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

Bakalářská práce

Workflow pro BCI experiment

Plzeň 2020 Pavel Průcha

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2018/2019**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Pavel PRŮCHA**
Osobní číslo: **A16B0114P**
Studijní program: **B3902 Inženýrská informatika**
Studijní obor: **Informatika**
Název tématu: **Workflow pro BCI experiment**
Zadávající katedra: **Katedra informatiky a výpočetní techniky**

Zásady pro vypracování:

1. Seznamte se s podstatou BCI experimentů a BCI aplikací vytvářených na katedře.
2. Prostudujte současné možnosti a dostupné nástroje pro vytváření workflow pro BCI aplikace.
3. Na základě bodu 1 navrhněte vhodnou modifikaci stávajícího BCI experimentu.
4. BCI experiment z bodu 3 implementujte v nástroji vybraném v bodu 2.
5. Ověřte a zhodnoťte výsledné řešení.

Rozsah grafických prací: **dle potřeby**

Rozsah kvalifikační práce: **doporuč. 30 s. původního textu**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Roman Mouček, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **10. října 2018**

Termín odevzdání bakalářské práce: **2. května 2019**

Radová

Doc. Dr. Ing. Vlasta Radová
děkanka

Západočeská univerzita v Plzni
L.S.
Fakulta aplikovaných věd

Brada

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 15. října 2018

Prohlášení

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

V Plzni dne 7. května 2020

Pavel Průcha

Poděkování

Rád bych poděkovat vedoucímu mé práce Ing. Romanovi Moučkovi, Ph.D. za odborné vedení a cenné rady při zpracování mé práce.

Abstract

Brain-computer interface (BCI) can significantly help people with not only movement problems. BCI works with electroencephalographic data (EEG). This work simplifies the work with the processing of this data. The aim of the bachelor thesis is to find suitable tools for creating a workflow and select the best one for the BCI workflow. After finding the most suitable tool, the task is to implement particular methods for reading, preprocessing and visualization of eeg data and make them as the nodes compatible for the selected tool. It will be possible to connect these nodes and create any workflow.

Abstrakt

Brain-computer interface (BCI) dokáže významně pomoci lidem s nejen pohybovými problémy. BCI pracuje s elektroencefalografickými daty (EEG). Tato práce zjednodušuje práci se zpracováním těchto dat. Cílem bakalářské práce je najít vhodné nástroje pro vytváření workflow a vybrat z nich ten nejlepší pro BCI workflow. Po nalezení nejvhodnějšího nástroje je úkolem implementovat jednotlivé metody pro načítání, zpracování a vizualizaci eeg dat a udělat z nich uzly kompatibilní pro vybraný nástroj. Tyto uzly bude možné spojovat a vytvářet tak libovolné workflow.

Obsah

1 Úvod

Tato práce se věnuje workflow pro BCI experiment. "Brain-computer interface, dále jen BCI, je rozhraní, které slouží k propojení zařízení snímající mozkovou aktivitu s počítačem. Ke snímání mozkové aktivity se používají zařízení založená na principu metody EEG (elektroencefalografie). Jedna z oblastí, ve které je možné efektivně využít možnosti BCI technologie, je pomoc lidem s motorickým postižením ke snadnějšímu ovládání nejen počítačů, ale i celé řady dalších externích zařízení, kde počítač hraje roli prostředníka mezi mozkem a zařízením."[13]

Práce je organizovaná do devíti kapitol. Pro motivaci a inspiraci k této práci jsem se nejprve seznámil s podstatou BCI experimentu a BCI aplikací vytvářených na katedře. Své poznatky uvádím v kapitole State of the Art, kde dále popisuji pojmy a nástroje důležité pro tuto práci. Poté jsem se soustředil na práci vlastní. Tou je prostudovat současné možnosti a dostupné nástroje pro vytváření workflow pro BCI aplikace, navržení vhodné modifikace BCI experimentu a její implementace ve vhodném nástroji.

Workflow je posloupnost kroků komplexnější činnosti rozdělené do menších částí, které na sebe navazují a předávají si data. Je to tedy tok dat, na jehož konci je výsledek, v podobě vizualizace nebo v textové podobě. Workflow je často vytvářené pomocí nástroje, který vlastní práci vytváření workflow usnadní. Problematika je řešena v kapitole Specifikace požadavku. Zde jsou sepsané metody, které by měly být ve vybraném nástroji modifikovány a implementovány. Následuje kapitola Analýza, ve které nástroje porovnávám, hodnotím a následně vybírám nejvhodnější nástroj pro BCI experiment. Ten jsem vybíral celkem ze čtyř nástrojů, které jsou vhodné pro vytváření workflow pro BCI aplikace. Po zhodnocení vybírám ten nejlepší pro dané workflow.

V kapitole Analýza a design vybraného nástroje popisuji funkce vybraného nástroje a jak nástroj vypadá. Hlavním cílem této práce je modifikace a implementace balíčku dílčích metod pro zpracování BCI dat do tohoto nástroje. Další kapitolou je tedy Implementace. V této kapitole vysvětluji funkci jednotlivých implementovaných metod a jejich použití v nástroji. Výsledkem práce je efektivní prostředek pro vytváření a správu BCI workflows. V kapitole Testování tyto metody testuji.

Další kapitolou je Zhodnocení výsledku, kde hodnotím výsledky svojí práce z praktického hlediska. Poslední kapitolou je závěr. Zde je souhrn celé práce a její celkové hodnocení.

2 State of the Art

V této kapitole jsou popsány pojmy a nástroje důležité pro tuto práci. Je zde vysvětlená podstata BCI experimentů, ukázány některé projekty z oblasti BCI a zmíněné důležité komponenty této práce.

2.1 BCI experiment

BCI je rozhraní propojující mozek s počítačem pomocí mozkové aktivity. BCI pomáhá lidem s motorickým postižením. Může jim tak poskytnout nový komunikační kanál, jako malou kompenzaci za pohybové schopnosti, o které přišli. Cílem je obnovit pohyb nebo poskytnout zařízení, která jim pomohou.

BCI neřeší jen nápravu pohybového aparátu, ale výzkum se zaměřil i na obnovu poškozeného zraku. Jedním z prvních vědců, kteří dokázali vytvořit mozkové rozhraní pro obnovení zraku, byl vědec William Dobelle. Systém používá malou kameru namontovanou do brýlí, které nosí slepý člověk. Obrazy z kamery jsou přenášeny na chirurgicky implantované elektrody připojené k mozku [12].

2.2 BCI aplikace na KIV

Důležitým projektem je **Czech-Bavarian BASIL project**. Na tomto projektu pracovaly týmy z fakulty aplikovaných věd na západočeké univerzitě v Plzni ve spolupráci se zahraničními týmy a nemocnicemi. Tento BCI projekt se zaměřil na používání osvědčených postupů pro design, vývoj, testování a on-site nasazení BCI systémů v nemocnicích, zdravotnických zařízení a v domácím prostředí. V rámci tohoto projektu vznikl nástroj *Dynamic Signal Processing Workflow Designer* ve spolupráci s International Neuroinformatics Coordinating Facility. Vice o tomto nástroji je v sekci Nástroje. V rámci projektu **The BASIL BCI project** vznikl také on-line klasifikátor pro BCI data **SSVEP on-line classification** vyvíjený Lukášem Vařekou.

2.3 BrainVision

BrainVision je formát *eeg* dat, který je použitý v této práci pro vstupní data pro workflow. *Eeg* je diagnostická metoda používána k záznamu elektrické aktivity mozku.

Struktura formátu BrainVision se skládá ze tří samostatných souborů:

- *.vhdr* – Hlavičkový soubor, který obsahuje metadata a odkazy na druhé dva soubory .vmrk a .eeg
- *.vmrk* – Markerový soubor obsahující informace o událostech v datech
- *.eeg* – Binární datový soubor s eeg daty

2.4 MNE-Python

MNE-Python je open-source balíček pro jazyk `Python`. Používá se pro výzkum, vizualizaci a analýzu neurofyziologických dat, jako jsou *MEG (Magnetoencefalografie)*, *EEG (Elektroencefalografie)*, *sEEG (Stereoelektroencefalografie)* a *ECoG (Electrocorticography)*. Balíček obsahuje několik modulů pro nahrání dat mnoha formátů, výstup (ve formě dat nebo různých grafů a jiných vizualizačních formátů), předzpracování dat, odhad zdroje, časově-frekvenční analýzu, konektivitu, strojové učení nebo statistiku [4].

V této práci je **MNE-Python** využitý při implementaci důležitých částí kódu pro workflow, jako například nahrání souboru, vizualizace a logika metod.

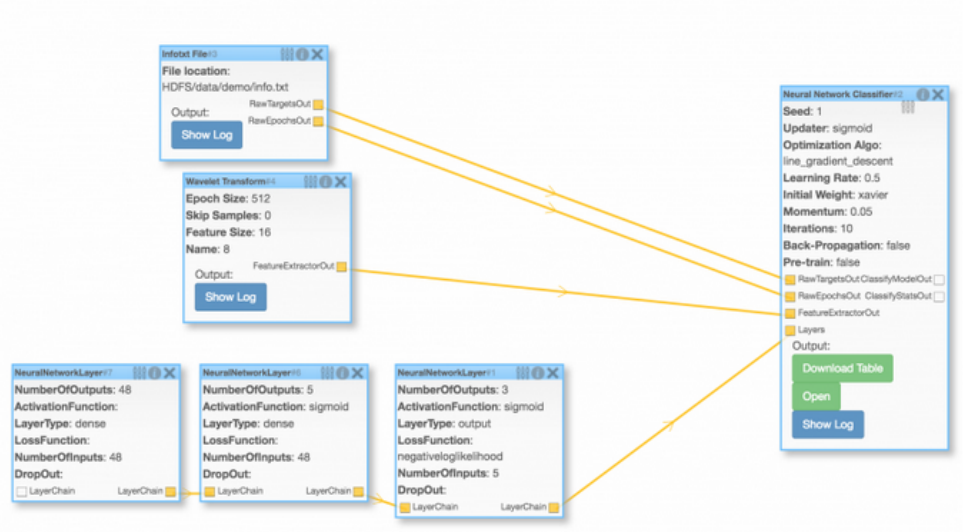
2.5 Nástroje

K vytvoření požadovaného workflow pro můj BCI experiment je vhodné použít nástroj, který práci usnadní. Nástrojů pro vytvoření workflow je dnes více, ale ne každý vyhovuje požadavkům pro tento BCI experiment. Pro tento účel jsem našel čtyři různé nástroje: *Dynamic Signal Processing Workflow Designer*, *Snakemake*, *NeuroPype* a *Orange*.

2.5.1 Dynamic Signal Processing Workflow Designer

Nástroj **Dynamic Signal Processing Workflow Designer** je vyvíjený na Katedře informatiky a výpočetní techniky na Západočeské univerzitě v Plzni ve spolupráci s INCF (International Neuroinformatics Coordinating Facility). Nástroj pracuje s metodami zpracování EEG signálu a evokovaných potenciálů (ERP) a umožňuje využít metody strojového učení pro vytváření asistenčních systémů pro osoby se sníženou schopností pohybu. Po shromáždění potřebných údajů o mozku dojde k manipulaci s jiným objektem (otevření dveří, okna, zapnutí televize,...). Díky grafickému rozhraní **Dynamic Signal Processing Workflow Designer** je snazší dosáhnout

modelování workflow. Workflow se ukládá jako soubor ve formátu JSON a GUI je implementováno v jazyce Java. Je možné, aby si uživatel implementoval i svoje vlastní metody. Příklad je možné vidět na obrázku 2.1 na straně 13.



Obrázek 2.1: Příklad workflow nástroje Dynamic Signal Processing Workflow Designer [9]
Na obrázku je vidět příklad jednoduchého EEG workflow, kde v horním bloku *Infotxt file* se načítá textový soubor. V prostředním bloku dochází k extrakci dat založené na diskrétní vlnkové transformaci. Ve spodní části workflow jsou tři vrstvy neuronové sítě. Výstup všech třech částí vstupuje do klasifikátoru neuronových sítí.

2.5.2 Snakemake

Snakemake je open-sourcový nástroj pro vytváření workflow. Workflow je popsané pomocí čitelného jazyka založeného na Pythonu. **Snakemake** pracuje na strukturách, podle kterých se řídí spouštění pravidel. Po spuštění vytvořených pravidel v souboru `Snakefile` se vytvoří DAG (Directed acyclic graph) – řízený acyklický graf. DAG ukazuje propojení pravidel ve workflow. Každý z daných uzlů v DAG je jedna úloha ze souboru `Snakefile`. Propojení dvou uzlů znamená, že výstup jednoho z uzlů je potřeba na výstupu uzlu druhého. Některé úlohy se dají provádět paralelně a některé sériově. **Snakemake** má mnoho využití. Jedním z nich je optimalizace využití CPU vzhledem k prahu dostupných jader. Díky tomu je umožněno škálování **Snakemake** do prostředí s pevným limitem použitých jader CPU [6].

2.5.3 NeuroPype

NeuroPype je výkonná platforma pro BCI (brain-computer interfacing) v reálném čase. Dá se použít i pro zobrazení nervového systému a zpracování neurálních/biologických signálů. Je z části založen na softwaru *MNE-Python*. **NeuroPype** je tedy programovací systém pro tok dat. Zahrnuje *NeuroPype Engine*, samostatný Python balíček, který implementuje všechny funkce zpracování dat **NeuroPype**, open-source *Pipeline Designer* a kolekci dalších nástrojů.

Pipeline Designer je GUI aplikace nástroje **NeuroPype** pro vytvoření workflow. Přetažením uzlů se z nabídky anotovaných a připravených balíčků dostanou uzly do pracovního prostředí, kde se můžou propojit s ostatními uzly a případně je možné jim dopsat/upravit argumenty, aby byly připraveny pro workflow. *Pipeline Designer* je založený na aplikaci *Orange* [11].

2.5.4 Orange

Orange je nástroj strojového učení a data mining pro analýzu dat pomocí vizuálního programování. **Orange** používá open-source knihovny pro vědecké účely jako je `numpy`, `scipy` a další, ale není problém použít pro vlastní implementaci jiné externí nebo vlastní knihovny. Základních předinstalovaných balíků je celkem šest – data, visualize, classify, regression, evaluate and unsupervised. Několik dalších, vyvíjených komunitou využívající tento nástroj, se dá dodat pomocí funkce *add-on*.

Nástroj **Orange** je open-source software pod *GPL (General Public License)*. Je implementovaný pomocí programovacího jazyka *C++* a *Pythonu*, nicméně funkční metody jsou pouze v jazyce *Python*. Grafické uživatelské rozhraní nástroje **Orange** pracuje na frameworku `Qt`, což je multiplatformní aplikační framework používaný pro vytváření aplikačního software s grafickým uživatelským rozhraním [3].

3 Specifikace požadavků

Workflow je potřeba sestavit z funkčních metod, na kterých závisí logika experimentu. Požadavkem pro tuto práci je tedy implementovat jednotlivé metody pro funkční BCI workflow. Tyto metody se dělí na pět částí: *Metody pro načítání / ukládání dat, Metody pro předzpracování dat, Metody pro extrakci příznaků, Metody pro klasifikaci a Vizualizace.*

Metody pro načítání / ukládání dat:

- BrainVision formát: načtení/uložení
- Soubory formátu LabStreamingLayer (LSL) (xdf): načtení/uložení – LSL jsou on-line streamy EEG dat a markerů
- Stream LSL (vstup, výstup)

Metody pro předzpracování:

- Extrakce epoch
- Frekvenční filtrování
- Diskrétní vlnková transformace
- Matching pursuit
- Frekvenční spektrum
- Analýza nezávislých komponent (ICA – Independent Component Analysis)
- Průměrování
- Segmentace

Metody pro extrakci příznaků:

- Podvzorkování
- Průměrování časových oken
- Společný prostorový vzor (CSP – Common spatial pattern)

Metody pro klasifikaci:

- Lineární diskriminační analýza (LDA – Linear Discriminant Analysis)
- Metoda podpůrných vektorů (SVM – Support Vector Machines)
- neuronové sítě:
  - Dopředná neuronová síť
  - Konvoluční neuronové sítě
  - Rekurentní neuronové sítě

Vizualizace:

- Graf průběhů amplitudy
- Mapa rozložení napětí na povrchu hlavy
- Časově-frekvenční mapy

4 Analýza

4.1 Nástroje

Částí mé práce je vybrat vhodný nástroj pro BCI experiment. V kapitole `State of the Art` jsem popsal všechny čtyři nástroje. Popsané jsou v sekcích `Dynamic Signal Processing Workflow Designer`, `Snakemake`, `NeuroPype` a `Orange`.

4.2 Porovnání nástrojů

Pro výběr nejlepšího nástroje pro tento BCI experiment z nástrojů *Dynamic Signal Processing Workflow Designer*, *Snakemake*, *NeuroPype* a *Orange* je potřeba nejprve dané nástroje porovnat, aby se dalo co nejlépe zhodnotit, který z nástrojů je nejlepší. Nástroje porovnávám podle jednoduchosti a efektivitě anotace, vybavení (co se týče anotovaných metod), dostupnosti neanotovaných metod, GUI, celkové spolupráce nástroje s uživatelem a licenčních podmínek.

4.2.1 Anotace

V softwarovém inženýrství je anotace definována jako speciální forma syntaktických metadat ve zdrojovém kódu. Anotace jsou vloženy do kódu a ovlivňují anotovanou třídu, metodu, proměnnou, parametr nebo balíček [8].

Anotace Dynamic Signal Processing Workflow Designer

Anotace je v *Dynamic Signal Processing Workflow Designeru* zápis, jak přiřadit nějakému elementu (třída, metoda, proměnná) příznak, metadata nebo informaci mimo běžný kód.

Pro použití vlastních metod ve workflow v *Dynamic Signal Processing Workflow Designer* je potřeba každou metodu, neboli uzel, uzavřít do tzv. bloků. Blok uzavírá implementaci jednotlivého uzlu. Obsahuje tedy definici vstupu, výstupu, atributů, logiky a dalších důležitých částí. Typ anotace ve *Dynamic Signal Processing Workflow Designer* určuje, jaký typ kódu (vstup, výstup,..) bude nadcházet. Anotace vždy začíná znakem zavináče – @. Pro definici bloku se musí určit typ bloku (@BlockType()), vstupy (@BlockInput()), výstupy (@BlockOutput()), vlastnosti (@BlockProperty()) a funkci

spuštění (@BlockExecute). Po správném doplnění všech potřebných anotací dostaneme funkční uzel.

Typ bloku **BlockType** je anotace třídy s atributy *type*, *family* a *runAsJar*. *Type* je identifikační řetězec konstanty pro typ bloku. *Family* je řetězec konstanty pro bloky, které mohou spolu nějak souviset. *RunAsJar* atribut je boolean. Pokud je *true*, bude blok spuštěn externě přes jeho soubor jar. Je možné využít i čtvrtý atribut popisu. Popis je znázorněn po kliknutí myší na 'i' v kroužku na bloku.

Příklad anotace BlockType:

```
@BlockType(type ="ARITHMETIC", family = "MATH", runAsJar = true)
public class ArithmeticBlock {
```

Vstup bloku **BlockInput** je anotace proměnné. Je možné ji použít, pokud je blok schopen přijmout nějaký vstup. Má atributy *name* a *type*. *Name* je identifikační řetězec proměnné pro vstup. *Type* je konstanta řetězce, která udává typ proměnné. Tato konstanta slouží k propojení vstupu s výstupem jiného bloku. Vstup bloku může ještě mít atribut *cardinality*, který udává kardinalitu vstupu.

Příklad anotace BlockInput:

```
@BlockInput(name = "Operand1", type = NUMBER,
    cardinality = ONE_TO_ONE)
private int op1=0;
```

Výstup bloku **BlockOutput** je také anotace proměnné. Je možné ji použít, pokud má blok nějaký výstup. I atributy jsou stejné: *name* a *type*. *Name* je identifikační řetězec konstanty pro výstup. *Type* je opět konstanta řetězce, která udává typ proměnné. Tato konstanta slouží k propojení výstupu se vstupem jiného bloku. Výstup bloku může také mít atribut *cardinality*, který udává kardinalitu výstupu.

Příklad anotace BlockOutput:

```
@BlockOutput(name = "Operand3", type = NUMBER,
    cardinality = ONE_TO_MANY)
```

```
private static int op3=0;
```

Vlastnost bloku **BlockProperty** je stejně jako výstup a vstup bloku anotace proměnné a má i stejné proměnné se stejnými funkcemi: *name* a *type*. Má navíc atribut *defaultValue*, což je řetězec, který slouží k inicializaci vlastnosti bloku. Jedná se o běhové parametry bloků a lze je změnit beze změny workflow.

Příklad anotace BlockProperty:

```
@BlockProperty(name ="Operation", type = STRING,
    defaultValue = "add")
private String operation;
```

Blok spuštění **BlockExecute** je anotace metody a je bez atributů. Provádí pak spuštění důležitých částí workflow.

