EL Missaoui Imane
First-year Master's in Healthcare Data Science

Internship report

# Uncovering The Topological Features For The Characterisation Of A New Biomarker In Parkinson's Disease

September 2, 2023

Jury members

| | |
|---|---|
| Pr Benjamin Ginouhya | President |
| Dr Amaël Broustet | Supervisor |
| Dr Anne-Sophie Rolland | Supervisor |
| Quentin Leroy | Research Engineer |

Faculty of Health Engineering and Management - ILIS 42 rue Ambroise Paré 59120 LOOS

# Acknowledgements

I would like to take this opportunity to express my sincere gratitude to all individuals who have contributed to the successful completion of this internship project on "Uncovering the Topological Features for the Characterization of a new biomarker in Parkinson's Disease." Let's start with my supervisors, Anne-Sophie Rolland and Amaël Broustet, who where sufficiently confident in my capabilities and gave me the opportunity to work on a technical research project. I am grateful for the effort and time spent on this project, the insightful feedback and the comfortable and motivating environment they have cultivated, which has enabled me to work well. I would like to extend my deepest appreciation to Lille Neuroscience and Cognition Center for granting me the opportunity to undertake my internship project. I would also like to extend my thanks to my cat, Mily, for supporting me during the late-night research sessions. Mily's presence and constant companionship have offered me comfort, and a sense of calm, making the journey more enjoyable and less daunting. I would also like to express my gratitude to my family and my friend Maky for their constant support and encouragement. Thank you all for your invaluable support and for being an integral part of this journey.

Sincerely,

Imane

# Table of Contents

# List of Figures

# Glossary

**CSD** Constrained Spherical Deconvolution.
**CSF** Cerebrospinal Fluid.

**DIPY** Diffusion Imaging In Python.
**dMRI** diffusion Magnetic Resonance Imaging.
**DTI** Diffusion Tensor Imaging.

**EEG** Electroencephalography.

**FA** Fractional Anisotropy.
**fMRI** functional Magnetic Resonance Imaging.
**FOD** fiber Orientation Distribution.
**fODF** fiber Orientation Distribution Function.

**GFA** Generalized Fractional Anisotropy.
**GM** Gray Matter.

**MD** Mean Diffusivity.
**MRA** Magnetic Resonance Angiography.

**PD** Parkinson's Disease.

**QBI** Q-ball Imaging.

**rsfMRI** resting state functional Magnetic Resonance Imaging.
**RUMBA-SD** Robust and Unbiased Model-Based Spherical Deconvolution.

**TDA** Topological Data Analysis.

**WM** White matter.

# Internship site

During the first year of my master's in Healthcare Data Science, I undertook an 18-week internship at the Lille Neuroscience & Cognition Center, led by Professor L. BUEE. The center is also acknowledged as a Joint Research Unit, UMR-S U1172, comprising 5 INSERM-accredited teams in partnership with the University of Lille and the CHU of Lille. I was specifically placed in the team focused on "Degenerative and Vascular Cognitive Disorders," led by Professor D. DEVOS. My internship was supervised by Dr A. Rolland and Dr A. Broustet. The Laboratory of Medical Pharmacology is a multidisciplinary team that focuses on clinical and pre-clinical studies in neurology, pharmacology, and imaging. The list of the DVCD team members and the organizational chart can be viewed in figure 1 and figure 2. For a comprehensive list and additional details, please visit the Lille Neuroscience and Cognition website.

The DVCD team's scientific goals focus on the growing societal concern of cognitive impairments stemming from neurological and psychiatric conditions. The prevalence of cognitive decline is projected to double within the next quarter-century, largely due to increased life expectancy. Vascular and metabolic factors significantly contribute to these cognitive issues. The risks amplify for individuals with conditions like strokes, Alzheimer's, or Parkinson's disease. To address these challenges, the team organizes its work into key projects, including Dr. M. BASTIDE's research on metabolic disorders and cognition, Pr. R. BORDET's studies on post-stroke cognitive issues, Dr. DEVEDJIAN's work on ferroptosis and neuronal death, and Pr. DEVOS's exploration of innovative cerebral treatment methodologies.

