měření.
MATLAB
http://nemo.nic.uoregon.edu/wiki/NEMO_ERP_Analysis_Toolkit

Spykeviewer
Software pro analýzu a vizualizaci elektrofyziologických dat.
Python
https://github.com/rproepp/spykeviewer

NEO
Software pro analýzu a vizualizaci elektrofyziologických dat.
Python
http://packages.python.org/neo/

Neuroshare
Nástroj pro zpřístupnění dat z proprietárních formátů v rámci neuroinformatiky.
Python, MATLAB
http://www.g-node.org/projects/neuroshare-tools

Pymeg
Analýza a vizualizace měření z MEG.
Python
https://github.com/badbytes/pymeg/wiki

Neurodata
Software pro analýzu a vizualizaci elektrofyziologických dat.
Matlab, Perl
http://sourceforge.net/projects/neurodata/

EEGLAB
Nástroj pro zpracování signálů EEG a MEG.
MATLAB
http://sccn.ucsd.edu/eeglab/

EEGview
Program k zaznamenávání a zobrazování EEG signálů.
ANSI C
http://cnbi.epfl.ch/software/eegview.html

Chronux
Software pro analýzu neuroinformatických dat.
MATLAB
http://www.chronux.org/

BlueSpike
Software pro analýzu neuroinformatických dat v reálném čase.
ANSI C
https://github.com/globalvariable/BlueSpike

Aghermann
Simulace vlny S v signálu EEG.
C++
http://johnhommer.com/academic/code/aghermann/

PyBCI
BCI systém.
Python
http://pybci.sourceforge.net/

NeuroTools
Framework pro simulaci neuronových sítí.
Python
http://neuralensemble.org/trac/NeuroTools

BrainVoyager QX
Komplexní nástroj pro neurovědy.
C++
http://www.brainvoyager.com/

BSMART
Software pro analýzu neuroinformatických dat.
MATLAB, ANSI C
http://software.incf.org/software/bsmart

eConnectome
Software pro vyzualizaci neuroinformatických dat.
MATLAB
http://econnectome.umn.edu/

Příloha B

Seznam nalezených řešení možné spolupráce jazyků MATLAB, Java a Python.

Java a MATLAB

Matlab nativně podporuje Javu a některé jeho části běží v JVM. Je možné volat nativní metody přímo z Javy, spouštět nativní bytecode a spouštět vlastní programy.

http://blogs.mathworks.com/community/2009/07/06/calling-java-from-matlab/
http://www.cs.yale.edu/homes/spielman/ECC/javaMatlab.html

MATLAB a Java

Společnost Mathworks nabízí komerční řešení, které přeloží libovolný MATLAB program do bytecode pro JVM. Existují i nekomerční řešení, které ovládají MATLAB nebo umí s ním spolupracovat a předávat zprávy (výsledky) JVM.

https://code.google.com/p/matlabcontrol/
http://www.mathworks.com/products/javabuilder/
http://www.mathworks.com/matlabcentral/fileexchange/10759
http://matlab4java.wordpress.com/instructions/

Java a Python

Javu lze integrovat do Pythonu různými způsoby:

- Překladač JPython využívá JVM a překládá Python pro Javu.
- Existují projekty, které překládají zdrojový kód Javy do Pythonu.
- Překladač který spojuje JVM a Python interpret .

http://jpype.sourceforge.net/
https://github.com/natural/java2python
http://www.slideshare.net/onyame/mixing-python-and-java

Python a Java

Překladač JPython využívá JVM a překládá Python pro Javu.

http://www.slideshare.net/onyame/mixing-python-and-java
http://wiki.python.org/jython/JythonFaq/GeneralInfo

Matlab a Python

Existují 2 možnosti: přeložit zdrojový kód MATLAB do jazyka Python nebo použít Matlab jako Python knihovnu.

https://github.com/jaderberg/python-matlab-bridge
http://ompc.juricap.com/
http://mlabwrap.sourceforge.net/
http://claymore.engineer.gvsu.edu/~steriana/Python/pymat.html

Python a Matlab

Tato možnost existuje díky využití interpretu Jython nebo spolupráce Matlabu a interpretu Python.

http://stackoverflow.com/questions/1707780/call-python-function-from-matlab

Příloha C

Tato příloha obsahuje uživatelskou dokumentaci pro přeložení a spuštění jednotlivých implementací MP. Všechny programy mají výstup do konzole.

C++

Zdrojové soubory implementace MP jsou uloženy ve složce CPP. Pro přeložení souborů je přiložen i soubor Makefile. Spustitelný soubor se překládá příkazem make. Pro úspěšný překlad musí mít uživatel nainstalovaný překladač jazyka C++:

Windows:	MinGW	http://www.mingw.org/
Linux:	GNU compiler	http://gcc.gnu.org/

Program se spouští skriptem cpp_speed.bat (cpp_speed.sh je určen pro operační systém Linux) a vypisuje koeficienty vypočítaných atomů pro předpřipravená data ve složce data.

Java

Zdrojové soubory jsou uloženy ve složce JAVA. Pro přeložení souborů a vytvoření spustitelného JAR souboru je přiložen soubor build.xml pro program Apache Ant. Ten se spouští příkazem ant make:

Apache Ant	http://ant.apache.org/

Pro spuštění programů Java musí mít uživatel nainstalované Java JRE, pro překlad JDK:

Java JRE	http://www.java.com/en/download/
Java JDK	http://www.oracle.com/technetwork/java/javase/downloads/

Skript java_graph.bat (java_graph.sh) spouští vizualizaci tří druhů signálů (bílý šum, funkce sinus a funkce sinus se šumem) a následně vizualizaci páté iterace algoritmu MP pro každý druh signálu.

Skript java_speed.bat (java_speed.sh) spouští program pro výpočet gaborových atomů předpřipravených ve složce data.

K programu jsou dodány knihovny pro zobrazování grafu (JCommon a JfreeChart) a pro matematické operace (commons-math), takže není potřeba žádná další instalace.

Python

Zdrojové soubory jazyka Python se nepřekládájí, pro spuštění testu rychlosti (python27.bat) musí mít uživatel nainstalován interpret jazyka Python a knihovnu NumPy:

Python : http://www.python.org/getit/
Numpy: http://www.scipy.org/Download

Pro vizualizaci výsledků musí uživatel navíc nainstalovat knihovny SciPy a matplotlib:

SciPy: http://www.scipy.org/Download
matplotlib: http://matplotlib.org/downloads.html

Skript python27_graph.bat (python27_graph.sh) spouští vizualizaci tří druhů signálů (bílý šum, funkce sinus a funkce sinus se šumem) a následně vizualizaci páté iterace algoritmu MP pro každý druh signálu.

Skript python27_speed.bat (python27_speed.sh) spouští program pro výpočet gaborových atomů předpřipravených ve složce data.