Příklad anotace BlockExecute:

```
@BlockExecute
    public String process() throws IOException {
        switch (operation){
            case "add":
                op3=op1+op2;
                break;
```

Příklad třídy je na obrázku 4.1 na straně 20. Tato třída složí jako blok jednoduché kalkulačky, která dokáže sčítat, odčítat nebo násobit (určuje vlastnost bloku) čísla zadaná jako vstup bloku. Výsledek pak vrátí jako výstup bloku.

```
package data;

import cz.zcu.kiv.WorkflowDesigner.Annotations.*;
import cz.zcu.kiv.WorkflowDesigner.Table;
import org.apache.commons.io.FileUtils;
import static cz.zcu.kiv.WorkflowDesigner.Type.NUMBER;
import static cz.zcu.kiv.WorkflowDesigner.Type.STRING;
import static cz.zcu.kiv.WorkflowDesigner.WorkflowCardinality.ONE_TO_MANY;
import static cz.zcu.kiv.WorkflowDesigner.WorkflowCardinality.ONE_TO_ONE;

@BlockType(type ="ARITHMETIC", family = "MATH", runAsJar = true)
public class ArithmeticBlock {

    @BlockInput(name = "Operand1", type = NUMBER, cardinality = ONE_TO_ONE)
    private int op1=0;

    @BlockInput(name = "Operand2", type = NUMBER, cardinality = ONE_TO_ONE)
    private int op2=0;

    @BlockOutput(name = "Operand3", type = NUMBER, cardinality = ONE_TO_MANY)
    private static int op3=0;

    @BlockProperty(name ="Operation", type = STRING ,defaultValue = "add")
    private String operation;

    @BlockExecute
    public String process() throws IOException {
        switch (operation){
            case "add":
                op3=op1+op2;
                break;
            case "subtract":
                op3=op1-op2;
                break;
            case "multiply":
                op3=op1*op2;
                break;
        }

        return String.valueOf(op3);
    }
}
```

Obrázek 4.1: Blok třídy ArithmeticBlock [9]

Celý projekt, který obsahuje třídy workflow, se musí zabalit do jar souboru a tento soubor nahrát na **Workflow designer server project**. Zde se z nich dá vytvářet workflow. Workflow je pak možné exportovat do souboru typu JSON. Příklad takového JSONu je možné vidět na obrázku 4.2 na straně 21 a pokračování na obrázku 4.3 na straně 22. Na prvním obrázku jsou definovány bloky workflow. Je zde vidět, že každý blok obsahuje svoje identifikační číslo, typ bloku, modul, do kterého blok patří a hodnoty svých vlastností. Na druhém obrázku jsou definovány hrany bloků, tedy sousedi uzlů, na které bloky ve workflow navazují. Každá hrana má svoje identifikační číslo, číslo bloku, z kterého hrana přichází, číslo bloku, do kterého vstupuje a informace o těchto blocích.

```
{
    //List of blocks
    "blocks": [
        {
            "id": 1,                    //Block ID (Integer)
            "type": "ARITHMETIC",       //Block type (String)
            "module": "test.jar:test", //Module of the block (JarName:Package)
            "values": {                 //Values of properties
                "Operation": "add"
            }
        },
        {
            "id": 2,
            "type": "CONSTANT",
            "module": "test.jar:test",
            "values": {
                "Value": "10"
            }
        },
        {
            "id": 3,
            "type": "CONSTANT",
            "module": "test.jar:test",
            "values": {
                "Value": "5"
            }
        }
    ],
```

Obrázek 4.2: Příklad souboru JSON 1 [9]

```
//List of Edges (Connections between blocks)
"edges": [
    {
        "id": 1,          //Edge id (Integer)
        "block1": 3,      //Connection from block id (Integer)
        "connector1": [
            "Operand",    //From Block type
            "output"      //From Field name
        ],
        "block2": 1,      //Connection to block id   (Integer)
        "connector2": [
            "Operand2",   //To block type
            "input"       //To Field name
        ]
    },
    {
        "id": 2,
        "block1": 2,
        "connector1": [
            "Operand",
            "output"
        ],
        "block2": 1,
        "connector2": [
            "Operand1",
            "input"
        ]
    }
]
}
```

Obrázek 4.3: Příklad souboru JSON 2 [9]

Některé metody, které se mohou hodit pro moje workflow (popsány ve Specifikaci požadavků), jsou již implementované v Javě a pro *Dynamic Signal Processing Workflow Designer* anotovány, a jsou tedy plně připraveny k použití. Jsou to tyto metody:

- BrainVision formát: načtení/uložení
- Extrakce epoch
- Diskrétní vlnkové transformace
- Frekvenční filtrování
- Frekvenční spektrum
- Průměrování

Některé metody jsou dohledatelné a implementované v Javě, nejsou však anotované a bylo by je potřeba anotovat. To jsou tyto metody:

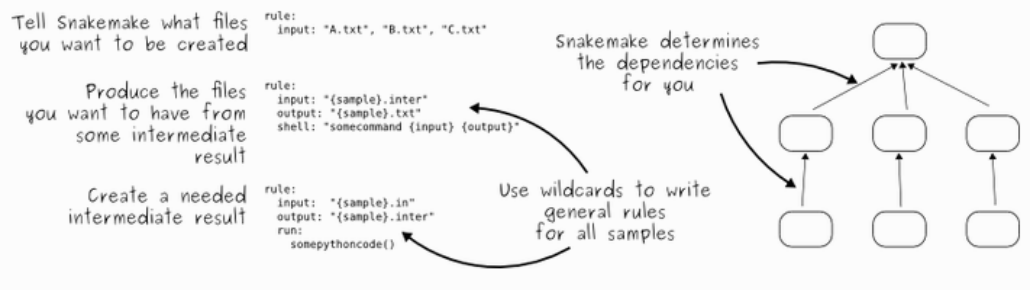
- Stream LSL (vstup, výstup)
- Matching pursuit
- Analýza nezávislých komponent (ICA)
- Segmentace (rovnoměrné úseky)
- Podvzorkování
- Lineární diskriminační analýza (LDA)
- Metoda podpůrných vektorů (SVM)
- Dopředná neuronová síť
- Konvoluční neuronové sítě
- Rekurentní neuronové sítě
- Graf průběhů amplitudy

Metody, které nejsou vůbec v Javě implementované:

- Soubory formátu LabStreamingLayer (LSL) (xdf): načtení/uložení
- Průměrování časových oken
- Společný prostorový vzor (CSP)
- Mapa rozložení napětí na povrchu hlavy
- Časově-frekvenční mapy

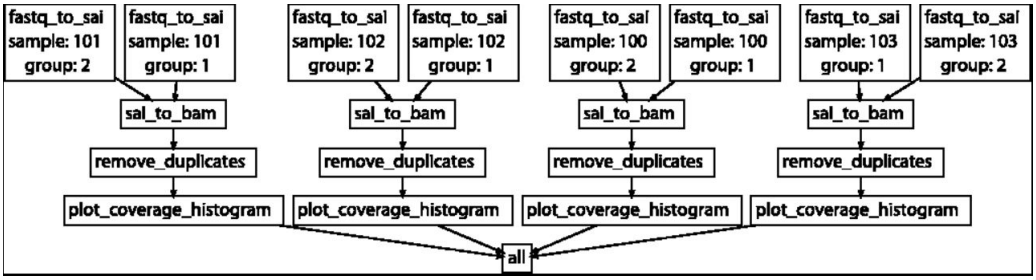
Anotace Snakemake

Anotace v nástroji *Snakemake* je o něco složitější, obzvlášť celá příprava workflow. Důležitý je zde soubor `Snakefile` (obrázek 4.4 na straně 24), který obsahuje všechny úkoly, pravidla a závislosti workflow napsané v Python skriptu. Pravidla popisují, jak ze vstupních souborů udělat soubory výstupní.



Obrázek 4.4: Příklad souboru Snakefile [6]

Závislosti pravidel ze souboru `Snakefile` jsou vidět po vygenerování DAG úloh. DAG je řízený acyklický graf, to je směrový graf s takovým postavením vrcholů, že každá hrana vede z vrcholu předka na vrchol potomka [5]. Příklad takového grafu je vidět na obrázku 4.5 na straně 24. Uzly na obrázku jsou úlohy, šipka mezi úlohou A a B znamená, že úloha B, potřebuje výstup úlohy A jako vstupní soubor. Cesta v DAG představuje sled úloh, které musí být provedeny sériově.



Obrázek 4.5: Příklad DAG [6]

Anotace NeuroPype

V *NeuroPypu* se jedna atomická část průběhu workflow nazývá uzel (často jedna třída). Každý uzel musí mít *DataPort*, který spojuje uzel s uzlem předchozím a následujícím pomocí přijímání a odesílání datových signálů. Může mít i další porty, které obsahují vlastnosti uzlu. Parametry těchto vlastností pak může uživatel měnit. A samozřejmě algoritmus, který se na daných datech aplikuje. Aby uzel správně fungoval, je nutné mu importovat engine modul, engine datového toku *NeuroPypu*.

```
from ...engine import *
```

Často je také potřeba importovat numpy, což je základní knihovna pro většinu našich uzlů.

```
import numpy as np
from ...utilities.helpers import parse_range
```

Dále musí každý uzel dědit z NeuroPype třídu *Node*

```
class Rereferencing(Node):
```

DataPort je třeba definovat. Jsou tři různé možnosti: IN - přijímání, OUT - odesílání a INOUT - přijímání a odesílání. Výchozí je INOUT, platí, pokud v *DataPort* není zadáno nic.

```
data = DataPort(Packet, "Data to process.")
```

Jeden uzel může mít i více DataPortů.

```
data1 = DataPort(Packet, "Input data 1.", IN)
data2 = DataPort(Packet, "Input data 2.", IN, mutating=False)
data3 = DataPort(Packet, "Input data 3.", IN, mutating=False)
data4 = DataPort(Packet, "Input data 4.", IN, mutating=False)
data5 = DataPort(Packet, "Input data 5.", IN, mutating=False)
outdata = DataPort(Packet, "Data to process.", OUT)
```

Vlastnosti uzlu, které je možné nastavit například v *Pipeline Designeru*, se definují pomocí *Portů* (Port, EnumPort, IntPort, FloatPort,...). Port ověří, jestli třída Portu je stejná její význam. V následujícím kódu má `axis` tedy typ *enum* a `cut_prop` typ *float*.

```
axis = EnumPort("space", tuple(axis_names), """Axis along
    which to take the reference or baseline.""",
    verbose_name='select reference along axis')
cut_prop = FloatPort(0.1, None, """Fractiont of the
    outliers to cut off for trim_mean option.""",
    verbose_name='propotion cutoff for trim mean option')
```

Aby se vytvořil nový uzel a přijal počáteční hodnoty portů, je nutné implementovat konstruktor třídy `__init__`, který použije rodičovskou třídu a vytvoří novou instanci uzlu s vlastnostmi třídy.

```
def __init__(self, **kwargs):
    """Create a new node. Accepts initial values for the ports."""
```

```
        super().__init__(**kwargs)
```

V Python kódu samozřejmě nesmí chybět algoritmus/logika funkce daného uzlu, bez které by uzel neměl význam.

V nástroji *NeuroPype*, jehož uzly jsou zpravidla implementovány v Pythonu, je anotováno velké množství uzlů, které bych mohl potřebovat do workflow, na kterém budu pracovat. Tyto uzly jsou již anotovány v nástroji *NeuroPype*:

- Soubory formátu LabStreamingLayer (LSL) (xdf): načtení/uložení
- Stream LSL (vstup, výstup)
- Extrakce epoch
- Frekvenční filtrování
- Frekvenční spektrum
- Analýza nezávislých komponent (ICA)
- Průměrování
- Segmentace
- Podvzorkování
- Průměrování časových oken
- Společný prostorový vzor (CSP)
- Lineární diskriminační analýza (LDA)
- Metoda podpůrných vektorů (SVM)
- Graf průběhů amplitudy
- Mapa rozložení napětí na povrchu hlavy
- Časově-frekvenční mapy

Některé metody jsou dohledatelné a implementované v Pythonu, ale nejsou anotované v *NeuroPype*. To jsou tyto uzly:

- BrainVision formát: načtení/uložení

- Diskrétní vlnková transformace
- Matching pursuit
- Dopředná neuronnová síť
- Konvoluční neuronové sítě
- Rekurentní neuronové sítě

Anotace Orange

V nástroji *Orange* se jedné atomické části workflow říká widget. Widget je většinou dílčí proces, který může navazovat na další widgety, kde výstupem tohoto widgetu je dále vstup widgetu nadcházejícího. Po takovém spojení více widgetů dostaneme workflow. Jde o podobný systém jako u uzlů u *Dynamic Signal Processing Workflow Designeru*. Pro vytvoření jednotlivého widgetu je nejprve potřeba importovat potřebné knihovní moduly pro tvorbu widgetů. Anotací u nástroje *Orange* je definice důležitých částí kódu, jako je například vstup, výstup nebo děděná třída. V základu by měl být importovaný z knihovny `Orange` *OWWidget*, *Input* a *Output*. *OWWidget* je hlavní komponenta pro implementaci widgetu ve workflow. Widget musí dědit od třídy *OWWidget*. Pomocí modulů *Input* a *Output* definujeme vstup a výstup widgetu.

```
from Orange.widgets.widget import OWWidget, Input, Output
```

Každý widget definuje svá metadata, která obsahují popis widgetu, jeho název a specifikace vstupu a výstupu (pokud nějaký vstup nebo výstup je). V definici vstupu i výstupu musí být název a typ proměnné. Metadata můžou také obsahovat cestu ke zdrojovému souboru ikony, pro její zobrazení. V následujícím kódu je příklad metadat widgetu popsaných výše.

```
class Adder(OWWidget):
    name = "Add two integers"
    description = "Add two numbers"
    icon = "icons/add.svg"

    class Inputs:
        a = Input("A", int)
        b = Input("B", int)

    class Outputs:
```

```
        sum = Output("A + B", int)
```

Navíc každý vstup musí mít svou vlastní metodu jako handler.

```
@Inputs.a
    def set_A(self, a):
        """Set the ‘A‘ input."""
        self.A = a

    @Inputs.b
    def set_B(self, b):
        """Set the ‘B‘ input."""
        self.B = b
```

Výstup je pak přenesen tímto způsobem:

```
self.Outputs.sum.send(self.a + self.b)
```

Ani zde samozřejmě nechybí algoritmus/logika funkce daného widgetu, bez které widget nemá význam. Pro využití grafického uživatelského rozhraní pro widget je vhodné použít modul `Orange.widgets.gui` [3].

Nevýhoda *Orange* je, že pro `BCI` experiment zde nejsou žádné užitečné implementované metody. Metody jsou ale dohledatelné a implementované v Pythonu, jen nejsou anotované v nástroji *Orange*.

Anotace stejné metody

Pro porovnání anotací jednotlivých nástrojů jsem zdokumentoval anotaci stejné metody pro všechny nástroje. Pro tento příklad jsem zvolil metodu *Diskrétní vlnkové transformace* (Discrete wavelet transform).

U nástroje **Dynamic Signal Processing Workflow Designer** není potřeba vytvářet novou třídu, protože Diskrétní vlnková transformace již je v jazyce `Java` implementovaná a v nástroji anotovaná.

Zde je nejprve použit **BlockType** jako anotace třídy *WaveletTransformBlock* se složkou *family* – `FeatureExtraction` a *runAsJar* nastaveným jako `true`.

```
@BlockType(type="WaveletTransformBlock", family =
    "FeatureExtraction", runAsJar = true)
```

```
public class WaveletTransformFeatureExtraction implements
    IFeatureExtraction, Serializable {
```

Pro vstup má metoda **BlockInput** se jménem vstupu *EEGData* a typem *EEGDataList*.

```
@BlockInput(name = "EEGData", type = "EEGDataList")
private EEGDataPackageList epochs;
```

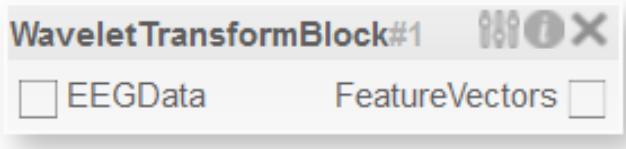
Pro výstup má **BlockOutput** se jménem výstupu *FeatureVectors* a typem *List<FeatureVector>*. V tomto výstupním listu je aproximovaný i detailový koeficient Diskrétní vlnkové transformace.

```
@BlockOutput(name = "FeatureVectors", type = "List<FeatureVector>")
private List<FeatureVector> featureVectors;
```

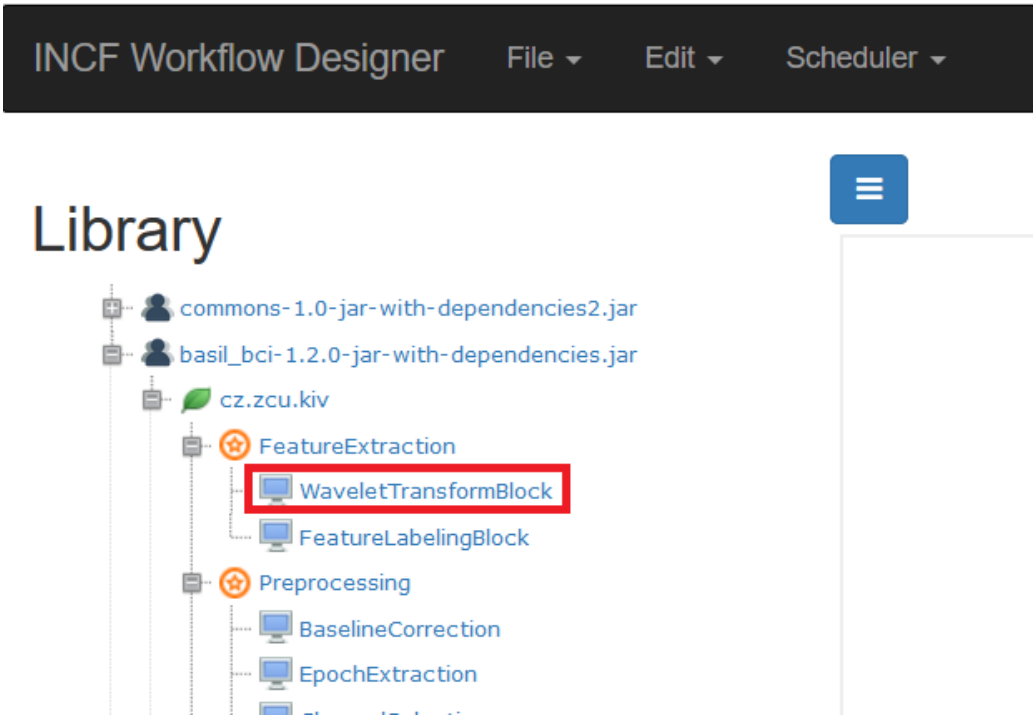
Poslední blok metody `WaveletTransformBlock` je blok spuštění **BlockExecute**.

```
@BlockExecute
public void process(){
    List<EEGDataPackage> inputDataPackages =
        epochs.getEegDataPackage();
    for (EEGDataPackage dataPackage: inputDataPackages) {
        FeatureVector outputVector = extractFeatures(dataPackage);
        this.featureVectors.add(outputVector);
    }
}
```

Výsledkem je blok třídy **WaveletTransformBlock** (obrázek 4.6 na straně 30) zařazený v složce *FeatureExtraction* (obrázek 4.7 na straně 30).



Obrázek 4.6: Blok třídy Diskrétní vlnkové transformace



Obrázek 4.7: Zařazení třídy Diskrétní vlnkové transformace

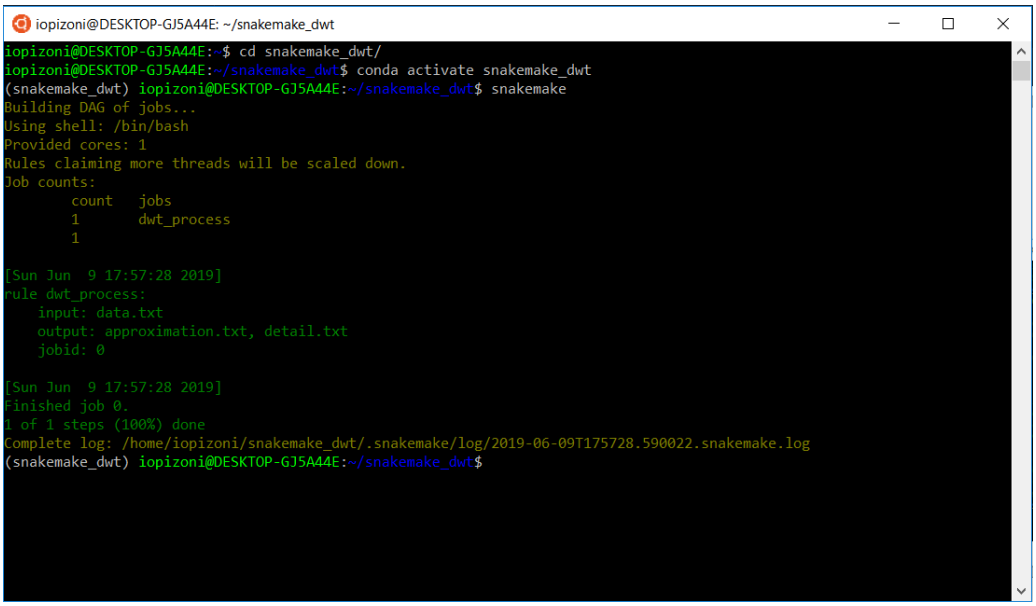
Snakemake je nástrojem pracujícím podle pravidel souboru `Snakefile`, nejprve jsem tedy vytvořil právě tento soubor. `Snakefile` má pouze jediné pravidlo a tím je `dwt_process`. Zde se nejprve načte vstupní soubor `data.txt` v sekci *input*, pak výstupní soubory `approximation.txt` a `detail.txt` v sekci *output* a poté skript se souborem `dwt.py` v sekci *script*. Výstupní soubory jsou dva, protože Diskrétní vlnková transformace má dvě výstupní hodnoty – aproximovaný a detailový koeficient.

```
rule dwt_process:
    input:
        "data.txt"
    output:
        "approximation.txt",
        "detail.txt"
    script:
        "dwt.py"
```

Pro vlnkové transformace v jazyce Python existuje užitečná knihovna `PyWavelets` [7], kterou jsem použil právě ve skriptu `dwt.py`. Tuto knihovnu jsem nainstaloval a poté ve skriptu importoval. Skript obsahuje algoritmus pro načtení dat ze souboru `data.txt` a poté jejich použití pro metodu *dwt* z knihovny `PyWavelets`. Výstupem metody *dwt* je aproximovaný a detailní koeficient. Aproximovaný koeficient je uložen v souboru `approximation.txt` a detailní koeficient pak v souboru `detail.txt`. V následujícím kódu se nejprve importuje knihovna `pywt`, poté se načte vstupní soubor, na data ze vstupního souboru se aplikuje metoda *dwt* a výsledky se vypíší do prvního a druhého výstupního souboru.

```
import pywt

fIn = open(snakemake.input[0], "r");
data = fIn.read().split(" ");
fIn.close();

cA, cD = pywt.dwt(data, 'db1');

fOut1 = open(snakemake.output[0], "a");
fOut1.write("cA = {}\n".format(list(cA)));
fOut1.close();

fOut2 = open(snakemake.output[1], "a");
fOut2.write("cD = {}\n".format(list(cD)));
fOut2.close();
```

Průběh použití *Diskrétní vlnkové transformace* pomocí nástroje **snakemake** je uveden na obrázku 4.8 na straně 32.



Obrázek 4.8: Průběh diskrétní vlnkové transformace v nástroji snakemake

U nástroje **NeuroPype** uzel *Diskrétní vlnkové transformace* anotovaný není. Musel jsem tedy uzel anotovat sám a do nástroje ho přidat.

Zde jsem opět použil knihovnu `PyWavelets` [7] pro metodu *dwt*.

Uzel *Diskrétní vlnkové transformace* u nástroje *NeuroPype* obsahuje, kromě povinných importů, popisu, konstruktoru a dědění třídy `Node`, jeden vstup a dva výstupy pro aproximovaný a pro detailový koeficient.

```
data = DataPort(Packet, "Data to process", IN)
outdata1 = DataPort(Packet, "Output data of approximation
    coefficient.", OUT)
outdata2 = DataPort(Packet, "Output data of detail coefficient.",
    OUT)
```

Pro funkčnost uzlu je potřeba určit *Wavelet family*, k tomu slouží atribut *family*, který zadá uživatel v GUI uzlu.

```
family = StringPort('db1', """Wavelet transform to use.
        """, verbose_name='Wavelet family')
```

Nakonec je zde hlavní logika uzlu, která je díky knihovně `PyWavelets` poměrně jednoduchá.

```
@data.setter
    def data(self, v):
        (cA, cB) = pywt.dwt(v, 'db1')
        self._outdata1 = cA
        self._outdata2 = cB
```

Ani u posledního nástroje **Orange** metoda *Diskrétní vlnková transformace* implementovaná není, tedy i zde jsem ji musel implementovat a anotovat sám opět s použitím knihovny `PyWavelets` [7]. Po importování potřebných knihoven včetně `PyWavelets` (`import pywt`) jsem založil třídu, která dědila od `OWWidget` a vyplnil název a popis.

```
class OWDWT(OWWidget):
    name = "Discrete wavelet tranformation"
    description = "Computes approximation and detail coefficient."
```

Poté bylo potřeba definovat vstup a výstup.

```
class Inputs:
    data = Input("Epoch data", mne.Epochs)

class Outputs:
    approximation = Output("Approximation coefficient",
        numpy.ndarray)
    detail = Output("Detail coefficient", numpy.ndarray)
```

Každý vstup však musí být zpracován metodou, tzv. handlerem.

```
@Inputs.data
    def set_input_data(self, input_data):
        """Initializes and modifies the input data and then sends
           the data on the output."""
        self.data = input_data
        if self.data is not None:
            self.data = self.data.copy()
            self.dwt()
        self.commit()
```

a po dosažení výsledků odeslat výstup.

```
def commit(self):
        self.Outputs.approximation.send(self.data_approximation)
        self.Outputs.detail.send(self.data_detail)
```

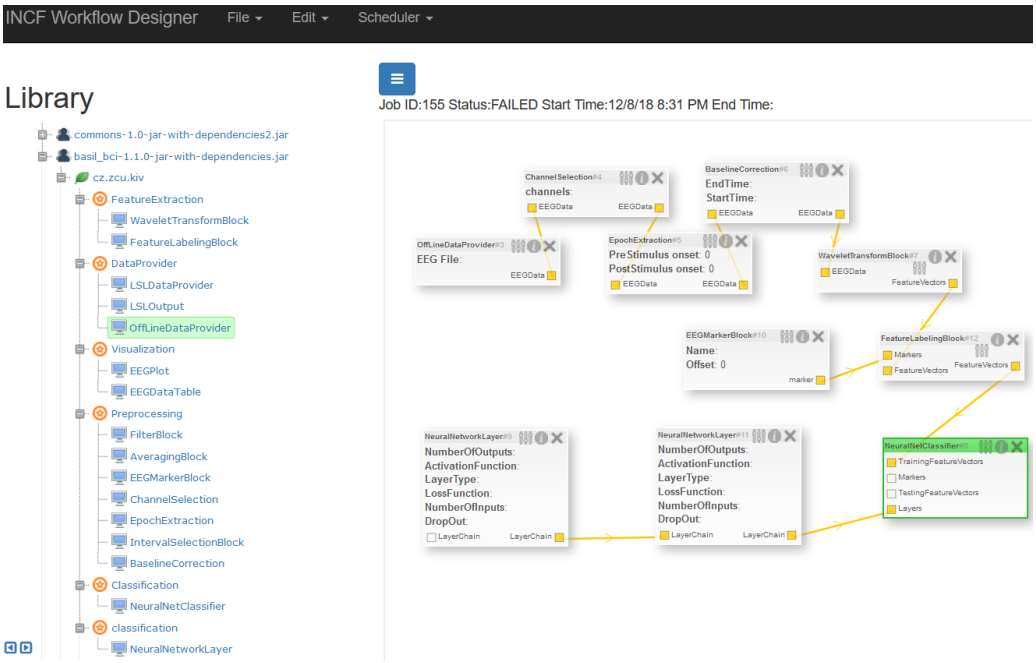
Nechybí samozřejmě použití knihovny `pywt` pro získání výsledků *Diskrétní vlnkové transformace*.

```
def dwt(self):
        for i in range(self.degrees):
            self.data_approximation, self.data_detail =
                pywt.dwt(self.data, self.wave_family)
            self.data = self.data_approximation
```

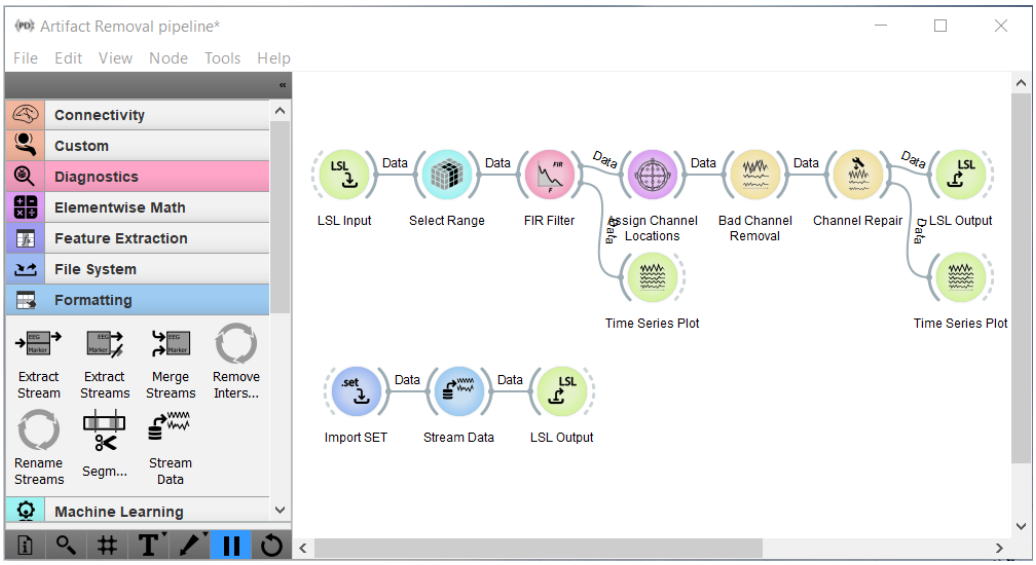
Widget navíc obsahuje grafické rozhraní implementované uvnitř třídy, ale pro porovnání anotace nástrojů ho zde není potřeba uvádět.