During my internship period, I had the opportunity to interact with different actors, including PhD students, assistant engineers, research engineers, physicians, and researchers. And as part of the DVCD team, my project focuses on a mathematical approach to characterize a new biomarker in Parkinson's disease using diffusion MRI data.

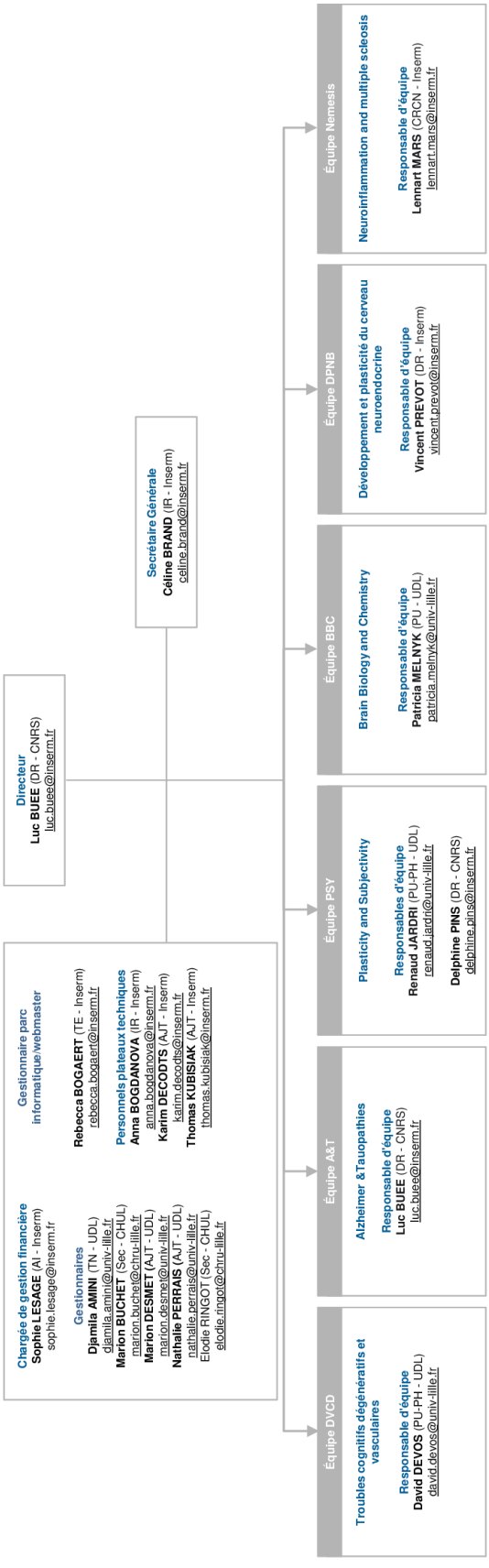
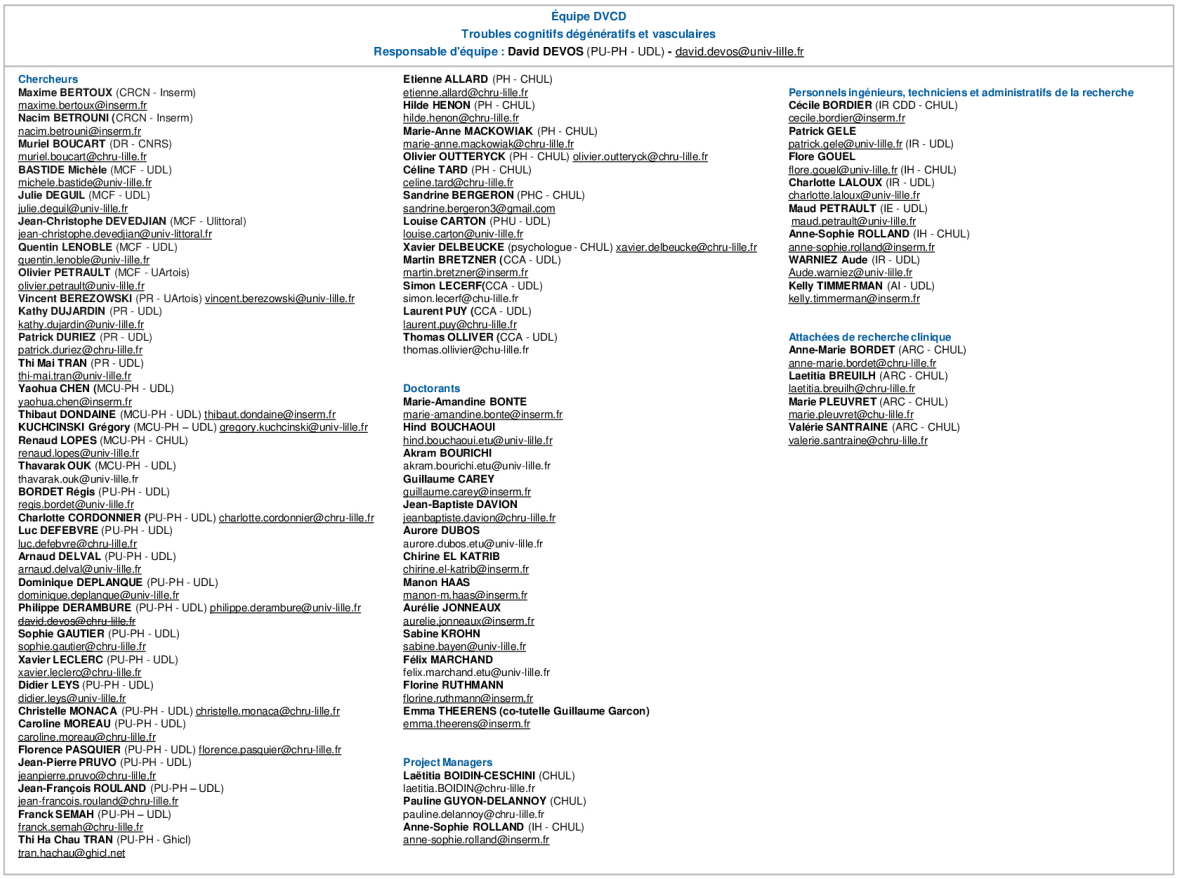