4.2.2 GUI

Každý z nástrojů má odlišné grafické uživatelské rozhraní. U nástroje **Snakemake** je jediným grafickým rozhraním DAG - řízený acyklický graf nebo jiný vygenerovaný graf. U **Dynamic Signal Processing Workflow Designer** je to lepší díky blokům, které mohou hezky znázornit dané workflow a nabídky, z které jde metody přetáhnout do workflow stylem *drag and drop*. GUI *Dynamic Signal Processing Workflow Designeru* je vidět na obrázku 4.9 na straně 35. Zdaleka nejkvalitnější GUI má **Orange** a **NeuroPype**, který je na nástroji *Orange* postavený. Stejně jako u *Dynamic Signal Processing Workflow Designer* můžou uzly být pomocí metody *drag and drop* přetáhnouté do okna s workflow. Většina uzlů má svojí jedinečnou ikonku, a je tedy snazší rozeznat od sebe uzly na první pohled. Poměrně široká je zde i hlavní nabídka s případnou pomocí, exportu, atd. Obrázek GUI *NeuroPypu* 4.10 je na straně 35 a obrázek GUI nástroje *Orange* 4.11 je na straně 36.



Obrázek 4.9: Dynamic Signal Processing Workflow Designer GUI

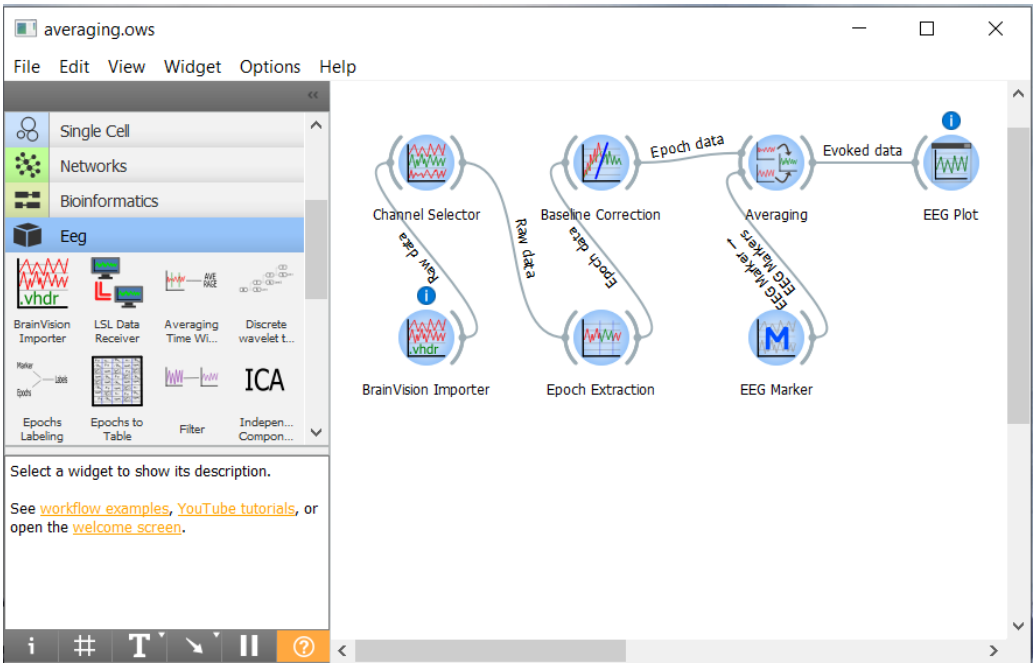


Obrázek 4.10: NeuoPype GUI

Hodnocení nástrojů				
Hodnotící kritéria	DSPWD*	Snakemake	NeuroPype	Orange
Anotace	3	1	4	4
Vybaveni	3	0	4	1
Dostupnost	3	0	5	5
GUI	3	0	4	4
Spolupráce	3	1	4	5
Licenční podmínky	5	5	3	5
Součet	20	7	24	24

*Dynamic Signal Processing Workflow Designer

Tabulka 4.1: Hodnocení nástrojů



Obrázek 4.11: Orange GUI

4.3 Výběr nástroje

Po zhodnocení jednoduchosti a efektivitě anotace jednotlivých nástrojů, vybavení (co se týče anotovaných metod), dostupnosti neanotovaných metod, GUI, celkové spolupráce nástroje s uživatelem a licenčních podmínek je nejlepší nástroj pro vypracování mého workflow *Orange*.

Anotace metod je vcelku snadná u *Dynamic Signal Processing Workflow*

Designeru i u *NeuroPype* a *Orange*. U *Snakemaku*, jak již bylo zmíněno, je trochu složitější. Co se týče vybavení metod, tak je na tom zdaleka nejlépe *NeuroPype*, který již obsahuje téměř všechny metody, které jsou zapotřebí a v dostupnosti neanotovaných mají výhodu jak *Snakemake*, *NeuroPype*, tak *Orange* díky oblíbenosti Pythonu. I v GUI je *NeuroPype* s nástrojem *Orange* lepší. Co se týče celkové spolupráce nástroje s uživatelem, tak s nástroji, se kterými se lépe pracovalo, jsou *Orange* a *NeuroPype*. Nicméně *NeuroPype*, na rozdíl od *Orange*, má některé uzly pouze v binární podobě, a nelze se tedy podívat do jejich kódu. S *Dynamic Signal Processing Workflow Designerem* byla práce složitější při testování testovacího workflow a u *Snakemaku* byly velké problémy už při instalování nástroje, které se mi podařily vyřešit až po dlouhé době. Problém u *NeuroPypeu* nastává v případě licence. Zatímco *Snakemake*, *Dynamic Signal Processing Workflow Designeru* a *Orange* je open-sourceový nástroj, licenci k *NeuroPypeu* je třeba zakoupit nebo si zažádat o *NeuroPype Suite Academic Edition* pro studenty, akademiky a výzkumníky. Verze *NeuroPype Suite Academic Edition* je ale čistě pro nekomerční účely, což je v tomto případně důležité, protože výsledek této práce by měl mít možnost použití i pro účely komerční.

Pro lepší přehled hodnocení jsem vytvořil tabulku (4.1 na straně 36) *Hodnocení nástrojů* s obodovanými kritériemi od 0 do 5 bodů podle mého názoru, kde 0 bodů představuje špatné nebo žádné podmínky pro dané kritérium a 5 bodů výborné podmínky. Nejvíce bodů má nástroj *NeuroPype* a *Orange*, nicméně kvůli licenčním podmínkám dostal přednost *Orange*.

5 Analýza a design vybraného nástroje

Vybraným nástrojem pro tuto práci je nástroj **Orange** pro jeho snadné použití, dostupnost a licenční podmínky. Pro spojování funkčních metod slouží v nástroji **Orange** komponenty, zvané widgety. Widgety představují některé samostatné funkce a poskytují grafické uživatelské rozhraní. Komunikují spolu a předávají objekty prostřednictvím komunikačních kanálů. Každý widget patří do určité kategorie.

Tento nástroj vypadá graficky velice pěkně a zobrazení jednotlivých komponent je rozmístěné tak, aby je mohl uživatel snadno najít a použít. Na obrázku 5.1 na straně 39 je zobrazen ilustrační příklad nástroje, kde jsou v červeném obdélníku zvýrazněny části nástroje pro práci s ním.

V levém horním rohu je název workflow. Na obrázku je označený číslem jedna.

Menu nástroje označené dvojkou obsahuje nabídku *File*, *Edit*, *View*, *Options* a *Help*. Nejdůležitější je *File*, kde lze ukládat nebo nahrávat hotová workflow, *Options*, kde je možné přidat nový balík widgetů – tzv. add-on a případně *Help*, ve kterém je možné dohledat příklady workflow, pomocná videa anebo najít dokumentace k jednotlivým balíkům Orange.

Číslem tři jsou označené balíky widgetů. Každý takový balík má název, který určuje, do jaké kategorie widgety patří.

Po rozkliknutí balíku (na obrázku *Eeg*) se otevře jeho obsah, tedy widgety, které sem patří. Tento obsah je na obrázku označený číslem čtyři.

Ve spodní části, označené číslem pět, se nachází stručný popis posledního navštíveného widgetu.

Pod ním se nachází rychlá pomocná nabídka. Zleva je tam tlačítko označené jako *i*, které zobrazuje popis workflow, a je ho zde také možné editovat. Další tlačítko je `#` a zarovnává widgety do mřížky. Na dalším tlačítku je T a je možné pomocí něj vkládat do workflow text, podobně jako další tlačítko s šipkou, které umožňuje vkládat šipky. Většinou pak šipka vede od textu k widgetu jako přiřazení. Předposledním tlačítkem je `||`, které zastaví tok dat, případně při opětovném stisknutí ho zase spustí. A posledním tlačítkem, kterým je `?`, se dá zavřít nebo otevřít stručný popis widgetu nad touto rychlou pomocnou nabídkou.

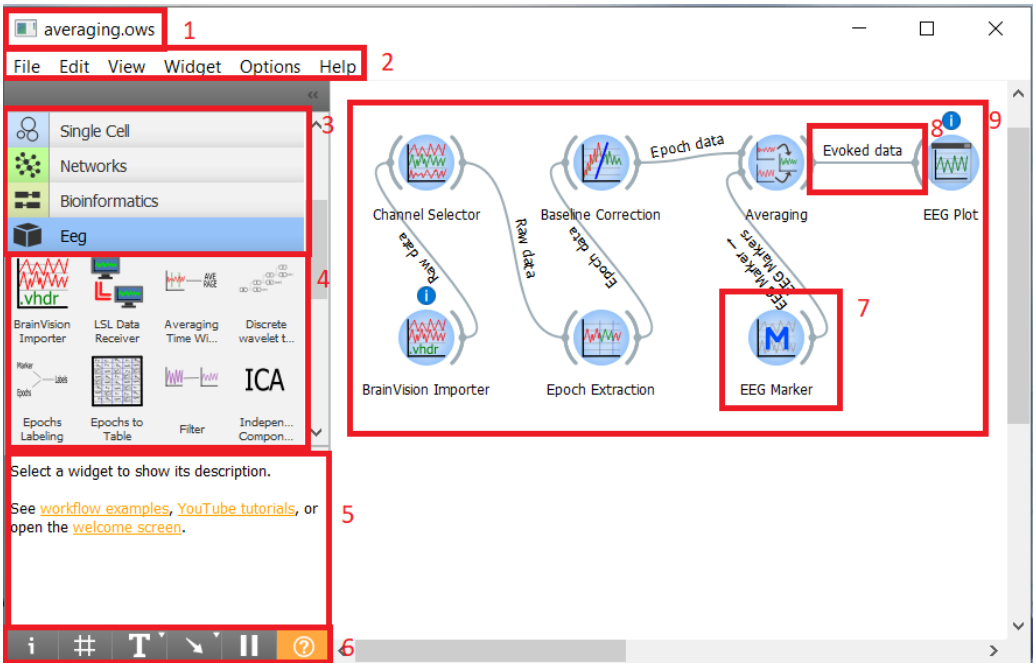
Pod číslem sedm je jednotlivý widget, který obsahuje název widgetu a

obsahuje vstup, výstup nebo obojí.

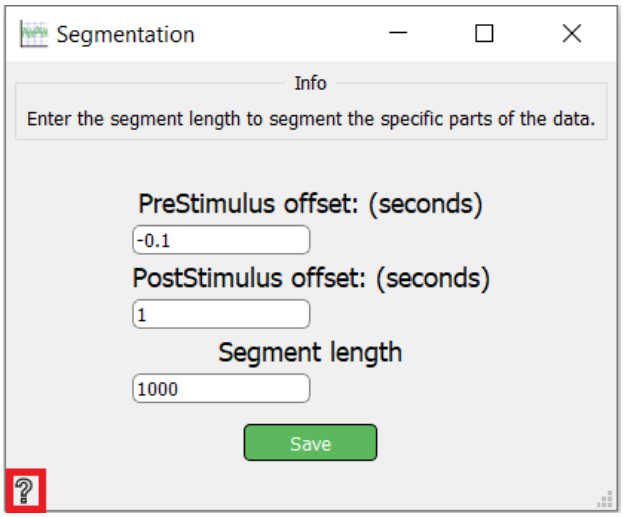
Číslem osm je označené spojení mezi dvěma widgety. Toto je vždy výstupem jednoho widgetu a vstupem jiného. Nad čárou spojující widgety je typ přenášených dat, která by zde měla protékat. Pokud je čára čárkovaná, pak tam žádná data zatím neprotékají, pokud je plná, tak ano.

Posledním označením části je číslo devět. Zde je označené celé workflow, tedy několik spojených widgetů.

Pro každý widget existuje dokumentace, kterou je možné načíst stiskem tlačítka otazníku v levém spodním rohu po otevření widgetu, jak je zvýrazněno na obrázku 5.2 na straně 40.



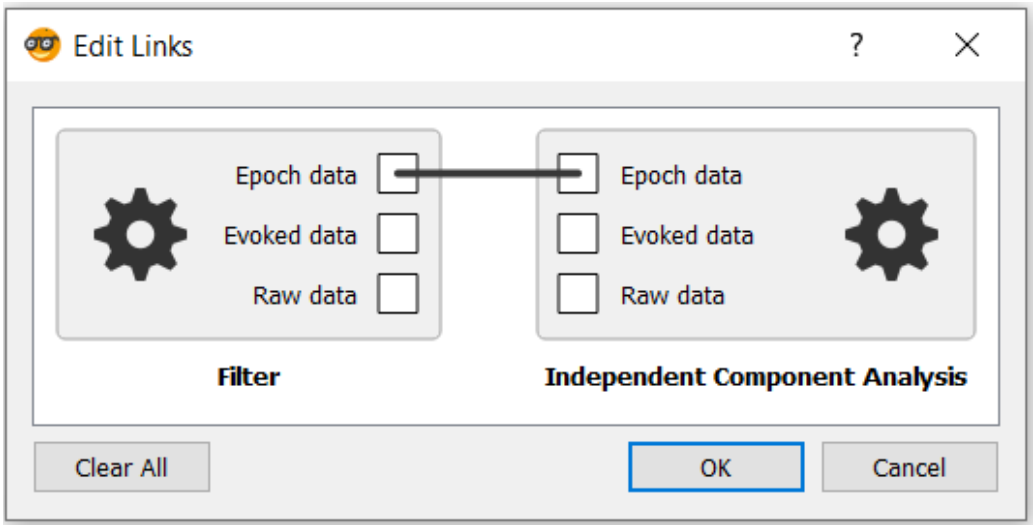
Obrázek 5.1: Ilustrační příklad nástroje Orange



Obrázek 5.2: Zobrazení dokumentace

Vytvoření workflow pro daný problém je v nástroji Orange opravdu snadné, pokud máme k dispozici potřebné widgety. Stačí widgety vybrat z nabídky. To uděláme buď kliknutím na ně nebo přetáhnutím na bílou plochu vpravo. Poté je třeba widgety pospojovat. Každý widget může mít dvě možnosti spojení, jedna je vlevo od widgetu a jde o vstupní data widgetu, druhá je vpravo a jde o výstupní data. Widget může mít i více vstupů nebo výstupů. Pokud widget patří právě do této skupiny s více vstupy nebo výstupy, je potřeba dávat pozor, jestli data tečou do správného vstupu/ze správného výstupu. Tedy pokud například chceme výstupní data z widgetu *Filter* použít jako vstupní data widgetu *Independent Component Analysis*, kde jak na výstupu widgetu *Filter*, tak na vstupu widgetu *Independent Component Analysis* mohou být data *Epochs*, *Evoked* a *Raw*, ale víme, že data jsou *Epochs*, dát si pozor, že je tak vybráno i ve spojení těchto widgetů ve workflow. Změnit spojení je možné po jeho rozkliknutí. Obrázek 5.3 nastavení spojení je na straně 41. Název dat, která jsou vybrána, je pak nad čárou spojení. Pokud je čára plná, pak byla data předána. Pokud je čárkovaná, pak data předána nebyla.

Vstupní widgety bývají widgety, které mají jen výstupní data. Do takových widgetů se často vkládají nějaké soubory jako vstupní data a spustí tím workflow. Naopak na konci workflow bývají widgety, které mají jen vstupní data a žádná výstupní. U takových widgetů je většinou nějaký vizuální výstup, jako je graf nebo tabulka.



Obrázek 5.3: Spojení widgetů Filter a Independent Component Analysis

6 Implementace

V této části jsou vysvětlené důležité struktury použité při implementaci a popsané implementované widgety pro zpracování eeg dat a widgety pomocné. Tyto widgety patří do kategorie *EEG*.

Při implementaci widgetů nástroje *Orange* jsem použil open-source balíček `MNE-Python` pro zpracování EEG dat v jazyce `Python`. Tento balíček obsahuje tři základní struktury pro práci s daty: **mne.io.Raw, mne.Epochs a mne.Evoked**. **Mne.io.Raw** jsou surová data načtená (v mém případě) z hlavičkového souboru s EEG daty *.vhdr*. Odpovídají plynulému EEG signálu. Data ve struktuře **Raw** jsou data s dvou-dimenzionální maticí *[kanál:čas]*, kde čas je úplný nepřetržitý signál EEG. **Raw** data je možné extrahovat na specifická časová okénka, která jsou obvykle uzamčena s ohledem na událost, jako je například vizuální podnět. Tato okénka se nazývají epochy a patří do struktury **mne.Epochs**. Tato struktura má matici třídimenzionální, protože zde přibyly epochy – *[epocha:kanál:čas]*. Třetí strukturou je **mne.Evoked**, evokovaná data. Evokovaná data jsou získána průměrováním epoch. Evokovaný objekt je obvykle konstruován pro každý subjekt a každou podmínku, ale lze jej také získat zprůměrováním seznamu evokovaných dat nad různými subjekty.

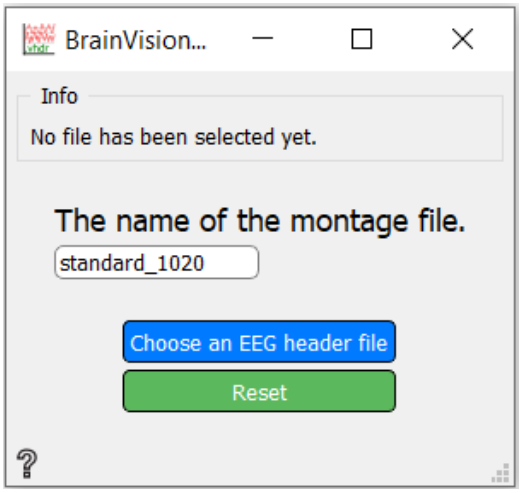
6.1 Widgety

Widgety pro nástroj *Orange* jsem implementoval v jazyce `Python`. Všechny widgety, které jsem v rámci této práce vytvořil, patří do kategorie *EEG*. Kategorie je instalována jako add-on do nástroje *Orange*. Add-on s kategorií *EEG* je součástí add-onu *Orange3-Eeg*. Některé widgety jsem převzal z projektu ZSWI-BCI a podle potřeby upravil. Jedná se o widgety *BrainVision Importer, LSL Data Reciever, Averaging, Baseline Correction, Channel Selector, EEG Marker, EEG Plot a Epoch Extraction* [10]. Všechny widgety mají svoji unikátní ikonu a svou dokumentaci, která je spolu s ikonami a celou kategorií *EEG* součástí add-onu *Orange3-Eeg*.

Celý projekt je na githubu: `https://github.com/Iopi/orange3-eeg`
Dokumentace widgetů je na stránce: `https://orange3-eeg.readthedocs.io/en/latest/`
Dále jsou popsané a vysvětlené jednotlivé widgety. Více informací o daném widgetu uživatel dostane v dokumentaci každého widgetu.

6.1.1 BrainVision Importer

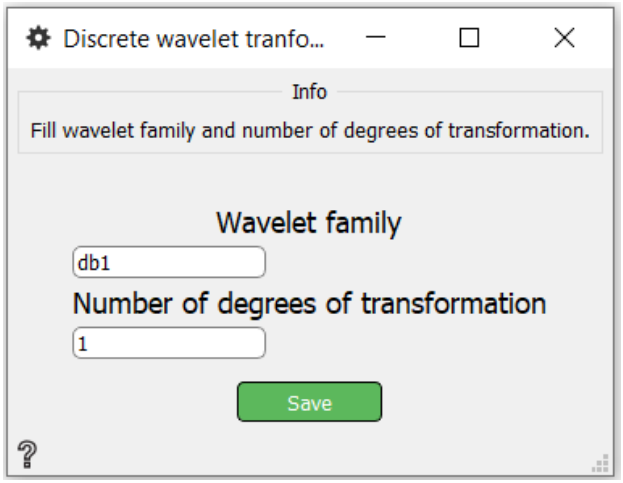
Widget **BrainVision Importer** čte data BrainVision z vybraného hlavičkového souboru EEG dat `.vhdr`. Tento widget je vstupní a nemá možný vstup od jiného widgetu. Vstupem je tedy právě soubor EEG dat. Výstupem tohoto widgetu jsou *Raw* data. Otevřený widget je zobrazen na obrázku 6.1 na straně 43. Kromě výběru souboru eeg dat je zde parametr pro soubor montáže elektrodové čepice. Oproti původnímu widgetu, jsem zde doplnil parametr pro soubor montáže, který je potřebný například u widgetu `Independent Component Analysis`. Změny se uloží tlačítkem *Save*.