Figure 1: General organizational chart of Lille Neuroscience & Cognition, Medical Pharmacology team U1172



**Équipe DVCD**
**Troubles cognitifs dégénératifs et vasculaires**
**Responsable d'équipe : David DEVOS** (PU-PH - UDL) **-** david.devos@univ-lille.fr

**Chercheurs**
**Maxime BERTOUX** (CRCN - Inserm)
maxime.bertoux@inserm.fr
**Nacim BETROUNI** (CRCN - Inserm)
nacim.betrouni@inserm.fr
**Muriel BOUCART** (DR - CNRS)
muriel.boucart@chru-lille.fr
**BASTIDE Michèle** (MCF - UDL)
michele.bastide@univ-lille.fr
**Julie DEGUIL** (MCF - UDL)
julie.deguil@univ-lille.fr
**Jean-Christophe DEVEDJIAN** (MCF - Ulittoral)
jean-christophe.devedjian@univ-littoral.fr
**Quentin LENOBLE** (MCF - UDL)
quentin.lenoble@univ-lille.fr
**Olivier PETRAULT** (MCF - UArtois)
olivier.petrault@univ-lille.fr
**Vincent BEREZOWSKI** (PR - UArtois) vincent.berezowski@univ-lille.fr
**Kathy DUJARDIN** (PR - UDL)
kathy.dujardin@univ-lille.fr
**Patrick DURIEZ** (PR - UDL)
patrick.duriez@chru-lille.fr
**Thi Mai TRAN** (PR - UDL)
thi-mai.tran@univ-lille.fr
**Yaohua CHEN** (MCU-PH - UDL)
yaohua.chen@inserm.fr
**Thibaut DONDAINE** (MCU-PH - UDL) thibaut.dondaine@inserm.fr
**KUCHCINSKI Grégory** (MCU-PH – UDL) gregory.kuchcinski@univ-lille.fr
**Renaud LOPES** (MCU-PH - CHUL)
renaud.lopes@univ-lille.fr
**Thavarak OUK** (MCU-PH - UDL)
thavarak.ouk@univ-lille.fr
**BORDET Régis** (PU-PH - UDL)
regis.bordet@univ-lille.fr
**Charlotte CORDONNIER** (PU-PH - UDL) charlotte.cordonnier@chru-lille.fr
**Luc DEFEBVRE** (PU-PH - UDL)
luc.defebvre@chru-lille.fr
**Arnaud DELVAL** (PU-PH - UDL)
arnaud.delval@univ-lille.fr
**Dominique DEPLANQUE** (PU-PH - UDL)
dominique.deplanque@univ-lille.fr
**Philippe DERAMBURE** (PU-PH - UDL) philippe.derambure@univ-lille.fr
~~david.devos@chru-lille.fr~~
**Sophie GAUTIER** (PU-PH - UDL)
sophie.gautier@chru-lille.fr
**Xavier LECLERC** (PU-PH - UDL)
xavier.leclerc@chru-lille.fr
**Didier LEYS** (PU-PH - UDL)
didier.leys@univ-lille.fr
**Christelle MONACA** (PU-PH - UDL) christelle.monaca@chru-lille.fr
**Caroline MOREAU** (PU-PH - UDL)
caroline.moreau@chru-lille.fr
**Florence PASQUIER** (PU-PH - UDL) florence.pasquier@chru-lille.fr
**Jean-Pierre PRUVO** (PU-PH - UDL)
jeanpierre.pruvo@chru-lille.fr
**Jean-François ROULAND** (PU-PH – UDL)
jean-francois.rouland@chru-lille.fr
**Franck SEMAH** (PU-PH – UDL)
franck.semah@chru-lille.fr
**Thi Ha Chau TRAN** (PU-PH - GhicI)
tran.hachau@ghicl.net

**Etienne ALLARD** (PH - CHUL)
etienne.allard@chru-lille.fr
**Hilde HENON** (PH - CHUL)
hilde.henon@chru-lille.fr
**Marie-Anne MACKOWIAK** (PH - CHUL)
marie-anne.mackowiak@chru-lille.fr
**Olivier OUTTERYCK** (PH - CHUL) olivier.outteryck@chru-lille.fr
**Céline TARD** (PH - CHUL)
celine.tard@chru-lille.fr
**Sandrine BERGERON** (PHC - CHUL)
sandrine.bergeron3@gmail.com
**Louise CARTON** (PHU - UDL)
louise.carton@univ-lille.fr
**Xavier DELBEUCKE** (psychologue - CHUL) xavier.delbeucke@chru-lille.fr
**Martin BRETZNER** (CCA - UDL)
martin.bretzner@inserm.fr
**Simon LECERF**(CCA - UDL)
simon.lecerf@chu-lille.fr
**Laurent PUY** (CCA - UDL)
laurent.puy@chru-lille.fr
**Thomas OLLIVER** (CCA - UDL)
thomas.ollivier@chu-lille.fr