Obrázek 6.1: Widget BrainVision Importer

6.1.2 Discrete Wavelet Transformation

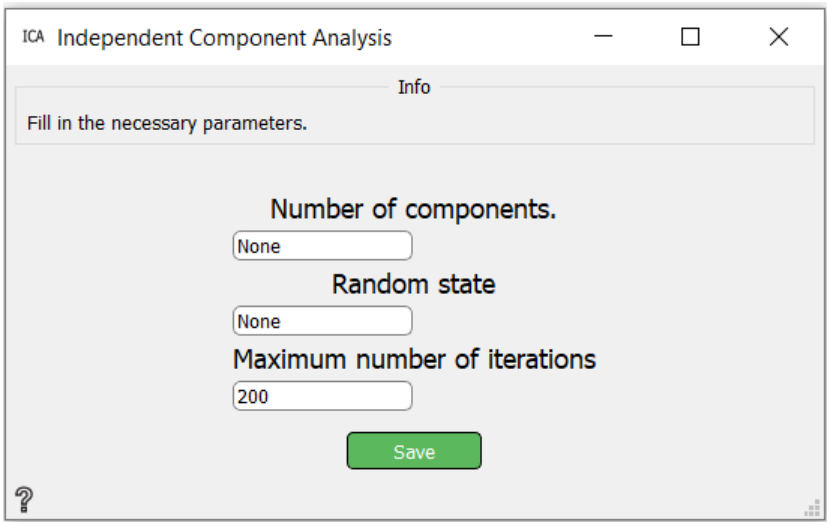
Widget diskrétní vlnkové transformace **Discrete Wavelet Transformation** je transformace odvozená z vlnkové transformace pro diskrétní vlnky. Vstupem je struktura *mne.Epochs*. Vypočítá aproximovaný a detailní koeficient, kde jsou oba výstupem widgetu. Otevřený widget je zobrazen na obrázku 6.2 na straně 44. Parametry jsou dva: *wavelet family* určující vlnkovou rodinu a *Number of degrees of transformation* zadávající počet stupňů transformace. Základní nastavení vlnkové rodiny je *db1* a počtu stupňů transformace *1*. Vlnková rodina udává typ vln pro výpočet. Po stisku tlačítka *Save* se uloží změny.



Obrázek 6.2: Widget Discrete Wavelet Transformation

6.1.3 Independent Component Analysis

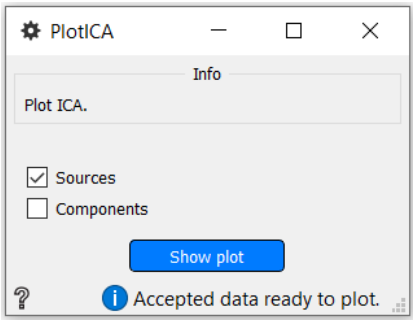
Widget **Independent Component Analysis** vypočítává nezávislou analýzu složek, neboli jde o metodu pro rozdělení vícerozměrného signálu na aditivní dílčí komponenty. Na vstupu můžou být struktury *mne.io.Raw*, *mne.Epochs* a *mne.Evoked*. Na výstupu je objekt nezávislé analýzy složek *mne.preprocessing.ICA*. Parametry jsou celkem tři. První je *Number of components*, který určuje počet hlavních komponent, které se během montáže předávají algoritmu ICA. Montáž je systém rozmístění elektrod na elektrodové čepici. První parametr nemá nastavenou žádnou základní hodnotu. Druhý parametr je *Random state* a určuje náhodný stav pro inicializaci odhadu ICA. Ani druhý parametr nemá nastavenou žádnou základní hodnotu. Třetí parametr je *Maximum number of iterations*. Tento parametr určuje maximální počet iterací během montáže a jeho základní hodnota je *200*. Uložit změny je možné tlačítkem *Save*. Otevřený widget je zobrazen na obrázku 6.3 na straně 45.



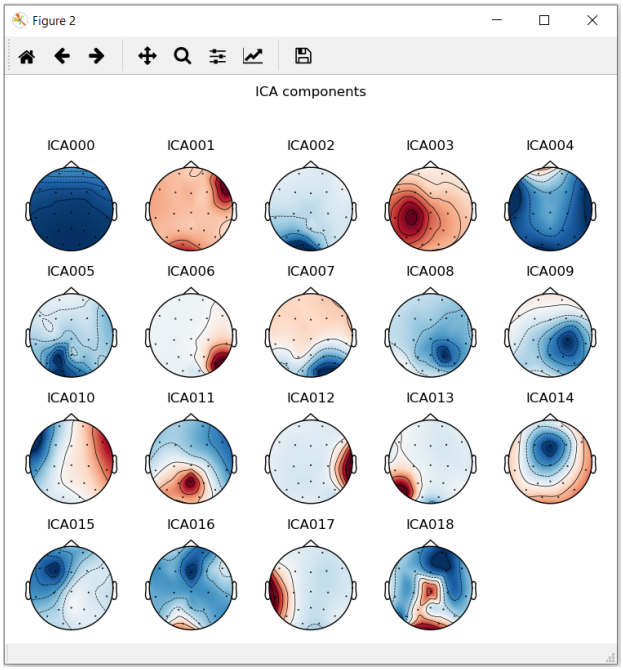
Obrázek 6.3: Widget Independent Component Analysis

6.1.4 Plot Independent Component Analysis

Pro vykreslení objektu *mne.preprocessing.ICA* z widgetu `Independent Component Analysis` je zde widget **Plot Independent Component Analysis**. Tento widget dokáže vykreslit zdroje i jednotlivé komponenty objektu *mne.preprocessing.ICA*. Vstupem je tedy objekt *mne.preprocessing.ICA* a struktury *mne.io.Raw*, *mne.Epochs* nebo *mne.Evoked*, která je stejná jako vstup do widgetu `Independent Component Analysis`. Možnosti jsou v grafickém uživatelském rozhraní jen zaškrtávací políčka pro vykreslení zdrojů (*Sources*) a komponent (*Components*). Tlačítkem *Show plot* se zobrazí vybrané vizualizace. ICA je citlivá na nízkofrekvenční drifty, a proto je důležité, aby byla data před montáží filtrována na horní propust. Otevřený widget je zobrazen na obrázku 6.4 na straně 45. Příklad vizualizace je na obrázku 6.5 na straně 46.



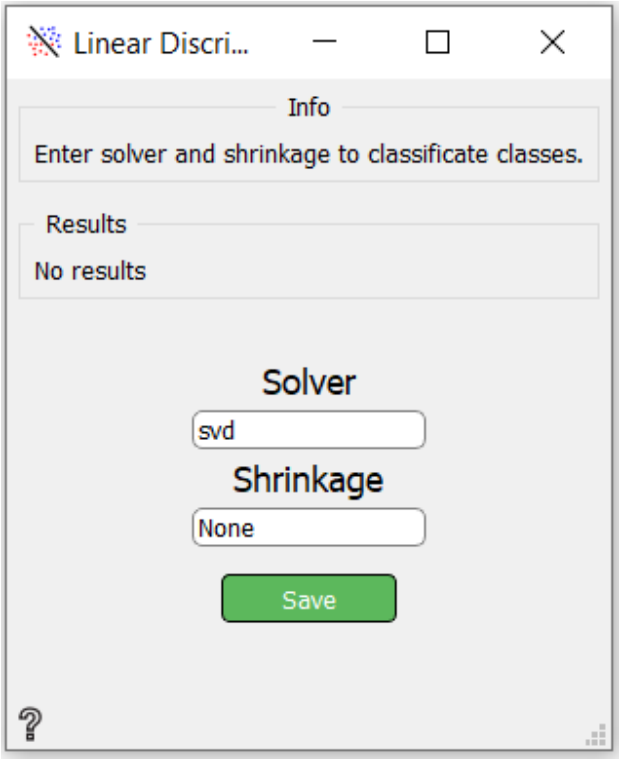
Obrázek 6.4: Widget Plot Independent Component Analysis



Obrázek 6.5: Příklad vizualizace widgetu Plot ICa

6.1.5 Linear Discriminant Analysis

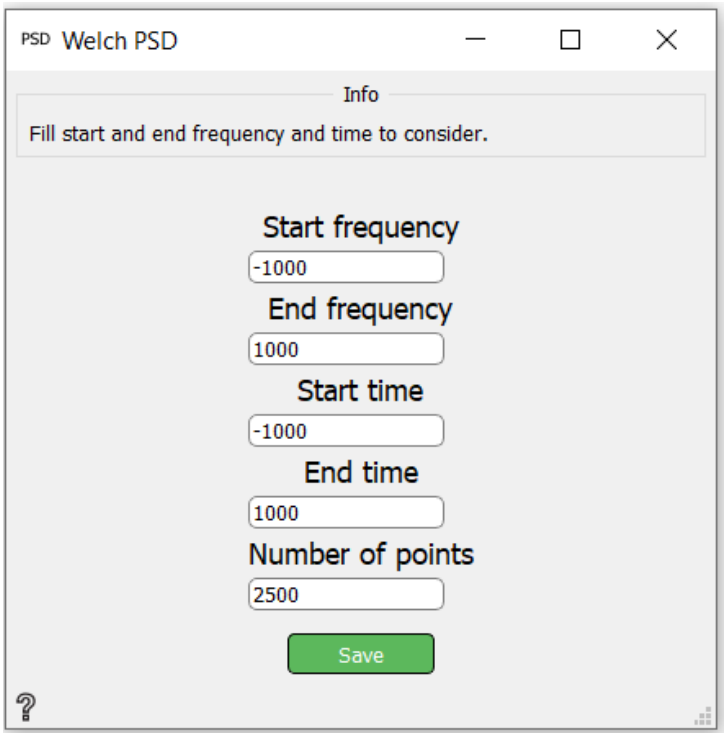
Jeden z klasifikačních widgetů je **Linear Discriminant Analysis**. **Linear Discriminant Analysis** je klasifikátor s lineární hranicí rozhodování generovaný přizpůsobením podmíněných hustot třídy datům a použitím Bayesova pravidla. Zde mi pomohla knihovna `sklearn`, z které jsem použil metodu `discriminant_analysis.LinearDiscriminantAnalysis`. Vstupy do tohoto widgetu jsou celkem čtyři. Jsou to trénovací data, jejich příznaky, testovací data a příznaky testovacích dat. Výstupem je *Accuracy*, tedy přesnost klasifikace všech tříd podle příznaků. Pokud jde o binární klasifikaci (klasifikace pouze dvou tříd), je výstupem i *Recall* a *Precision*. *Recall* je procento, které ukazuje kolik z relevantních položek je vybráno a *Precision* je procento, které ukazuje kolik z vybraných položek je relevantních. Tyto výstupní data se zobrazí i v otevřeném widgetu v kolonce *Results*. U tohoto widgetu jsou dva parametry: *Solver* se základní hodnotou *svd* (rozklad singulární hodnoty) a *shrinkage* s hodnotou *None*. *Shrinkage* je nástroj ke zlepšení odhadu kovariančních matic v situacích, kdy je počet tréninkových vzorků ve srovnání s počtem funkcí malý. Otevřený widget je zobrazen na obrázku 6.6 na straně 47.



Obrázek 6.6: Widget Linear Discriminant Analysis

6.1.6 Welch PSD

Welch PSD vypočte výkonovou spektrální hustotu (PSD) pomocí Welchovy metody. Vstupem do tohoto widgetu mohou být struktury *mne.io.Raw, mne.Epochs nebo mne.Evoked*. Výstupem je dvojice (`tuple` – v Pythonu slouží podobně jako `list`) prvků *psds* a *freqs*, kde *psds* představuje výkonové spektrální hustoty a *freqs* jsou frekvence signálu. U tohoto widgetu je pět parametrů. Prvním parametrem je *Start frequency*, minimální frekvence se základní hodnotou *-1000* Hz. Druhým parametrem je naopak *End frequency* – maximální frekvence se základní hodnotou *1000* Hz. Třetím a čtvrtým parametrem je minimální a maximální čas, *Start* a *End time* se základními hodnotami *-1000* a *1000* sekund. Při takových extrémních hodnotách si program sám upraví graf na optimální hodnoty. Tyto parametry jsou zde proto, kdyby uživatel chtěl, aby graf nepřesáhl jím zvolené hodnoty. Posledním parametrem je *Number of points*, který určuje počet bodů použitých při výpočtech Welch FFT. Jeho základní hodnota je *2500*. Uložit změny je možné tlačítkem *Save*. Otevřený widget je zobrazen na obrázku 6.7 na straně 48.

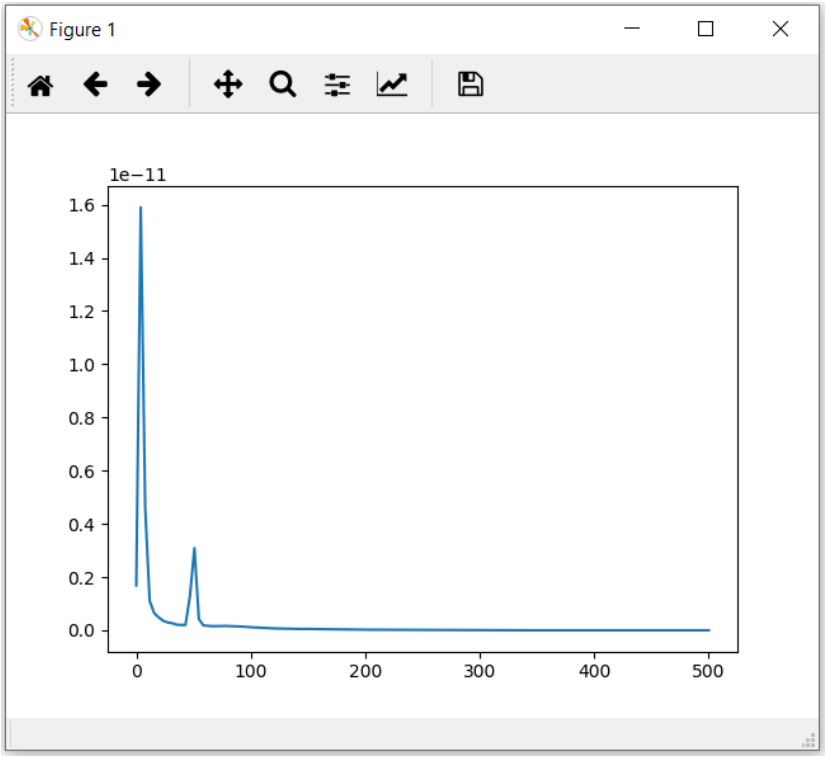


Obrázek 6.7: Widget Welch PSD

6.1.7 Plot Welch PSD

O zobrazení `Welch PSD` se stará widget **Plot Welch PSD**. Vsupní data je dvojice prvků *psds* a *freqs* získané z widgetu `Welch PSD`. Protože se jedná o výstupní uzel, výstupem je pouze vizuální vykreslení Welch PSD. Grafické uživatelské rozhraní pro tento widget obsahuje pouze tlačítko *Show plot*, pomocí kterého se zobrazí graf. Otevřený widget je zobrazen na obrázku 6.8 na straně 48. Na obrázku 6.9 na straně 49 je příklad výsledné vizualizace.

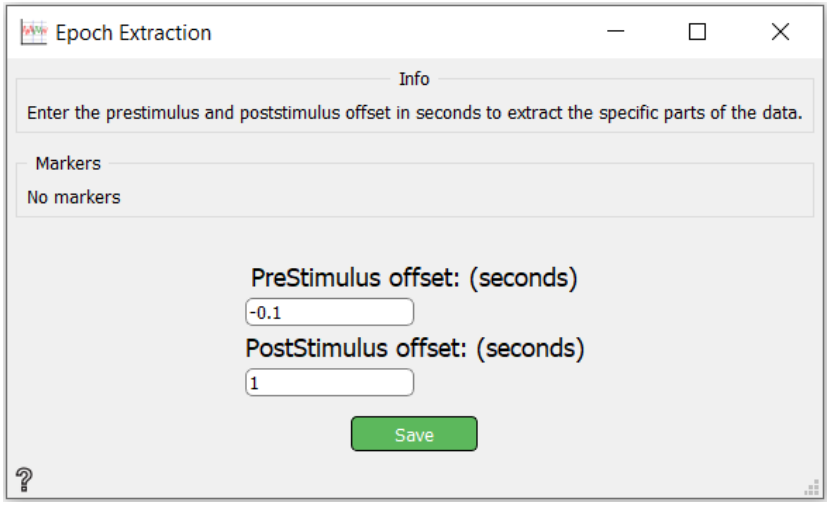
Obrázek 6.8: Widget Plot Welch PSD



Obrázek 6.9: Příklad vizualizace widgetu Plot Welch PSD

6.1.8 Epoch Extraction

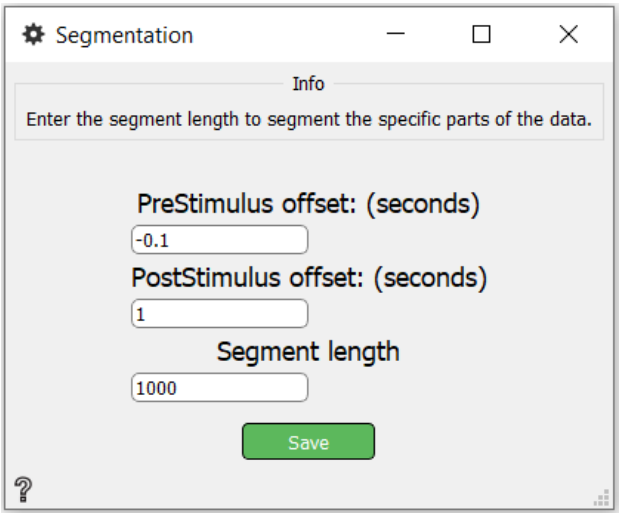
Widget **Epoch Extraction** extrahuje specifické epochy z dat EEG. Převádí tedy data ze struktury *mne.io.Raw* do struktury *mne.Epochs* podle stimulů. Vstupem jsou *Raw* data a výstupem *Epochy*. Na vstupu může být navíc jeden nebo vice markerů z widgetu `EEG Marker`, který určuje, že extrahované epochy budou pouze epochy s identifikačním číslem události podle přidaných identifikačních čísel markerů. Otevřený widget je zobrazen na obrázku 6.10 na straně 50. V grafickém uživatelském rozhraní se ukáží názvy vstupních markerů a jejich ofset. Parametry widgetu extrakce epoch jsou dva. Prvním je *PreStimulus offset*, který určuje čas začátku extrakce před událostí. A druhým je *PostStimulus offset*, který určuje čas konce extrakce po události. *PreStimulus* má základní nastavení hodnoty nastavené na *-0.1* a *PostStimulus* má základní hodnotu *1*. Uložit změny je možné tlačítkem *Save*.



Obrázek 6.10: Widget Epoch Extraction

6.1.9 Segmentation

Segmentation je widgetem, který extrahuje surová *mne.io.Raw* data na *Epochy*. Na rozdíl od widgetu `Epoch Extraction`, kde se epochy extrahují podle událostí, **Segmentation** rozděluje *Raw* data na *Epochy* podle délky segmentu. Vstupem je také struktura *mne.io.Raw* a výstupem struktura *mne.Epochs*. I první dva parametry (*PreStimulus a PostStimulus*) jsou stejné jako u widgetu `Epoch Extraction`. Je zde ještě třetí parametr *Segment length*, který udává délku segmentu a je nastaven na *1000*. Změny parametrů je možné uložit tlačítkem *Save*. Otevřený widget je zobrazen na obrázku 6.11 na straně 50.



Obrázek 6.11: Widget Segmentation

Zbylé widgety jsou popsané v příloze v sekci Implementované widgety.

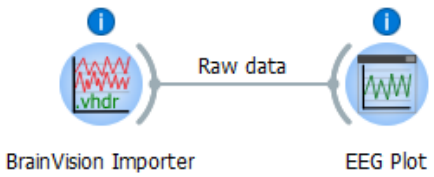
7 Testování

Pro ověření funkčnosti widgetů je potřeba je řádně otestovat. Widgety jsem postupně skládal do workflow a zhodnotil jsem, jestli výsledek dává vzhledem k zadaným widgetům a jejich parametrům smysl. Workflow při testování jsem použil tak, aby widgety logicky navazovali a výsledky tedy dávali smysl i v praxi.

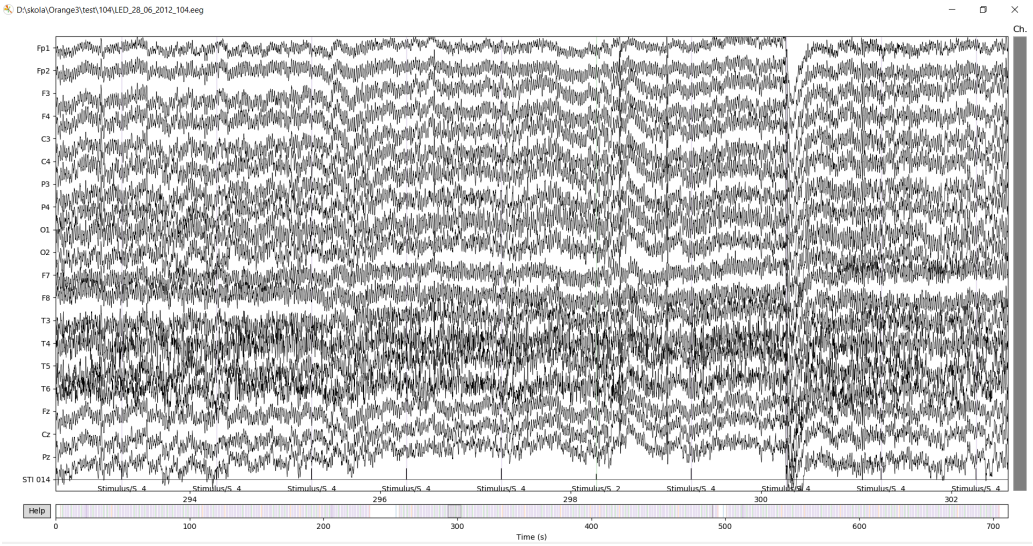
Při použití vstupního widgetu **BrainVision Importer** jsou vždy použity parametry název souboru montáže *standard_1020* a vstupní hlavičkový soubor s eeg daty *LED_28_06_2012_104.vhdr*, pokud nebude psáno jinak. V této montáži jsou elektrody pojmenovány a umístěny podle mezinárodního systému 10/20. Soubor *LED_28_06_2012_104.vhdr* je hlavičkový soubor, který obsahuje metadata a odkazy na soubory *.vmrk*, markerový soubor, a *.eeg*, binární datový soubor s eeg daty.

7.1 Testování widgetů BrainVision Importer a EEG Plot

Jako první test jsem zvolil co nejjednodušší workflow. Widget vstupu **BrainVision Importer**, který by měl nahrát hlavičkový soubor *.vhdr* s EEG daty. Výstup z tohoto widgetu navazuje na vstup druhého widgetu, který tvoří výstup celého workflow. Tímto widgetem je **EEG Plot**, který zobrazí základní graf *Raw* dat. Widget **EEG Plot** žádné parametry nemá. Obrázek 7.1 workflow je na straně 52 a jeho výsledný graf 7.2 je na straně 53. Tento test proběhl úspěšně, protože data ze souboru se skutečně vykreslila v grafu se všemi kanály.



Obrázek 7.1: Workflow testování widgetů BrainVision Importer a EEG Plot

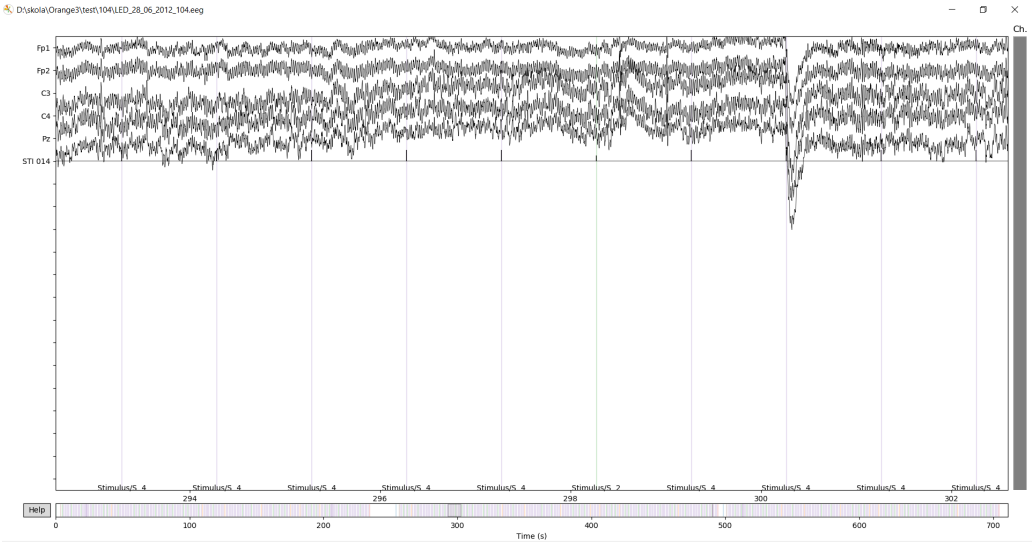


Obrázek 7.2: Výstup testování widgetů BrainVision Importer a EEG Plot

7.2 Testování widgetu Channel Selector

Druhým testem bylo otestování widgetu **Channel Selector**, který by měl ze všech kanálů ze vstupního souboru ponechat na výstupu jen kanály vybrané jako parametry widgetu. Vybrané kanály jsou *Fp1, Fp2, C3, C4 a Pz*. Oba další widgety `BrainVision Importer` a `EEG Plot` jsou stejné jako v prvním testovaní. Obrázek 7.3 workflow je na straně 53 a jeho výsledný graf 7.4 je na straně 54. Ve výsledném grafu jsou skutečně jen vybrané kanály, test tedy proběhl také úspěšně.

Obrázek 7.3: Workflow testování widgetů Channel Selector

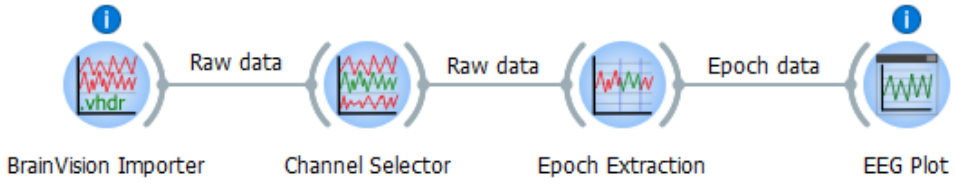


Obrázek 7.4: Výstup testování widgetů Channel Selector

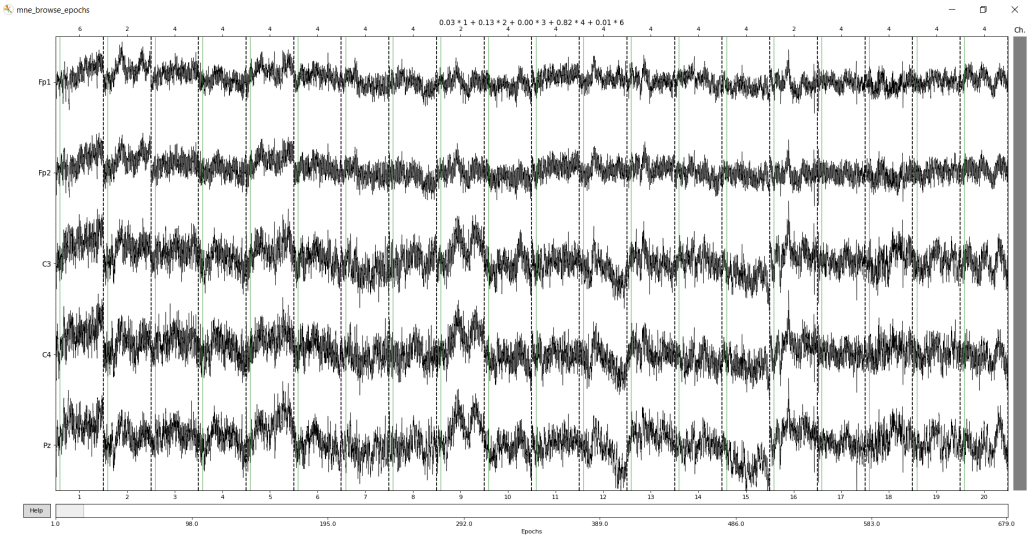
7.3 Testování widgetů Epoch Extraction a Segmentation

Pro vytvoření epoch (`Epochs`) ze surových dat (`Raw`) jsou v Eeg balíku dva widgety. Jeden z nich je **Epoch Extraction** a druhým je **Segmentation**. Oba widgety jsem testoval pomocí widgetů z předchozího testování a se stejnými parametry, tedy s `BrainVision Importer`, `Channel Selector` a `EEG Plot`.

Parametry **Epoch Extraction** jsou pro *PreStimulus -0.1* a *PostStimulus 1*. Surová data mají 679 událostí, vytvořených epoch by tedy mělo být 679. Obrázek 7.5 workflow je na straně 54 a jeho výsledný graf 7.6 je na straně 55. Výsledkem je skutečně 679 epoch, widget je tedy v pořádku.

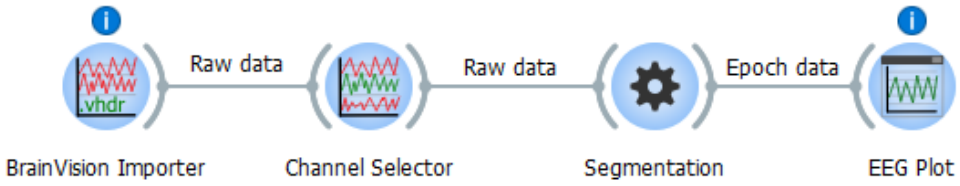


Obrázek 7.5: Workflow testování widgetu Epoch Extraction

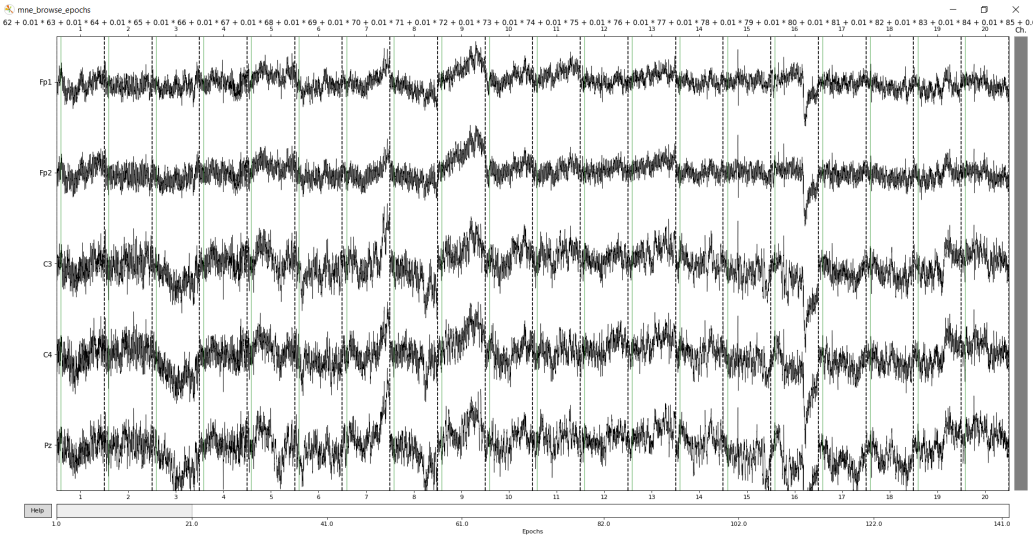


Obrázek 7.6: Výstup testování widgetu Epoch Extraction

Pro widget **Segmentation** jsou parametry stejné jako u **Epoch Extraction**. Je zde navíc parametr *Segment length* a ten je nastavný na *5000*. Počet epoch by měl tedy být přibližně délka signálu vydělená délkou segmentu, tedy *710940 / 5000*. Obrázek 7.7 workflow je na straně 55 a jeho výsledný graf 7.8 je na straně 56. Výsledkem je 141 epoch, je tedy v pořádku.



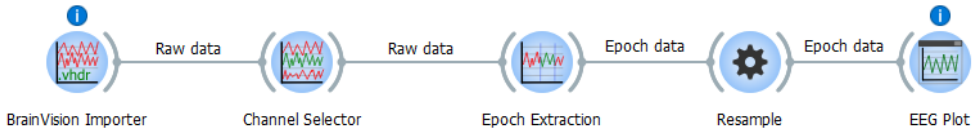
Obrázek 7.7: Workflow testování widgetu Segmentation



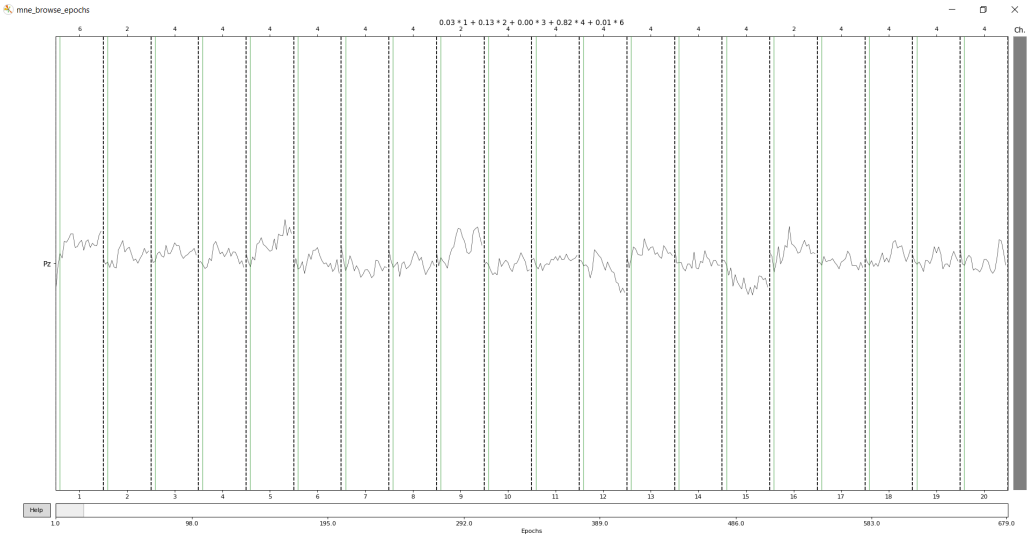
Obrázek 7.8: Výstup testování widgetu Segmentation

7.4 Testování widgetu Resample

Widget **Resample** je testován ve workflow s widgety `BrainVision Importer`, `Channel Selector` s jedním kanálem *Pz*, `Epoch Extraction` s parametry *PreStimulus -0.1* a *PostStimulus 1* a `EEG Plot`. **Resample** má *Sample rate to use 20*. Obrázek 7.9 workflow je na straně 56. Na výsledném grafu 7.10 na straně 57 je převzorkovaný graf, který je viditelně hladší než nepřevzorkovaný graf 16 na straně 90.



Obrázek 7.9: Workflow testování widgetu Resample

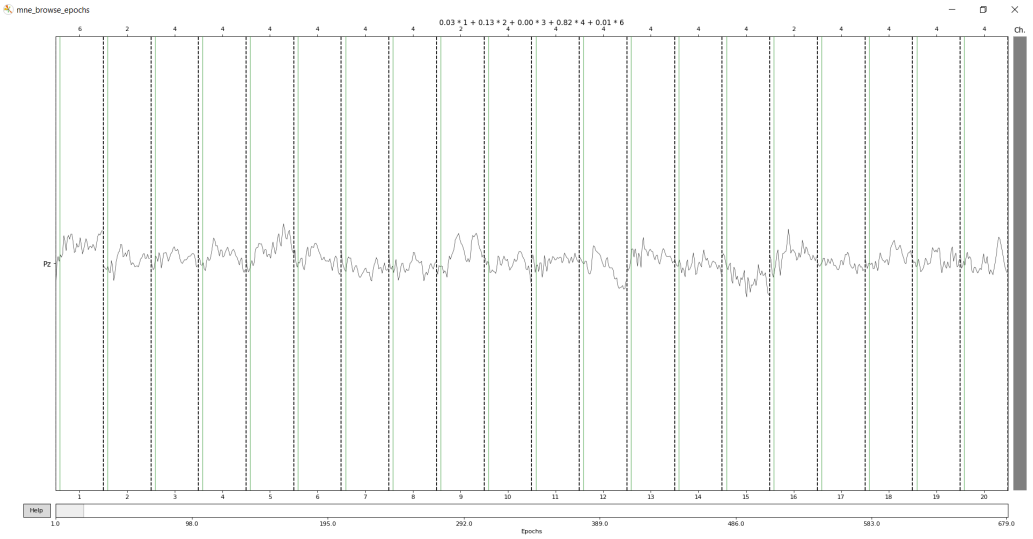


Obrázek 7.10: Výstup testování widgetu Resample

7.5 Testování widgetu Filter

Widget **Filter** je testován také ve workflow s widgety `BrainVision Importer`, `Channel Selector`, `Epoch Extraction` a `EEG Plot` se stejnými parametry jako u testování widgetu `Resample`. **Filter** má *The lower pass-band edge* nastaven na *0.1* a *The upper pass-band edge* na *12*. Obrázek 7.11 workflow je na straně 57. Na výsledném grafu 7.12 na straně 58 je filtrovaný graf. Je možné ho porovnat s nefiltrovaným grafem 16 na straně 90. Je vidět, že signál je nyní skutečně filtrovaný.

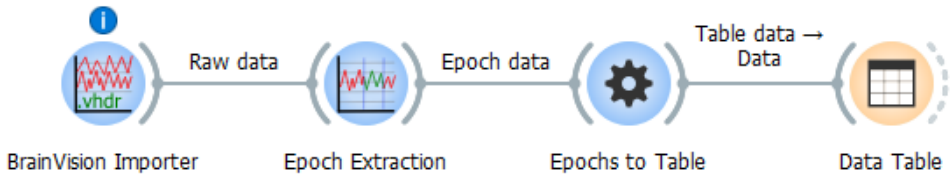
Obrázek 7.11: Workflow testování widgetu Filter



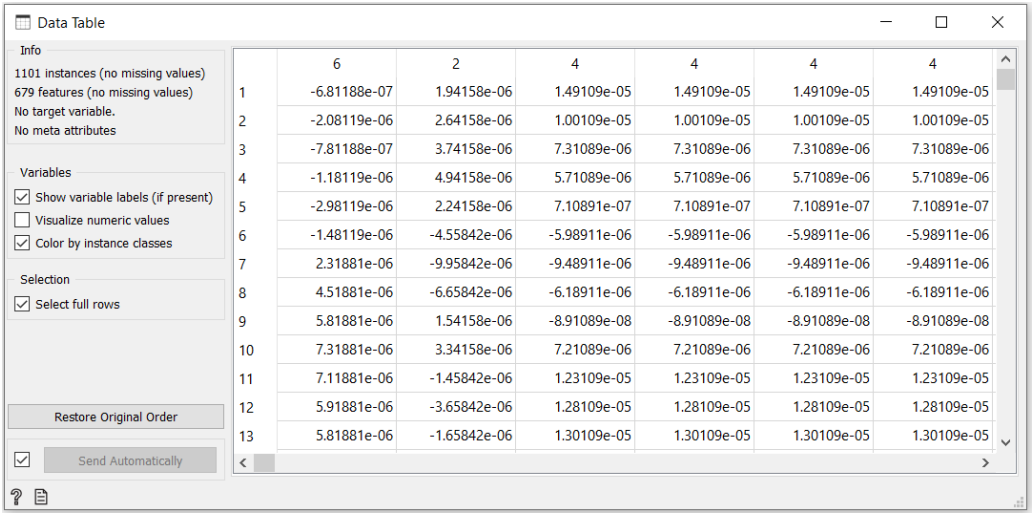
Obrázek 7.12: Výstup testování widgetu Filter

7.6 Testování widgetu Epochs to Table

V dalím testu testuji widget **Epochs to Table** pomocí widgetů `BrainVision Importer`, `Epoch Extraction` s parametry *PreStimulus -0.1* a *PostStimulus 1* a widgetu `Data Table`, který již byl součástí nástroje `Orange`. Widget **Epochs to Table** má parametr kanálu, který by se měl zobrazit ve výsledné tabulce, nastaven na *0*. Obrázek 7.13 workflow je na straně 58. Výsledkem je tabulka na obrázku 7.14 na straně 59, tedy test je úspěšný.



Obrázek 7.13: Workflow testování widgetu Epochs to Table



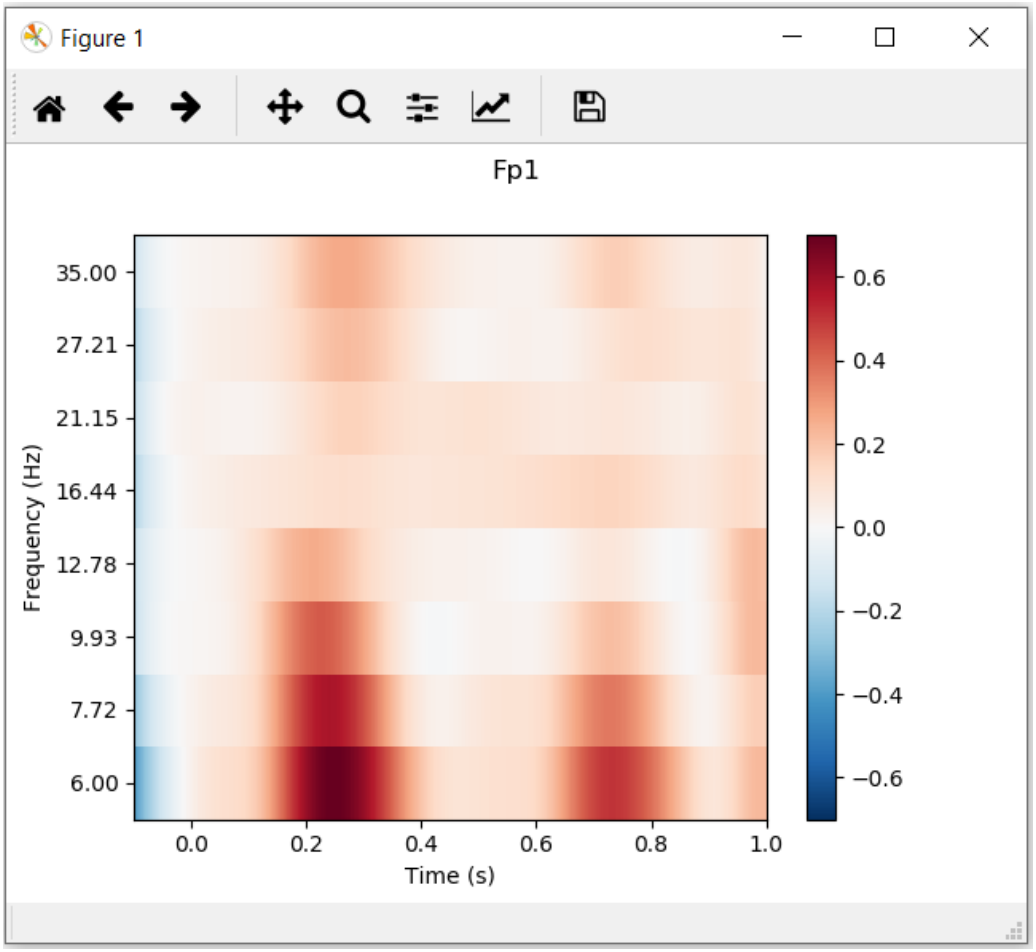
	6	2	4	4	4	4
1	-6.81188e-07	1.94158e-06	1.49109e-05	1.49109e-05	1.49109e-05	1.49109e-05
2	-2.08119e-06	2.64158e-06	1.00109e-05	1.00109e-05	1.00109e-05	1.00109e-05
3	-7.81188e-07	3.74158e-06	7.31089e-06	7.31089e-06	7.31089e-06	7.31089e-06
4	-1.18119e-06	4.94158e-06	5.71089e-06	5.71089e-06	5.71089e-06	5.71089e-06
5	-2.98119e-06	2.24158e-06	7.10891e-07	7.10891e-07	7.10891e-07	7.10891e-07
6	-1.48119e-06	-4.55842e-06	-5.98911e-06	-5.98911e-06	-5.98911e-06	-5.98911e-06
7	2.31881e-06	-9.95842e-06	-9.48911e-06	-9.48911e-06	-9.48911e-06	-9.48911e-06
8	4.51881e-06	-6.65842e-06	-6.18911e-06	-6.18911e-06	-6.18911e-06	-6.18911e-06
9	5.81881e-06	1.54158e-06	-8.91089e-08	-8.91089e-08	-8.91089e-08	-8.91089e-08
10	7.31881e-06	3.34158e-06	7.21089e-06	7.21089e-06	7.21089e-06	7.21089e-06
11	7.11881e-06	-1.45842e-06	1.23109e-05	1.23109e-05	1.23109e-05	1.23109e-05
12	5.91881e-06	-3.65842e-06	1.28109e-05	1.28109e-05	1.28109e-05	1.28109e-05
13	5.81881e-06	-1.65842e-06	1.30109e-05	1.30109e-05	1.30109e-05	1.30109e-05

Obrázek 7.14: Výsledná tabulka testování widgetu Epochs to Table

7.7 Testování widgetu Time Frequency Maps

Widget **Time Frequency Maps** je testován pomocí widgetů `BrainVision Importer` a `Epoch Extraction` s parametry *PreStimulus -0.1* a *PostStimulus 1*. **Time Frequency Maps** má parametry nastavené: *freqs* na *(6., 7.71912254, 9.930801079, 12.77618833, 16.43682721, 21.1463139, 27.2051647, 35.)*, *n_cycles* na *scale*, *baseline a* na *-0.5*, *baseline b* na *0*, *mode* na *logratio* a *channel* na *0*. Obrázek 7.15 workflow je na straně 59. Výsledkem, na obrázku 7.16 na straně 60, je časově-frekvenční mapa kanálu *Pz*. Výsledkem je časově frekvenční mapa, testování je tedy úspěšné.