**Doctorants**
**Marie-Amandine BONTE**
marie-amandine.bonte@inserm.fr
**Hind BOUCHAOUI**
hind.bouchaoui.etu@univ-lille.fr
**Akram BOURICHI**
akram.bourichi.etu@univ-lille.fr
**Guillaume CAREY**
guillaume.carey@inserm.fr
**Jean-Baptiste DAVION**
jeanbaptiste.davion@chru-lille.fr
**Aurore DUBOS**
aurore.dubos.etu@univ-lille.fr
**Chirine EL KATRIB**
chirine.el-katrib@inserm.fr
**Manon HAAS**
manon-m.haas@inserm.fr
**Aurélie JONNEAUX**
aurelie.jonneaux@inserm.fr
**Sabine KROHN**
sabine.bayen@univ-lille.fr
**Félix MARCHAND**
felix.marchand.etu@univ-lille.fr
**Florine RUTHMANN**
florine.ruthmann@inserm.fr
**Emma THEERENS (co-tutelle Guillaume Garcon)**
emma.theerens@inserm.fr

**Project Managers**
**Laëtitia BOIDIN-CESCHINI** (CHUL)
laetitia.BOIDIN@chru-lille.fr
**Pauline GUYON-DELANNOY** (CHUL)
pauline.delannoy@chru-lille.fr
**Anne-Sophie ROLLAND** (IH - CHUL)
anne-sophie.rolland@inserm.fr

**Personnels ingénieurs, techniciens et administratifs de la recherche**
**Cécile BORDIER** (IR CDD - CHUL)
cecile.bordier@inserm.fr
**Patrick GELE**
patrick.gele@univ-lille.fr (IR - UDL)
**Flore GOUEL**
flore.gouel@univ-lille.fr (IH - CHUL)
**Charlotte LALOUX** (IR - UDL)
charlotte.laloux@univ-lille.fr
**Maud PETRAULT** (IE - UDL)
maud.petrault@univ-lille.fr
**Anne-Sophie ROLLAND** (IH - CHUL)
anne-sophie.rolland@inserm.fr
**WARNIEZ Aude** (IR - UDL)
Aude.warniez@univ-lille.fr
**Kelly TIMMERMAN** (AI - UDL)
kelly.timmerman@inserm.fr

**Attachées de recherche clinique**
**Anne-Marie BORDET** (ARC - CHUL)
anne-marie.bordet@chru-lille.fr
**Laetitia BREUILH** (ARC - CHUL)
laetitia.breuilh@chru-lille.fr
**Marie PLEUVRET** (ARC - CHUL)
marie.pleuvret@chu-lille.fr
**Valérie SANTRAINE** (ARC - CHUL)
valerie.santraine@chru-lille.fr

Figure 2: The organization of the Degenerative and Vascular Cognitive Disorders (DVCD) team.

# Introduction

Parkinson's Disease (PD) is a progressive neurodegenerative disorder that affects millions of individuals worldwide. It is characterized by a wide range of motor and non-motor symptoms, including tremors, rigidity, bradykinesia, and postural instability, which significantly impact the quality of life of patients. While extensive research has been conducted to understand the underlying mechanisms and develop effective diagnostic tools for PD, the complexity and heterogeneity of the disease present significant challenges.

In recent years, advanced data analysis, particularly artificial intelligence, has been instrumental in enhancing our understanding of diseases like PD. Biomarkers, whether prognostic, theranostic, and derived from biological or imaging sources, have gained prominence. Topological Data Analysis (TDA) is a technique that has been effectively used to uncover hidden patterns and structures in datasets from various fields, such as neuroscience for brain connectivity analysis, molecular biology for protein structure research, and engineering for sensor network design. By utilizing concepts from algebraic topology, TDA enables the extraction of topological features, such as connected components, loops, and voids, from high-dimensional data.

The objective of this project is to use TDA to uncover topological features that can help to differentiate the different stages of the Parkinson's disease progression, but also discriminate between parkinsonians and healthy individuals. The identification and analysis of these topological features have the potential to characterize a new biomarker that can help in patient prognosis and stratification to improve patient's selection in clinical trials.

In this report, we will present the methodology used in the study, including the dataset, the TDA techniques applied, and the analysis procedures undertaken. We will then present and discuss the findings obtained through the application of TDA, along with their interpretation and relevance in the context of Parkinson's Disease. Furthermore, we will highlight the potential contributions and limitations of our approach. Finally, we will summarize the key findings and highlight potential future research directions based on the outcomes of this study.