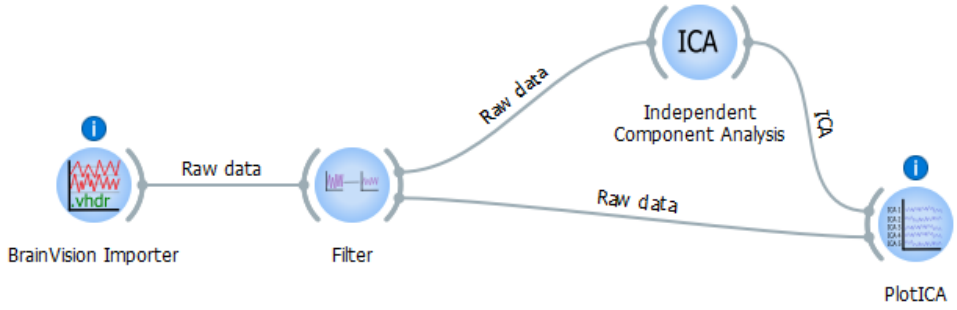
Obrázek 7.15: Workflow testování widgetu Time Frequency Maps



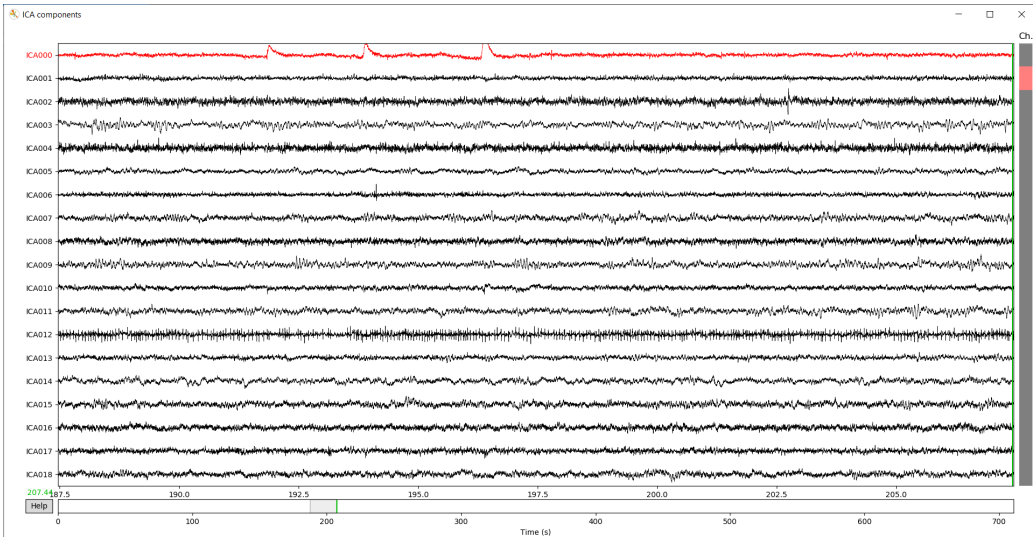
Obrázek 7.16: Výstup testování widgetu Time Frequency Maps

7.8 Testování widgetů Independent Component Analysis a Plot ICA

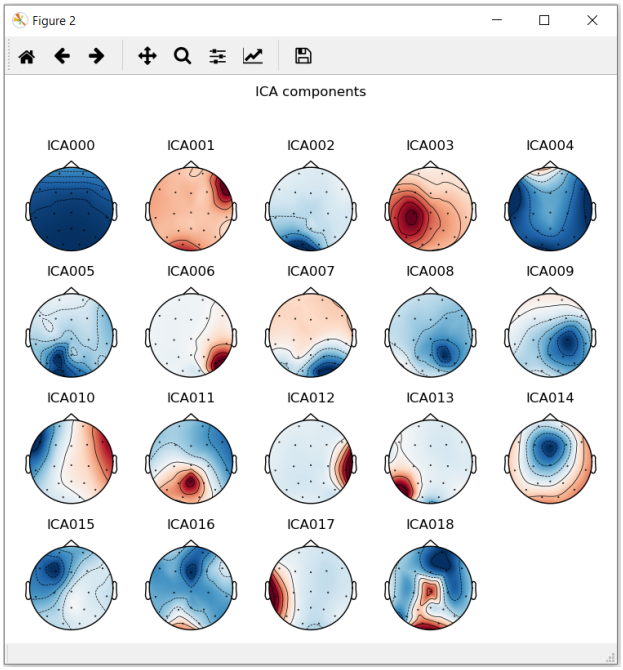
Dalším testováním je testování widgetů **Independent Component Analysis** a **Plot ICA**. Dalšími widgety v testovacím workflow jsou vstupní widget `BrainVision Importer` a `Filter` s *The lower pass-band edge* s hodnotou *1* a *The upper pass-band edge* s hodnotou *None*. Obrázek 7.17 workflow je na straně 61. Zobrazení zdrojů je na obrázku 7.18 na straně 61 a zobrazení komponent je na obrázku 7.19 na straně 62. Komponenty i zdroje byly správně vykresleny, test je tedy v pořádku.



Obrázek 7.17: Workflow testování widgetů Independent Component Analysis a Plot ICA



Obrázek 7.18: Vykreslení zdrojů widgetu Independent Component Analysis

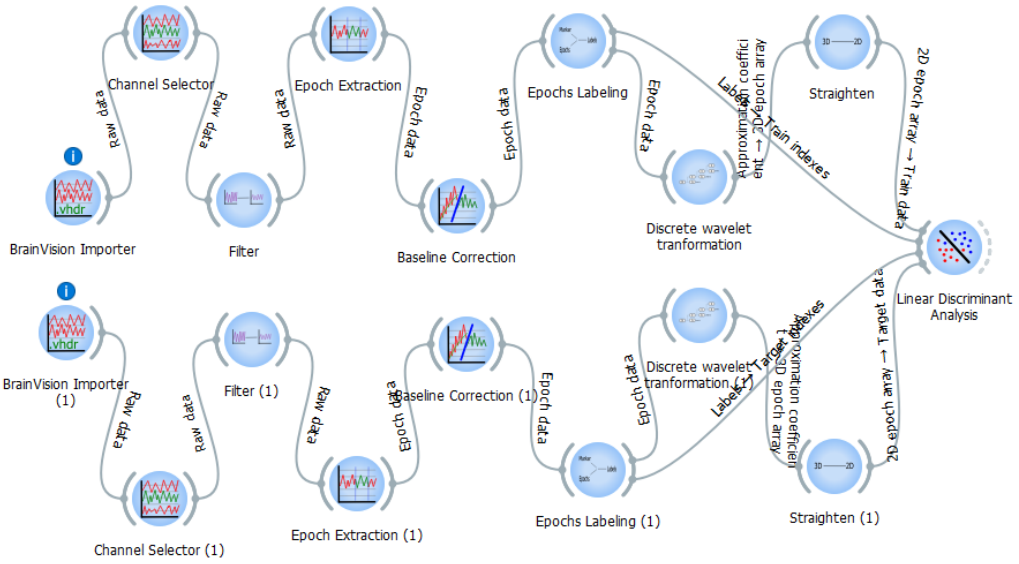


Obrázek 7.19: Vykreslení komponent widgetu Independent Component Analysis

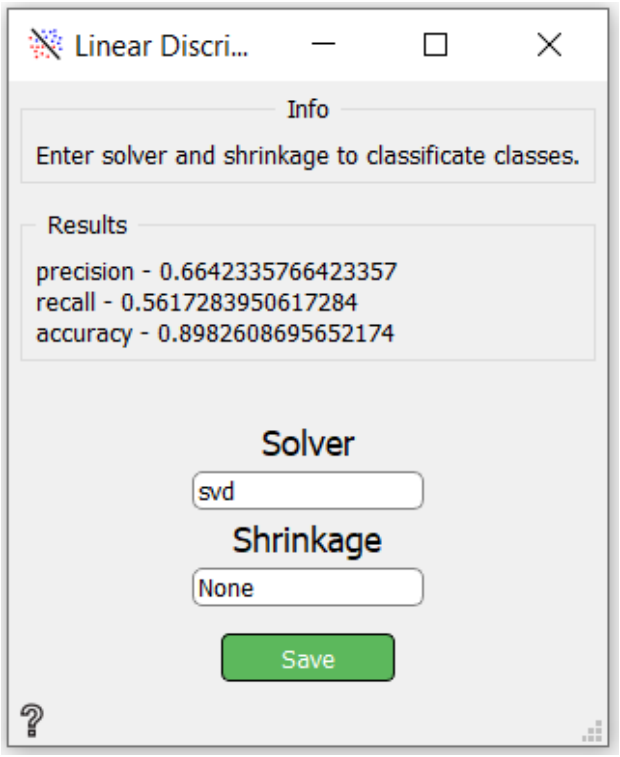
7.9 Testování widgetů Linear Discriminant Analysis, Epochs Labeling, Straighten a Discrete wavelet transformation

Při testování widgetů **Linear Discriminant Analysis**, **Epochs Labeling**, **Straighten** a **Discrete wavelet tranformation** jsem použil více widgetů: `BrainVision Importer`, `Channel Selector` s kanálem *Pz*, `Filter` s filtrací *0.1-12*, `Epoch Extraction` s parametry *PreStimulus* a *PostStimulus* nastavené na *-0.1* a *1* a `Baseline Correction` se *Start* a *End time 0 a 1*. **Epochs Labeling** má dvě třídy. V první je marker *S 2* a v druhé *S 4*. **Discrete wavelet tranformation** má hodnotu *Wavelet family* nastavenou na *db1* a *Number of degrees of transformation* na *1*. **Linear Discriminant Analysis** má *Solver* s hodnotou *svd* a *Shrinkage* s hodnotou *None*. Všechny widgety kromě **Linear Discriminant Analysis** jsou ve workflow dvakrát kvůli vstupu trénovacích a testovacích dat do **Linear Discriminant Analysis**. Vstupní eeg soubor widgetu `BrainVision Importer`, z kterého jsou trénovací data, je *LED_28_06_2012_104.vhdr* a soubor pro testovací data

je *LED_22_05_2012_86.vhdr*. Obrázek 7.20 workflow je na straně 63. Výsledky klasifikátoru jsou vidět na obrázku 7.21 na straně 64. Výsledky jsou u *precision, recall* i *accuracy* vyšší než 50%, test je tedy považován za úspěšný.



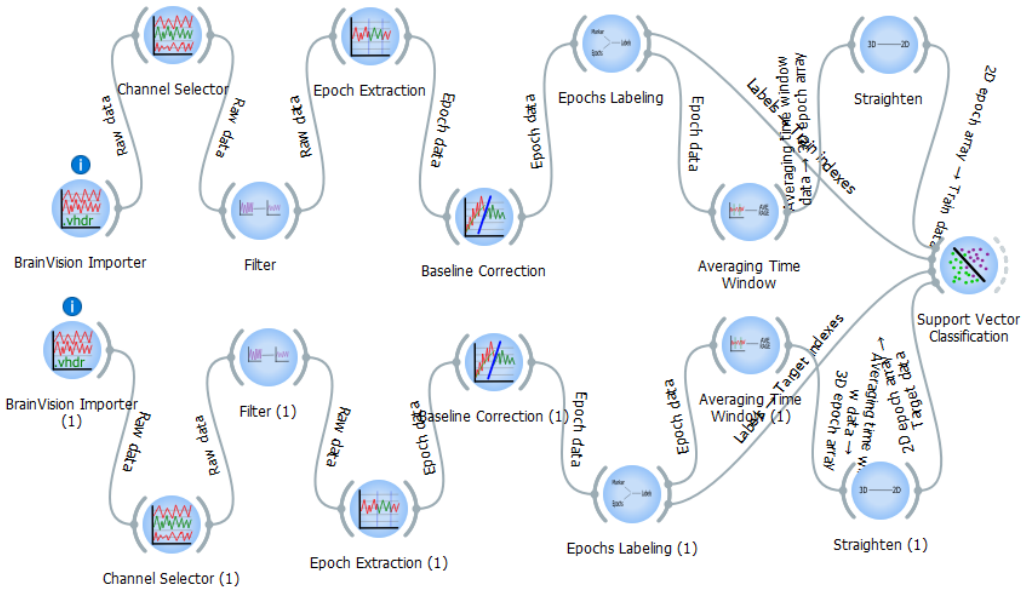
Obrázek 7.20: Workflow testování widgetů Linear Discriminant Analysis, Epochs Labeling, Straighten a Discrete wavelet transformation



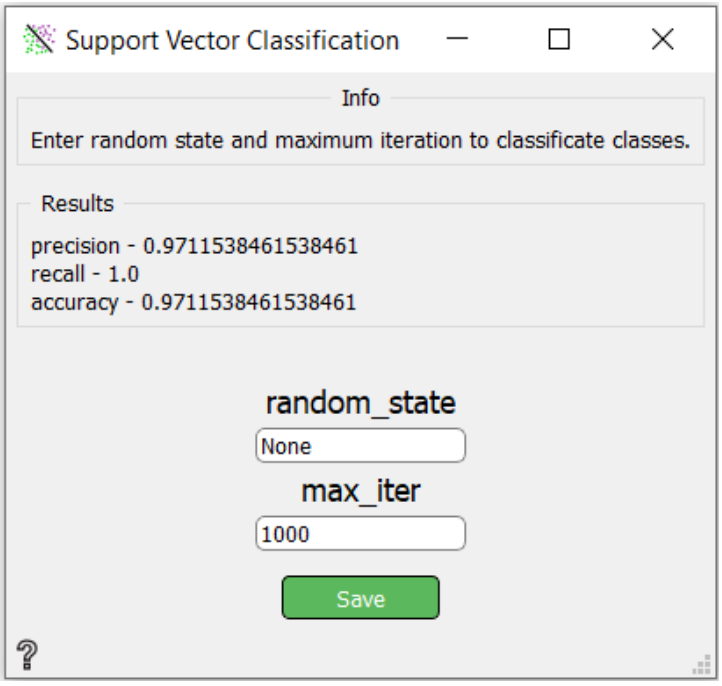
Obrázek 7.21: Výstup widgetu Linear Discriminant Analysis

7.10 Testování widgetů Support Vector Classification a Averaging Time Windows

Toto testování je podobné jako testování předchozí. Zaměněny jsou jen widgety **Averaging Time Windows** za `Discrete wavelet tranformation` a **Support Vector Classification** za `Linear Discriminant Analysis`. **Averaging Time Windows** má časová okna nastavená na *0.1,0.5* a *0.5,0.8*. U **Support Vector Classification** má *random state* hodnotu *None* a *max iter* má hodnotu *1000*. Obrázek 7.22 workflow je na straně 65. Výsledky klasifikátoru jsou vidět na obrázku 7.23 na straně 65. Tyto výsledky sice také přesahují 50%, ale hodnoty jsou až podezřele vysoké a *recall* se 100% je velice nepravděpodobný. Proto jsem se rozhodl otestovat vzájemně oba klasifikátory na stejných datech a klasifikátory tak porovnat v následujícím testování.



Obrázek 7.22: Workflow testování widgetů Support Vector Classification a Averaging Time Windows



Obrázek 7.23: Výstup widgetu Support Vector Classification

Porovnání klasifikátorů									
Trénovací data*	Testovací data*	LDA				SVC			
		TP	TN	FP	FN	TP	TN	FP	FN
104	86	868	1165	455	752	1495	304	1316	125
86	87	1136	1038	802	704	1467	749	1091	373
86	104	1034	1088	732	786	1483	979	841	337

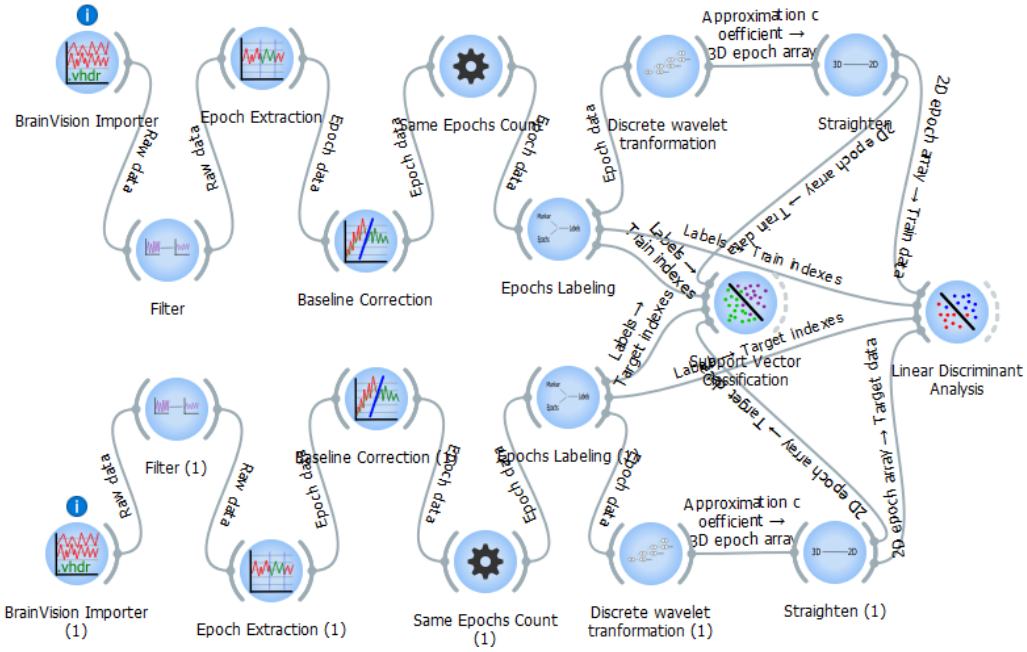
*Zkrácený název souboru s daty

Tabulka 7.1: Tabulka výsledků

7.11 Porovnání klasifikátorů

V tomto testu jde o porovnání klasifikátorů **Linear Discriminant Analysis (LDA)** a **Support Vector Classification (SVC)**. V testovacím workflow jsou také widgety `BrainVision Importer`, `Fiter`, `Epoch Extraction`, `Baseline Correction`, `Epochs Labeling`, `Discrete wavelet transformation`, `Straighten` a novým widgetem je zde widget `Same Epochs Count`, který jsem vytvořil čistě pro toto testování. Tento widget sjednocuje počet událostí obou tříd. Jeho parametry jsou *Event ID with smaller number* nastavený na *2* a *Event ID with larger number* nastavený na *4*. Ostatní jsou stejné jako u testování widgetů **Linear Discriminant Analysis, Epochs Labeling, Straighten a Discrete wavelet transformation** výše. Jen widget `Channel Selector` chybí. Kanály tedy budou ve výpočtu všechny. Widgety `BrainVision Importer` mají eeg soubor nastavený podle tabulky 7.1 na straně 66. Ve stejné tabulce jsou i výsledky těchto testování. Výsledky jsou hodnoty *true positive (TP)*, *true negative (TN)*, *false positive (FP)*, *false negative (FN)*, kde *TP* je počet správně klasifikovaných událostí z první třídy, *FP* je počet špatně klasifikovaných z první třídy, *TN* je počet správně klasifikovaných událostí z druhé třídy a *FN* je počet špatně klasifikovaných z druhé třídy. Vzhledem k tomu, že díky widgetu `Same Epochs Count` je počet událostí v první i druhé třídě stejný (1620), byl by v ideálním případě stejný maximální počet v *TP* i *TN* (1620) a v *FP* i *FN* (0). Pro důkaz funkčního klasifikátoru by ale stačilo, pokud by čísla *TP* a *TN* byly výrazně vyšší než čísla *FP* a *FN*.

Z výsledné tabulky je zřejmé, že **LDA** ve všech případech má výsledky dobré a **SVC** má výsledky velice nepřesné. Výsledkem hodnocení tedy je, že klasifikátor **LDA** funguje, zatímco klasifikátor **SVC** se nedá považovat za funkční. Workflow testování je na obrázku 7.24 na straně 67.



Obrázek 7.24: Workflow porovnání klasifikátorů

Zbylé testování jsou popsané v příloze v sekci Testování.

8 Zhodnocení výsledků

Výběr nástroje pro tvorbu workflow *Orange* před nástroji *Dynamic Signal Processing Workflow Designer*, *Snakemake* a *NeuroPype* se zdá být úspěšný. V nástroji se pracovalo dobře a implementace widgetů byla poměrně snadná. Nástroje *Neuropype* a *Orange* měli podobně vysoké hodnocení a *Orange* jsem vybral místo nástroje *Neuropype* jen kvůli lepším licenčním podmínkám. Nicméně zpětně mohu říci, že i kdyby byly licenční podmínky stejné, vybral bych si nyní také *Orange*. A to díky tomu, že je uživatelsky přístupný a mohu se podívat na všechny již implementované widgety pro snadnější pochopení jejich logiky a pro inspiraci při vyvíjení nových.

Widgety jsou kvalitně implementované pro snadné použití a zapojení do workflow. Pokud by budoucímu uživateli v nějakém widgetu chyběl parametr implementované metody, je snadné widget upravit dle potřeby uživatele. V tom tkví další výhoda nástroje *Orange*.

Některé Widgety nejsou podle `Specifikace požadavků` implementovány. Malá část widgetů s nízkou prioritou byla po dohodě s vedoucím práce odstraněna z plánu implementace, protože vyžadovala přítomnost a testování v laboratoři.

Při testování jsem zjistil, že widget Support Vector Classification jako jediný nefunguje dle očekávání. Vzhledem k tomu, že výsledky již implementované metody tohoto klasifikátoru používám z rozšířené knihovny `sklearn` a výsledná data se významně liší podle datasetů vložených jako testovací a trénovací data, je možné, že chyba může spočívat v nevhodně zvoleném klasifikátoru pro data tohoto typu (eeg data). Testování ostatních widgetů proběhlo podle očekávaných výsledků.

Výsledek práce vypadá i graficky velice pěkně díky povedenému gui nástroje a doplněným ikonám každého widgetu.

9 Závěr

V této práci jsem se seznámil s projekty vytvářenými na Katedře informatiky a výpočetní techniky na Západočeské univerzitě v Plzni, které mi přišly velice zajímavé a hlavně inspirující pro mou bakalářskou práci. Po prostudování současných možností a dostupných nástrojů, jsem shledal jako užitečné aplikace nástroje Dynamic Signal Processing Workflow Designer, Snakemake, NeuroPype a Orange. Při výběru vhodného nástroje pro BCI workflow jsem narazil na několik problémů, zvláště u nástroje Snakemake. Potíže jsem vyřešil, ale snížilo to hodnocení nástroje. Díky vysokému hodnocení a dobrým licenčním podmínkám jsem nakonec vybral jako vhodný nástroj pro tvorbu workflow Orange. Metody ze stávajícího BCI experimentu jsem modifikoval jako widgety a implementoval tak do nástroje Orange. Výsledné řešení jsem následně zhodnotil v kapitole Zhodnocení výsledků.

Implementované metody spolu s jejich ikonami a dokumentací, jsem zabalil do add-onu Orange3-Eeg. Tento add-on je možný nalézt na mém githubu https://github.com/Iopi/orange3-eeg. Github s projektem obsahuje také návod k instalaci add-onu. Z githubu je možné projekt stáhnout, nainstalovat do nástroje Orange a libovolně ho užívat s podmínkami obecné veřejné licence GNU General Public License. Do add-onu je možné dále přidat další metody pro zpracování eeg dat a rozšířit tak add-on. Rozšíření může spočívat například ve widgetech pro metodu Společný prostorový vzor, neuronové sítě nebo pro nahrání dalších datových formátu pro eeg. Případně si uživatel může upravit stávající widgety dle své potřeby. Add-on Orange3-Eeg je určen pro kohokoliv, kdo si chce usnadnit práci při zpracování nebo vizualizaci eeg dat.

Seznam obrázků

Literatura

[1] Christian Kothe, D. M. *Eeg data generator* [online]. 2016. [cit. 2020/23/4]. Dostupné z: https://github.com/sccn/lsl_archived/blob/master/LSL/liblsl-Python/examples/SendDataAdvanced.py.

[2] Christian Kothe, D. M. *Marker generator* [online]. 2016. [cit. 2020/23/4]. Dostupné z: https://github.com/sccn/lsl_archived/blob/master/LSL/liblsl-Python/examples/SendStringMarkers.py.

[3] Demšar, J. et al. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*. 2013, 14, s. 2349–2353. Dostupné z: http://jmlr.org/papers/v14/demsar13a.html.

[4] Gramfort, A. et al. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*. 2013, 7, s. 267. ISSN 1662-453X. doi: 10.3389/fnins.2013.00267. Dostupné z: https://www.frontiersin.org/article/10.3389/fnins.2013.00267.

[5] Hazelcast. *Directed acyclic graph* [online]. hazelcast, 2018. [cit. 2018/15/12]. Dostupné z: https://hazelcast.com/glossary/directed-acyclic-graph/.

[6] Köster, J. – Rahmann, S. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*. 2012, 28, 19, s. 2520–2522. doi: 10.1093/bioinformatics/bts480. Dostupné z: http://dx.doi.org/10.1093/bioinformatics/bts480.

[7] Lee, G. et al. PyWavelets: A Python package for wavelet analysis. *Journal of Open Source Software*. 2019, 4, 36, s. 1237. doi: 10.21105/joss.01237. Dostupné z: https://doi.org/10.21105/joss.01237.

[8] Merriam-Webster. *Annotaion* [online]. Merriam-Webster.com, 2018. [cit. 2018/15/12]. Dostupné z: https://www.merriam-webster.com/dictionary/annotation.

[9] Pinto, J. *Google Summer of Code 2018* [online]. 2018. [cit. 2018/07/12]. Dostupné z: http://www.zedacross.com/gsoc2018/.

[10] Radek Müller, J. R. J. K. *EEG Workflow v prostředí Orange*. 2019.

[11] Tim Mullen, A. G. C. K. *NeuroPype Architecture* [online]. 2018. [cit. 2018/15/12]. Dostupné z: https://www.neuropype.io/docs/developer/neuropype_architecture.html.

[12] Tuller, D. *Dr. William Dobelle, Artificial Vision Pioneer, Dies at 62* [online]. 2004. [cit. 2020/8/4]. Dostupné z: https://www.nytimes.com/2004/11/01/obituaries/dr-william-dobelle-artificial-vision-pioneer-dies-at-62.html.

[13] ŽÁK, R. *Zpracování mozkové aktivity v bci systémech* [online]. 2012. [cit. 2018/11/12]. Dostupné z: http://trilobit.fai.utb.cz/zpracovani-mozkove-aktivity-v-bci-systemech_e498d494-fd80-4948-a8d6-f5f3720bba2d.

Seznam zkratek

- **BCI** - Brain-Computer Interface
- **EEG** - Elektroencefalografie
- **MEG** - Magnetoencefalografie
- **sEEG** - Stereoelektroencefalografie
- **ECoG** - Electrocorticography
- **NTIS** - Nové technologie pro informační společnost
- **DAG** - Directed acyclic graph
- **JSON** - JavaScript Object Notation
- **GUI** - Graphical User Interface
- **GLP** - General Public License
- **ICA** - Independent Component Analysis
- **LDA** - Linear Discriminant Analysis
- **SVC** - Support Vector Classification
- **PSD** - Power Spectral Density
- **LSL** - LabStreaming Layer

Přílohy

A Instalační příručka

A.1 Instalační komponenty pro projekt

Pro funkční instalaci tohoto projektu je potřeba mít nainstalováno:

- **Python** verzi 3.6 nebo vyšší
- **mne-Python** verzi 0.17.1 – knihovna pro Python
- **AnyQt** – knihovna pro Python
- **PyQt5** – knihovna pro Python
- **pylsl** – knihovna pro Python
- **pywt** – knihovna pro Python
- **Orange3** verzi 3.23.1 – nástroj pro tvorbu workflow

Instalace nástroje **Orange3** je detailně popsána pro operační systém `Windows` na stránce `https://orange.biolab.si/download/#windows`

A.2 Instalace Orange3-Eeg add-on

Balíčky kategorií pro nástroj Orange3 se nazývají `add-ony`. Mnou implementovaný `add-on` se jmenuje **Orange3-Eeg**.

Instalace

Po úspěšné instalaci Orange3 můžete nainstalovat doplňkový balíček *Orange3-Eeg*.

1. Stáhněte si tento projekt z githubu (`https://github.com/Iopi/orange3-eeg`) a rozbalte ho tam, kde chcete mít `add-on`.
2. Přejděte do hlavní složky:

   ```
   cd orange3-eeg
   ```

3. Pro instalaci `add-onu`

   - spusťe příkaz:

     ```
     pip install .
     ```

   - nebo pokud chcete kód ponechat ve vývojovém adresáři spusťe příkaz:

     ```
     pip install -e .
     ```

   to umožní, aby Orange rozpoznal balíček, a když dojde ke změnám zdrojového kódu, také je rozpozná.

4. Po instalaci by měl Orange sledovat balíček. Pro spuštění aplikace spusťte příkaz:

   ```
   python -m Orange.canvas
   ```

   kategorie Eeg by se měla zobrazit v levém menu aplikace Orange.

B Uživatelská příručka

Aplikaci s nainstalovaným `add-onem` **Orange3-Eeg** je možné spustit příkazem

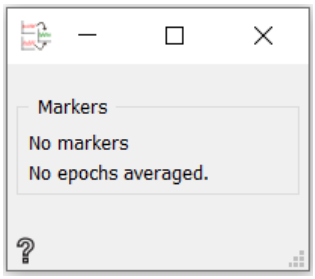
```
python -m Orange.canvas
```

Práce s nástrojem Orange a tvorba workflow zde je velice jednoduchá. Když otevřete Orange, první věc, co uvidíte, je uvítací obrazovka. Odtud můžete začít nové workflow, otevřít uložené nebo prozkoumat pomocné tutoriály. Jak bylo zmíněno v kapitole `Analýza a design vybraného nástroje`, v levé části od canvasu jsou kategorie a po kliknutí na kategorie se ukáží widgety. Výběr widgetu můžete zvolit buď tak, že na něj kliknete nebo metodou *drag and drop*, tedy přetáhnutím do canvasu. Otevření widgetu dosáhnete dvojitým kliknutím na widget, kde můžete například upravovat jeho parametry. Widgety spolu komunikují. Mají vstupní kanál, výstupní kanál nebo obojí. Spojení dosáhnete přetáhnutím čáry z výstupní strany jednoho widgetu do vstupní strany druhého. Spojováním widgetů dostanete workflow. Na začátku workflow bývá vstupní widget, který nemá na vstupu spojení s jiným widgetem, ale výběr souboru. V kategorii Eeg je takovým widgetem například *BrainVision Importer*. Výstupním widgetem bývají vizualizační widgety, například *EEG Plot*.

C Implementované widgety

C.1 Averaging

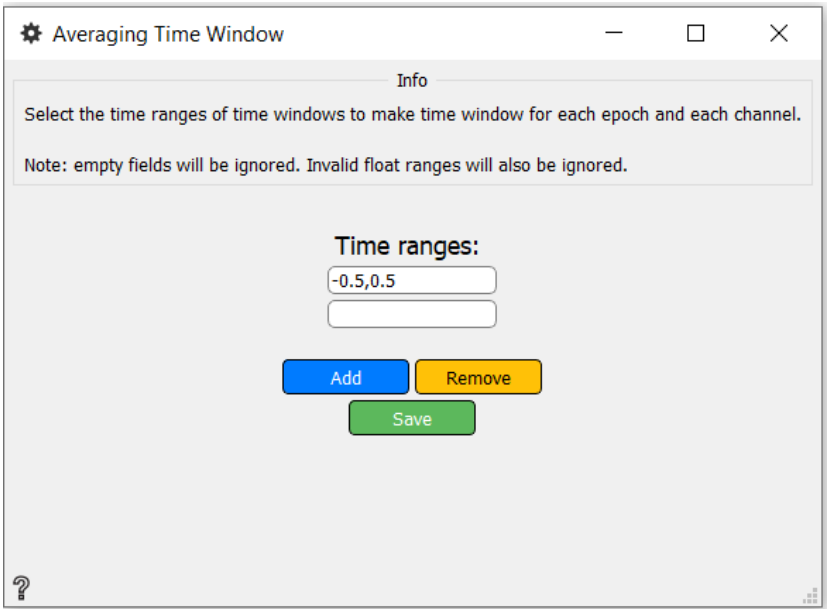
Widget **Averaging** průměruje data epoch na základě stimulačních markerů. Vstupní data jsou epochy předané strukturou *mne.Epochs* a jeden nebo více markerů z widgetu *EEG Marker* a výstupní data jsou evokovaná data – *mne.Evoked*. Otevřený widget je zobrazen na obrázku 1 na straně 79. Pokud jsou na vstupu markery, zobrazí se jejich jméno a po průměrování se zobrazí průměrovaný počet epoch.



Obrázek 1: Widget Averaging

C.2 Averaging Time Window

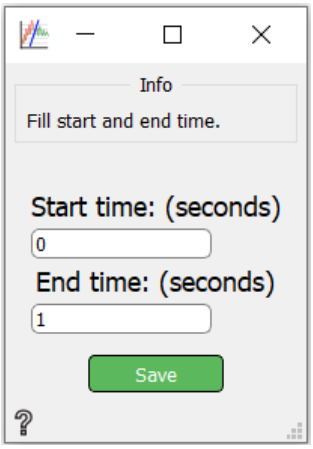
Widget **Averaging Time Window** vypočítá aritmetický průměr v definovaných časových oblastech pro každou epochu a každý kanál. Vstupem jsou *Epochy* a výstupem je výsledek tří-dimenzionálního pole ve formátu [epocha:kanál:průměr rozsahu]. Otevřený widget je zobrazen na obrázku 1 na straně 79. Parametrem jsou zde časové rozsahy (před a po události v epoše), které je možné přidávat tlačítkem *Add* a mazat tlačítkem *Remove*. Základní parametr widgetu je jeden časový rozsah od *-0.5* až *0.5* sekund. Uložit změny je možné tlačítkem *Save*.



Obrázek 2: Widget Averaging Time Window

C.3 Baseline Correction

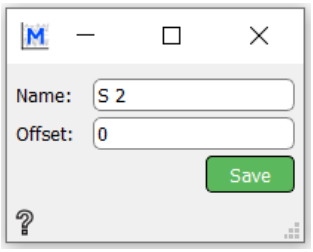
Widget **Baseline Correction** aplikuje základní korekci na data eeg. Vstupem jsou *Epochy* a výstupem stejné *Epochy* po korekci. Otevřený widget je zobrazen na obrázku 3 na straně 80. Zde jsou parametry dva – začátek a konec korekce v sekundách. Uložit změny je možné opět tlačítkem *Save*. Základní hodnota začátku korekce je *0* a konce korekce *1* sekunda. Korekce se provádí výpočtem průměrné základní periody a odečtením z dat.



Obrázek 3: Widget Baseline Correction

C.4 EEG Marker

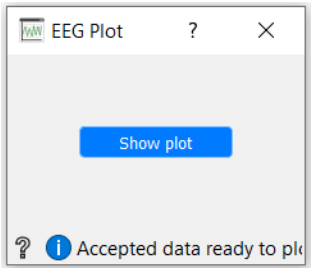
Widget reprezentující marker se nazývá **EEG Marker**. Jde o objekt obsahující název a ofset markeru. Název i ofset jsou parametry tohoto widgetu. Název navíc musí obsahovat identifikační číslo markeru oddělené mezerou od zbytku názvu. Základní název markeru je *S 2* a ofset je nevyplněný. Tlačítkem *Save* se uloží změněné hodnoty. Otevřený widget je zobrazen na obrázku 4 na straně 81. Výstupem je celý objekt markeru.



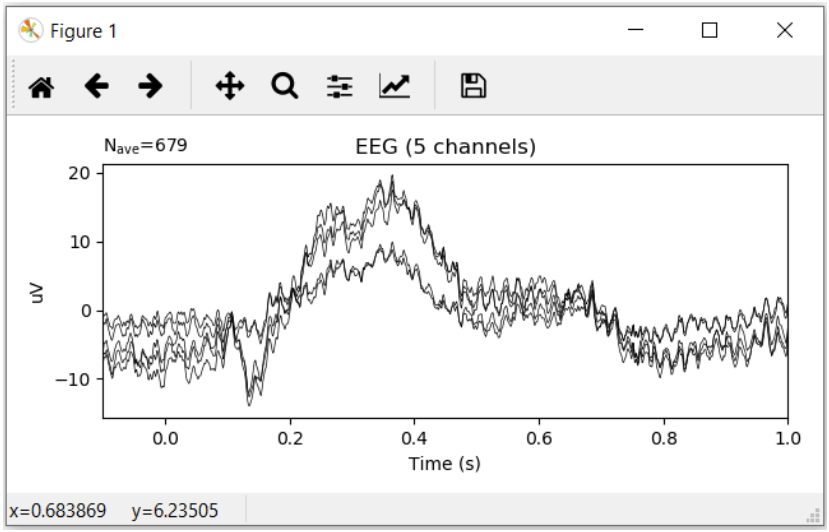
Obrázek 4: Widget EEG Marker

C.5 EEG Plot

Jeden z vizualizačních widgetů, je v kategorii *EEG* **EEG Plot**. Jedná se o výstupní widget, který vykresluje epochy, surová i evokovaná data. Vstupem tedy mohou být struktury *mne.io.Raw*, *mne.Epochs*, *mne.Evoked* a *RawBrainVision* data a výstupem graf vstupních dat. Widget nemá žádné parametry. V uživatelském grafickém rozhraní je jen tlačítko *Show plot* k zobrazení grafu. Otevřený widget je zobrazen na obrázku 5 na straně 81. Příklad vizualizace je na obrázku 6 na straně 82.



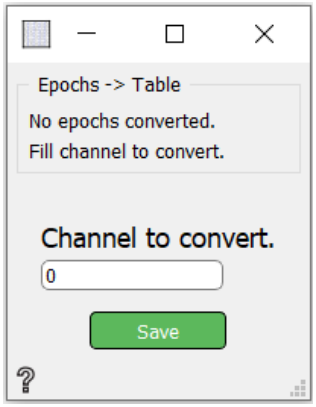
Obrázek 5: Widget EEG Plot



Obrázek 6: Příklad vizualizace widgetu EEG Plot

C.6 Epochs To Table

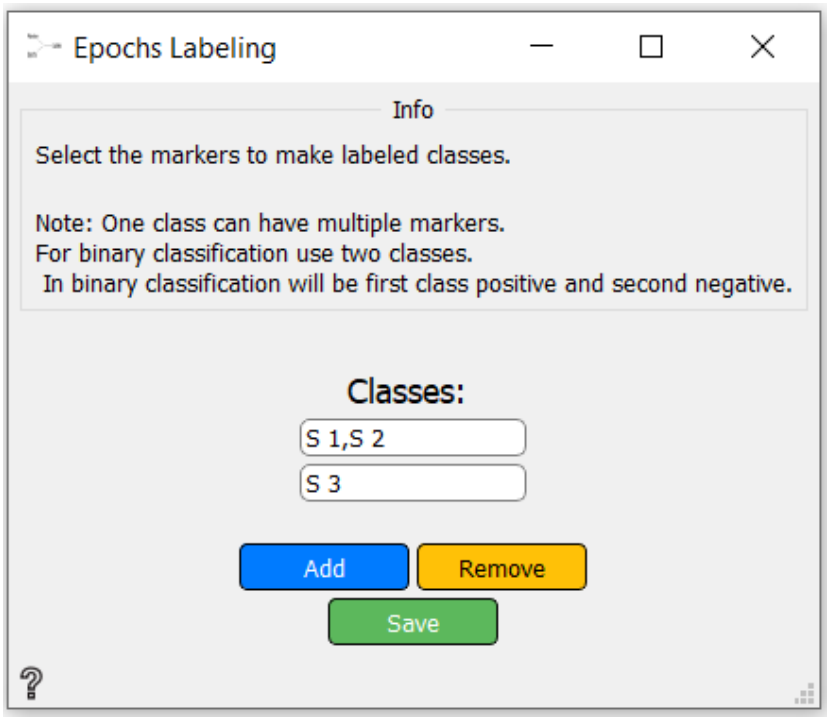
Speciálním widgetem je widget **Epochs To Table**, který jako jediný může přímo komunikovat s widgety mimo kategorii *Eeg*. Převádí data struktury *mne.Epochs* na strukturu *Table*, která je v nástroji *Orange* nejvíce rozšířená. Vstupem je tedy struktura *mne.Epochs* a výstupem struktura *Orange.data.Table*. Otevřený widget je zobrazen na obrázku 7 na straně 82. Widget má jeden parametr. Tento parametr se nazývá *Channel to convert* a určuje kanál z dat, který bude převeden na *Table* data. Jeho základní hodnota je *0*, tedy první kanál struktury *mne.Epochs*. Tlačítkem *Save* se uloží změněné hodnoty. Po procesu převodu se zobrazí *Epochs converted*, aby bylo zřejmé, že konverze se provedla úspěšně.



Obrázek 7: Widget Epochs To Table

C.7 Epochs Labeling

Widget **Epochs Labeling** označuje data epoch podle jejich událostí. Tato data jsou definována podle markerů. Na vstupu tohoto widgetu jsou *Epochy*. Výstupem jsou zkrácené *Epochy*, vybrané podle identifikačních čísel markerů a druhým výstupem je jejich označení podle tříd. Tyto třídy je možné definovat pomocí parametrů. V každé třídě může být více markerů oddělených čárkou. Třídu je možné přidat tlačítkem *Add*, odstranit tlačítkem *Remove* a uložit změny pak tlačítkem *Save*. Otevřený widget je zobrazen na obrázku 8 na straně 83.



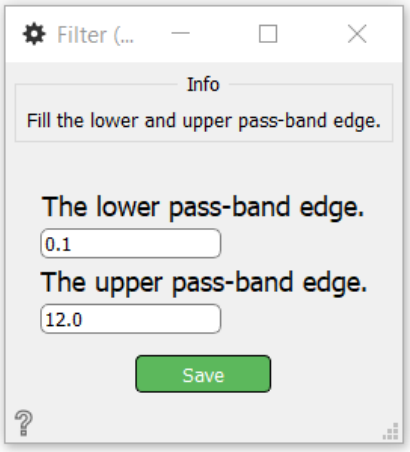
Obrázek 8: Widget Epochs Labeling

https://www.overleaf.com/project/5c3de34ce5e907418b696a14

C.8 Filter

Pro filtrování dat v kategorii *Eeg* slouží widget **Filter**. Tento widget může filtrovat všechny tři struktury – *mne.io.Raw, mne.Epochs i mne.Evoked*. Vstupem tedy mohou být *Raw* data, *Epochy* nebo *Evokovaná* data. Stejný typ dat pak bude na výstupu. Filtrování závisí na dvou parametrech. Jedním z nich je *The lower pass-band edge* a jedná se o dolní mezní frekvenci. Pokud žádná dolní mezní frekvence není zadána a horní ano, jde o dolní propust

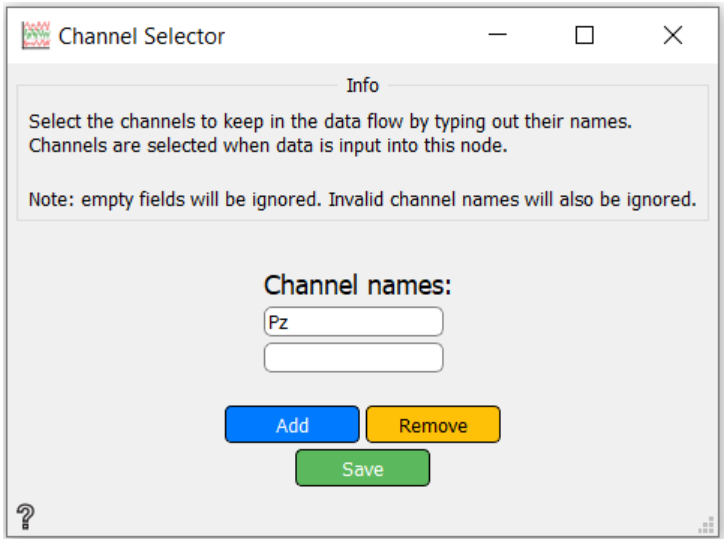
dat. Druhým parametrem *The upper pass-band edge*, kde jde naopak o horní mezní frekvenci. Pokud ta není zadána, pak se jedná o horní propust dat. Parametr *The lower pass-band edge* má základní hodnotu nastavenou na *0.1* a parametr *The upper pass-band edge* na *12*. Uložit změny je možné tlačítkem *Save*. Otevřený widget je zobrazen na obrázku 9 na straně 84.



Obrázek 9: Widget Filter

C.9 Channel Selector

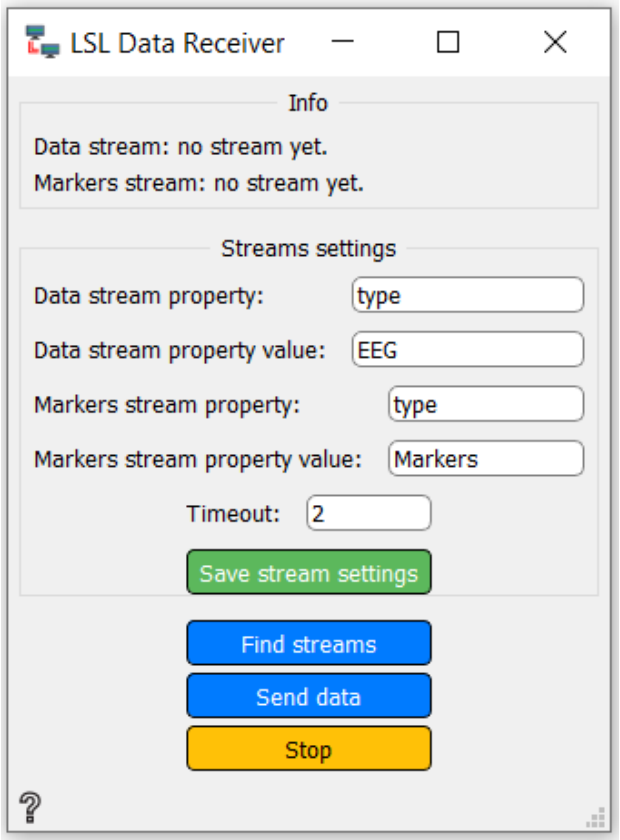
Důležitým widgetem pro zpracování dat je widget **Channel Selector**. **Channel Selector** vybere zvolené kanály jako parametr widgetu a pošle tato data dál. Vstupem je struktura *mne.io.Raw* a výstupem stejná struktura jen s vybranými kanály. Parametrem jsou tedy kanály, které jsou ponechány v datech. Kanál *Pz* je zde jako základní přednastavená hodnota, kterou je možné změnit. Tlačítkem *Add* je možné přidat další kanál a tlačítkem *Remove* je možné je mazat. Uložit změny je možné tlačítkem *Save*. Otevřený widget je zobrazen na obrázku 10 na straně 85.



Obrázek 10: Widget Channel Selector

C.10 Lsl Data Receiver

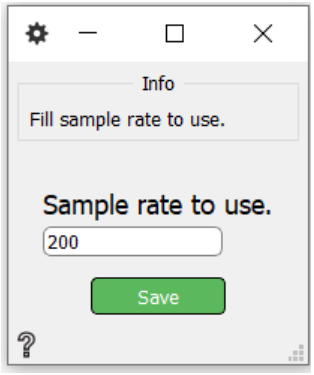
Druhým vstupním widgetem v kategorii *Eeg* je widget **Lsl Data Receiver**. Tento widget sbírá data z online streamů a vytvoří z nich strukturu *mne.io.Raw*. Vstupem jsou tedy online streamy eeg dat a markerů a výstupem struktura *mne.io.Raw*. Widget má celkem pět parametrů a čtyři tlačítka v grafickém uživatelském rozhraní. Prvním parametrem je vlastnost datového toku *Data stream property* se základní hodnotou *type*. Druhým parametrem je *Data stream property value*. *Data stream property value* je hodnota vlastnosti datového toku a její základní hodnota je *EEG*. Stejné parametry jako pro data, jsou i pro markery, tedy *Markers stream property* se základní hodnotou *type* a *Markers stream poperty value* se základní hodnotou *Markers*. Pátým parametrem je *Timeout* s hodnotou nastavenou na *2*. *Timeout* je volitelný časový limit operace hledání eeg dat a markerů v sekundách. Tlačítko *Save stream setting* uloží změny nastavení v parametrech widgetu. Tlačítkem *Find streams* začne hledání streamů dat a markerů a tlačítkem *Send data* se pošlou výsledná data na výstup. Tlačítkem *Stop* se zastaví načítaní dat ze streamů. Otevřený widget je zobrazen na obrázku 11 na straně 86.



Obrázek 11: Widget Lsl Data Receiver

C.11 Resample

Widget **Resample** převzorkuje data. Účelem této funkce je především urychlit výpočty. Na vstupu widgetu **Resample** mohou být struktury *mne.io.Raw, mne.Epochs a mne.Evoked* a na výstupu stejná, převzorkovaná data. Jediným parametrem je zde *Sample rate to use*, který nastavuje vzorkovací frekvenci. Jeho základní hodnotou je *200*. Uložit změnu této hodnoty je možno tlačítkem *Save*. Otevřený widget je zobrazen na obrázku 12 na straně 87.



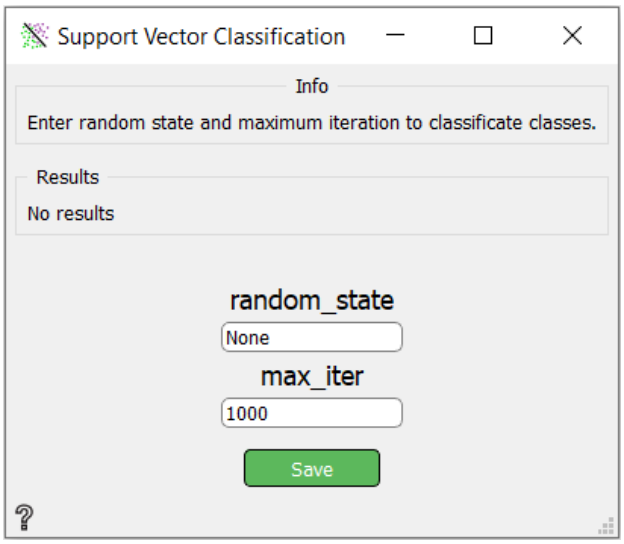
Obrázek 12: Widget Resample

C.12 Straighten

Straighten je pomocný widget pro klasifikaci dat. Klasifikátory, jako `Linear Discriminant Analysis` a `Support Vector Classification` mají na vstupu kromě příznaků také dvou-dimenzionální trénovací nebo testovací data. *Epochy* jsou data tří-dimenzionální (*[epocha:kanál:čas]*). Je potřeba tedy data přesunout do dvou dimenzí. Vstupem je tedy tří-dimenzionální pole a výstupem pouze dvou-dimenzionální pole ve tvaru *[epocha*kanál:čas]*. Widget nemá žádný parametr.

C.13 Support Vector Classification

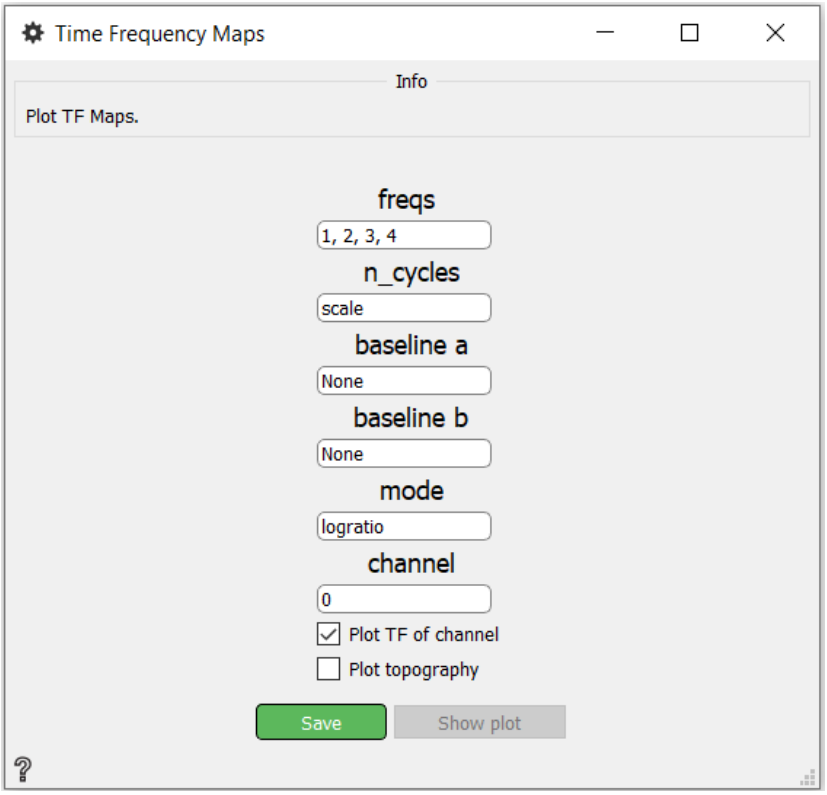
Druhým klasifikátorem v kategorii *Eeg* je widget **Support Vector Classification**. Pro tento widget jsem použil metodu `svm.LinearSVC` z knihovny `sklearn`. Vstupy do tohoto widgetu jsou opět čtyři. Jsou to trénovací data, jejich příznaky, testovací data a příznaky testovacích dat. Výstupem je *Accuracy*, tedy přesnost klasifikace všech tříd podle příznaků. Pokud jde o binární klasifikaci, je výstupem i *Recall* a *Precision*. Tyto výstupní data se zobrazí i v otevřeném widgetu v kolonce *Results*. U tohoto widgetu jsou parametry *random state* a *max iter*. *Random state* je "náhodné semínko"generátoru pseudonáhodných čísel, které se použije při zamíchání dat pro pokles duálních souřadnic. Jeho hodnota je *None*. *Max iter* je maximální počet iterací, ve kterých má být klasifikátor spuštěn se základní hodnotou *1000*. Otevřený widget je zobrazen na obrázku 13 na straně 88.



Obrázek 13: Widget Support Vector Classification

C.14 Time-Frequency Maps

Posledním widgetem je **Time-Frequency Maps**. **Time-Frequency Maps** je výstupním grafem a jeho výstupem je časově-frekvenční mapa. Tento widget má na vstupu *mne.Epochs* nebo *mne.Evoked*. Parametry této metody jsou *freq, n_cycles, baseline a, baseline b, mode a channel*. *Freq* nastavuje frekvenci a má základní ilustrační hodnotu *1, 2, 3, 4*, *n_cycles* je globálně počet cyklů nebo počet cyklů pro každou frekvenci. Pokud parametr *n_cycles* je *scale*, tak *n_cycle = freqs / 2*. *Baseline a* a *b* jsou začátek a konec časového intervalu k použití základní korekce. Oba parametry jsou nastavené na *None*. Parametr *mode* je režim pro provedení základní korekce a jeho hodnota je *logratio* (mód vydělení střední hodnoty). A parametr *channel* je kanál, pro který je časová frekvence zobrazena. Uložit jejich změny je možné pomocí tlačítka *Save*. K zobrazení si lze díky zaškrtávacím políčkům vybrat, zda zobrazit časovou frekvenci kanálu, celou topografii nebo případně obojí. Nakonec je zde tlačítko *Show plot* pro zobrazení map. Otevřený widget je zobrazen na obrázku 14 na straně 89.

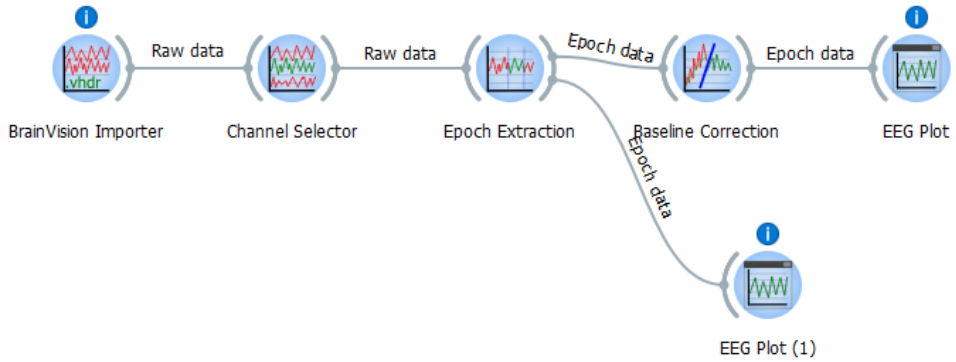


Obrázek 14: Widget Time-Frequency Maps

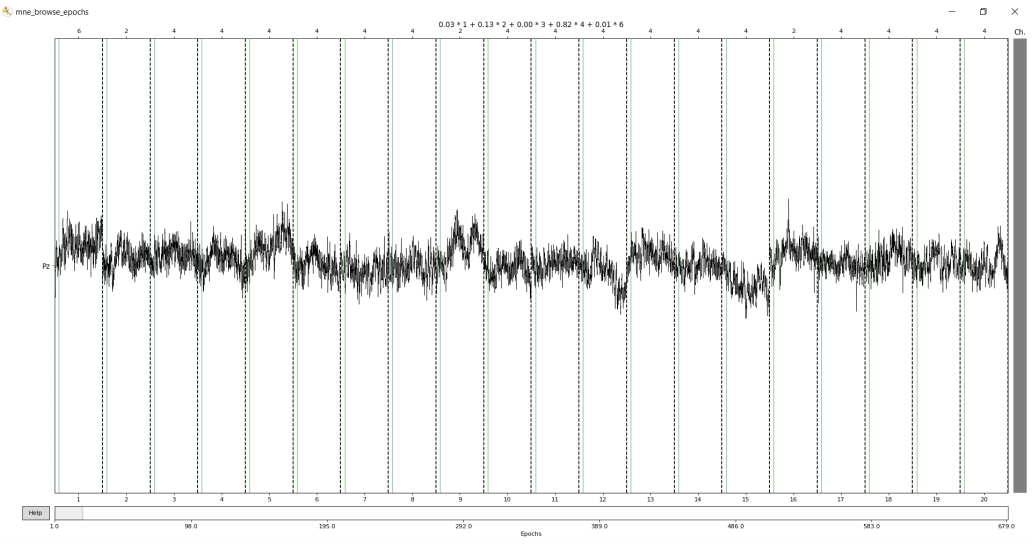
D Testování

D.1 Testování widgetu Baseline Correction

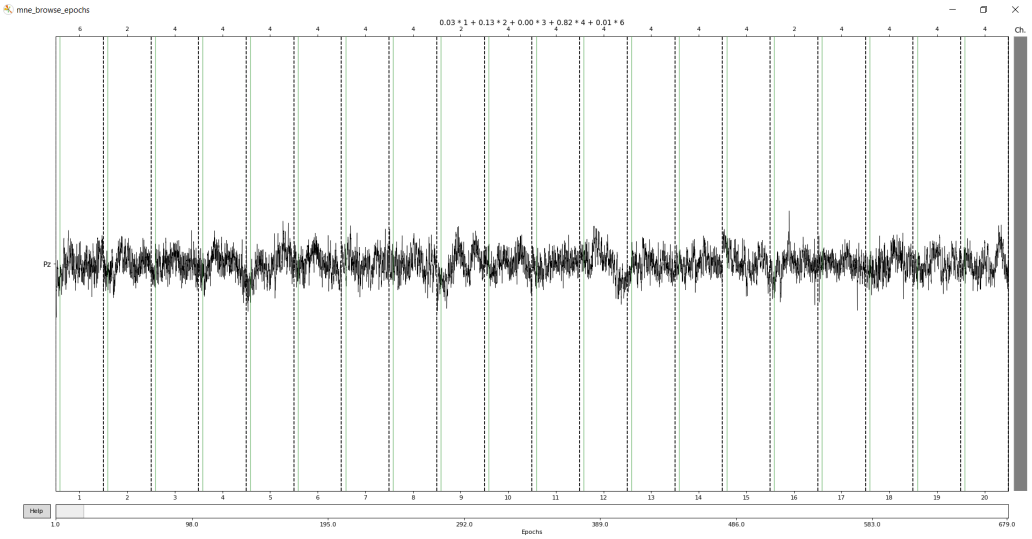
Při testování widgetu **Baseline Correction** byly použity widgety `BrainVision Importer`, `Channel Selector` s jedním nastaveným kanálem *Pz*, `Epoch Extraction` s parametry *PreStimulus* s hodnotou *-0.1* a *PostStimulus* s hodnoutou *1* a `EEG Plot`. Widget **Baseline Correction** má parametry *Start time 0 a End time 1*. Obrázek 15 workflow je na straně 90. Po porovnání grafu 16 na straně 90 před korekcí a grafu 17 na straně 91, je vidět, že graf po korekci je signál více redukovaný vzhledem k parametrům korekce.



Obrázek 15: Workflow testování widgetu Baseline Correction



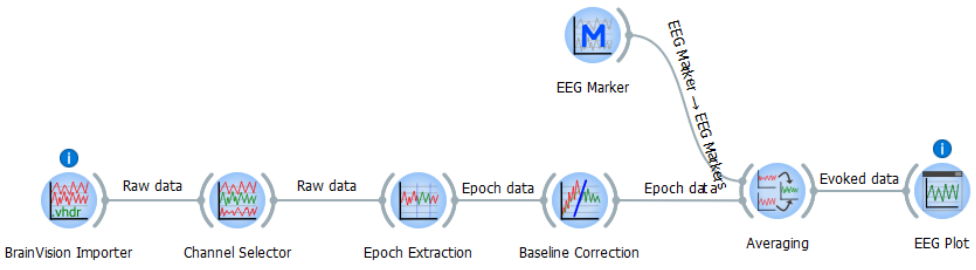
Obrázek 16: Výsledek grafu widgetu Baseline Correction před korekcí



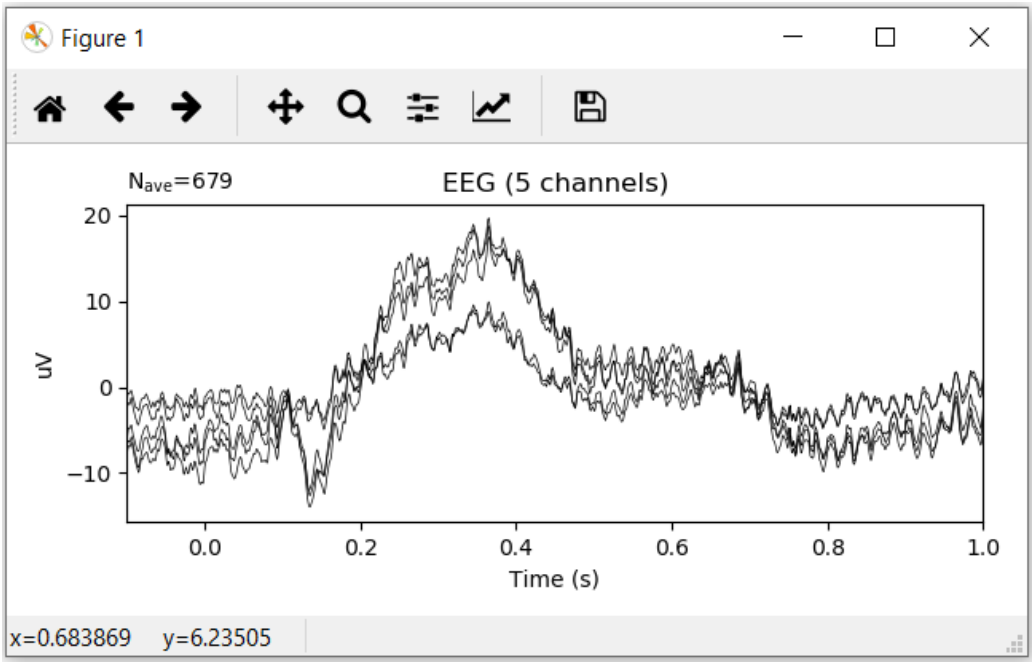
Obrázek 17: Výsledek grafu widgetu Baseline Correction po korekci

D.2 Testování widgetů Averaging a EEG Marker

V dalším testování jsem otestoval widget **Averaging** spolu s **EEG Markerem** za použití widgetů `BrainVision Importer`, `Channel Selector`, `Epoch Extraction`, `Baseline Correction` a `EEG Plot`. `Channel Selector` obsahuje pět kanálů a těmi jsou *Fp1, Fp2, C3, C4 a Pz*, `Epoch Extraction` má *PreStimulus* nastaven na *-0.1* a *PostStimulus* na *1*, `Baseline Correction` má parametry *Start time 0* a *End time 1* a **EEG Marker** má marker *S 2* a *offset 0*. Obrázek 18 workflow je na straně 91. Výsledkem, jak je vidět na obrázku 19 na straně 92, jsou evokovaná data s pěti kanály.



Obrázek 18: Workflow testování widgetů Averaging a EEG Marker

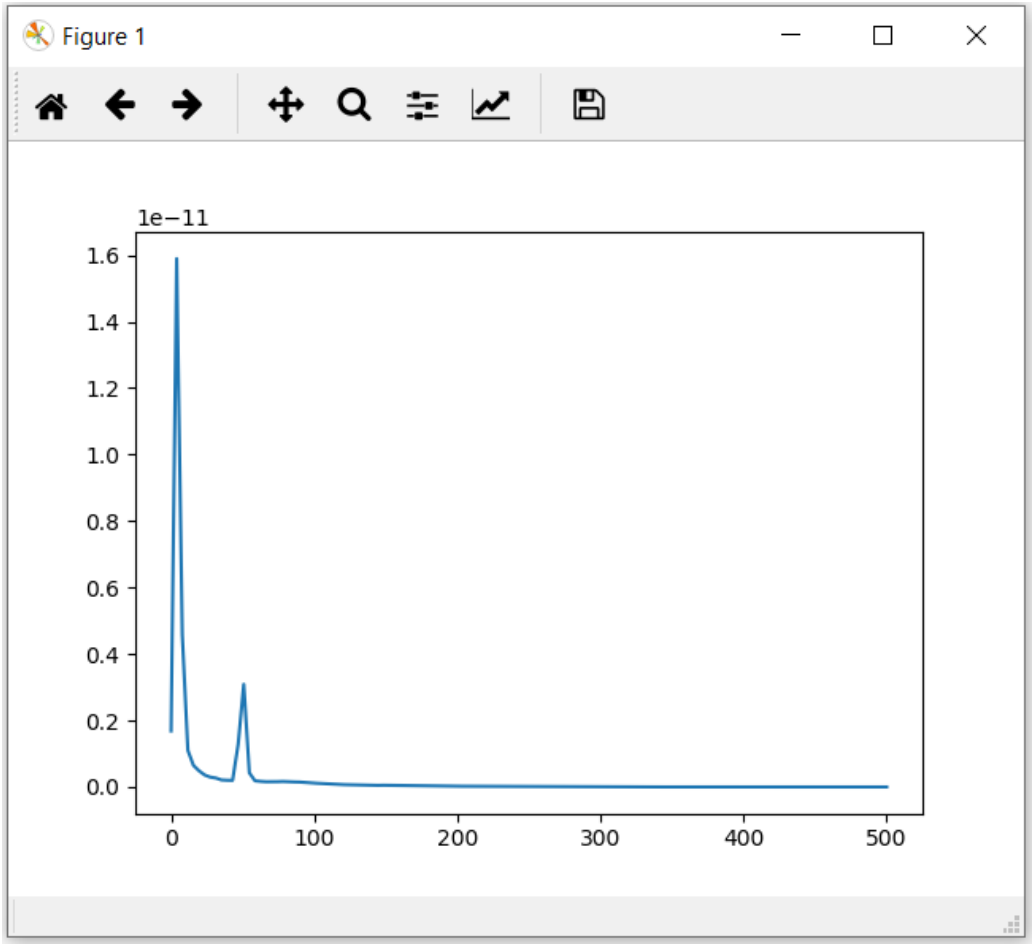


Obrázek 19: Výstup testování widgetů Averaging a EEG Marker

D.3 Testování widgetů Welch PSD a Plot Welch PSD

Tento test testuje widgety **Welch PSD** a **Plot Welch PSD** jen za pomocí widgetu `BrainVision Importer`. **Welch PSD** má parametr *Start frequency* nastavený na *-1000* Hz, *End frequency* na *1000* Hz, *Start time* na *-1000* sekund, *End time* na *1000* sekund a *Number of points* na *2500*. Obrázek 20 workflow je na straně 92. Výsledkem je graf výkonné spektrální hustoty na obrázku 21 na straně 93. Test je v pořádku.

Obrázek 20: Workflow testování widgetů Welch PSD a Plot Welch PSD



Obrázek 21: Výstup testování widgetů Welch PSD a Plot Welch PSD

D.4 Testování widgetu LSL Data Receiver

Jiná testovací data jsou použita u testování widgetu **LSL Data Receiver**. **LSL Data Receiver** je widget vstupní. Protože sbírá data z online streamů, nechal jsem pro testování data náhodně generovat. Náhodný generátor pro eeg data vypadá takto: [1]

```
info = StreamInfo('BioSemi', 'EEG', 8, 100, 'float32', 'myuid2424')
info.desc().append_child_value("manufacturer", "BioSemi")
channels = info.desc().append_child("channels")
for c in ["C3", "C4", "Cz", "Pz", "POz", "CPz", "O1", "O2"]:
    channels.append_child("channel") \
        .append_child_value("label", c) \
        .append_child_value("unit", "microvolts") \
```

```
        .append_child_value("type", "EEG")
outlet = StreamOutlet(info, 32, 360)

print("now sending data...")
while True:
    mysample = [rand(), rand(), rand(), rand(), rand(), rand(),
        rand(), rand()]
    stamp = local_clock()-0.125
    outlet.push_sample(mysample, stamp)
    time.sleep(0.01)
```

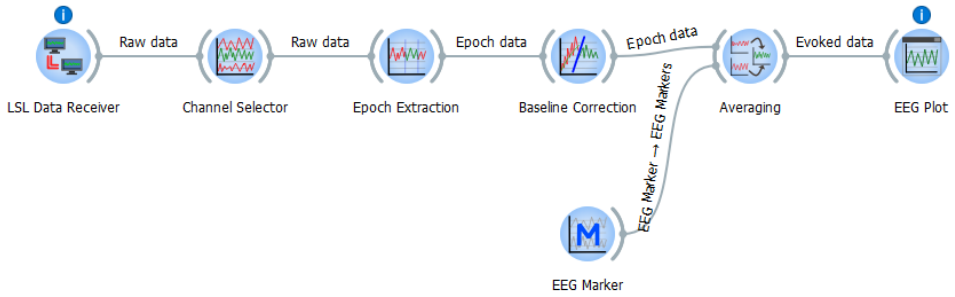
Parametry `StreamInfo` jsou – název streamu *BioSemi*, typ streamu *EEG*, počet kanálů – *8*, nominální síto – *100*, formát kanálů *'float32'* a identifikační číslo zdroje *'myuid2424'*. K těmto informacím se doplní kanály s náhodnými hodnotami a takový výsledek se odešle několikrát za sekundu.

A generátor pro markery vypadá takto: [2]

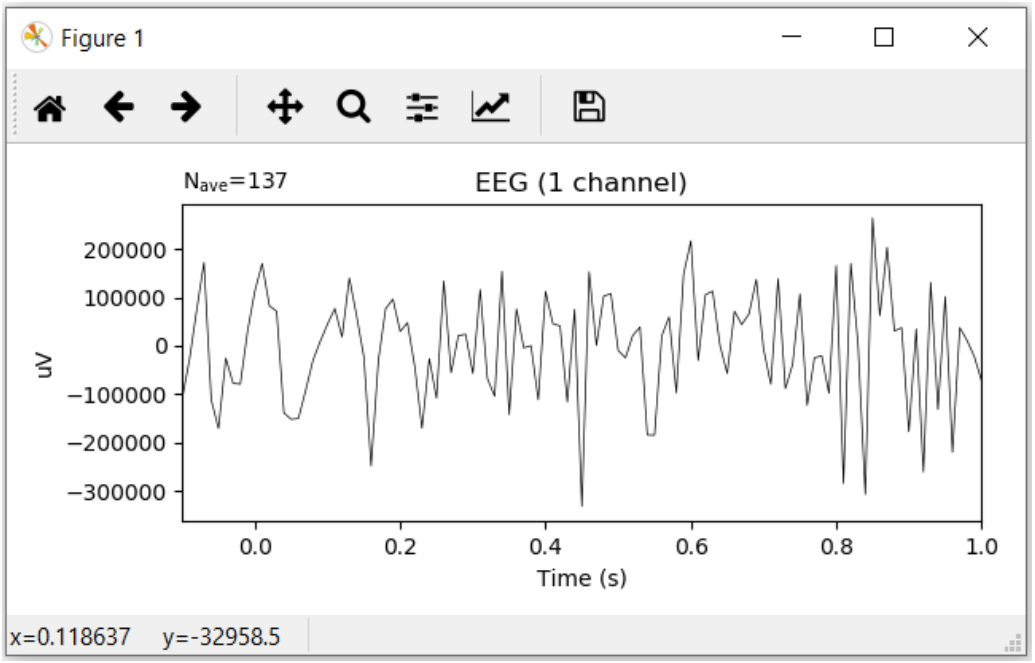
```
info = StreamInfo('MyMarkerStream', 'Markers', 1, 0, 'string',
    'myuidw43536')
outlet = StreamOutlet(info)

print("now sending markers...")
markernames = ['Test 1', 'S 2', 'Marker 3', 'XXX 4', 'Testtest 5',
    'Test-1-2-3 6']
while True:
    outlet.push_sample([random.choice(markernames)])
    time.sleep(random.random()*3)
```

U generátoru markerů má `StreamInfo` název streamu *MyMarkerStream*, typ streamu *Marker* a jen jeden kanál. Poté se, několikrát po náhodné době, náhodně posílají markery z listu `markernames` .

Ve workflow tohoto testování jsou kromě widgetu **LSL Data Receiver** také widgety `Channel Selector` se jedním kanálem *Pz*, `Epoch Extraction` s parametrem *PreStimulus* nastaveným na *-0.1* a parametrem *PostStimulus* nastaveným na *1*, `Baseline Correction` se *Start* a *End Time* nastavené na *0 a 1*, `EEG Marker` s markerem *S 2*, `Averaging` a `EEG Plot`. Obrázek 22 workflow je na straně 95. Výsledný graf je na obrázku 23 na straně 95.



Obrázek 22: Workflow testování widgetu LSL Data Receiver



Obrázek 23: Výstup testování widgetu LSL Data Receiver