Chapter 1

# Parkinson's Disease and brain connectivity

## 1.1 Overview of Parkinson's Disease

PD is a progressive neurodegenerative disorder that primarily affects automacity. Its occurrence is widespread, making it the second most common neurodegenerative disease after Alzheimer's disease [28]. The pathological hallmark of PD is the degenerescence of the nigro-striatal pathway, a region of the brain involved in motor control, and the presence of Lewy bodies, which are protein aggregates [26]. (Figure 1.1)

The cardinal motor symptoms of PD are typically bradykinesia (slowness of movement), resting tremor, rigidity, and postural instability[18]. In addition, a range of non-motor symptoms such as cognitive impairment, mood disorders, sleep disturbances, and autonomic dysfunction are also associated with the disease[32].

The etiology of PD is multifactorial, with both genetic and environmental factors contributing to the disease risk. Several genes have been identified that, when mutated, can cause familial forms of PD. However, these genetic cases represent only 10% of all PD cases. Most are sporadic, with aging being the primary risk factor. Some environmental factors, such as exposure to pesticides, have also been associated with increased PD risk [3].

Despite extensive research, the exact mechanisms leading to the neurodegeneration in PD are still not fully understood. However, several ones have been implicated, including oxidative stress, mitochondrial dysfunction, protein misfolding, and neuroinflammation[24] .

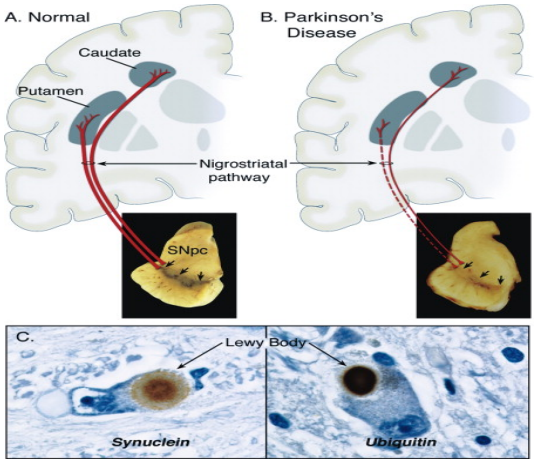



Figure 1.1: Neuropathology of Parkinson's Disease
(A) Nigro-striatal pathway in healthy people (shown in red). Dopaminergic neurons of the substantia nigra pars compacta (SNpc), shown by arrows send axons (thick red lines) to the striatum (putamen and caudate nucleus). We can notice black neuronal pigment in these neurons, due to the presence of neuromelanin (B) Degenerated nigro-striatal pathway in parkinsonian patient. The pathway (shown in red) breaks down. Connectivities with the putamen are lost (indicated by dashed lines), with comparatively reduced impairment extending to the caudate nucleus(shown with a thin red solid line). (C) Brain aggregates: Two examples of Lewy bodies (black arrows), ones with synuclein aggregates and ones with ubiquitin aggregates, [11]

## 1.2 Altered Brain Connectivity in Parkinson's Disease

### 1.2.1 Introduction to Brain Connectivity

Brain connectivity, the pattern of anatomical links or functional interactions between different areas in the brain, is a fundamental aspect of neuroscience research. The study of brain connectivity allows us to understand the organization of the brain network, providing insights into how different brain regions interact and coordinate to carry out various cognitive and motor functions [6].

There are two main types of brain connectivity: structural and functional. Structural connectivity refers to the physical (anatomical) connections between brain regions, primarily formed by white matter tracts. On a larger scale, the term "structural connectome" is used to describe the comprehensive map of these neural connections throughout the brain, representing how the different regions of the brain are wired. Functional connectivity refers to the statistical dependencies or correlations between neurophysiological events in different brain regions, which can be inferred from neuroimaging techniques like functional Magnetic Resonance Imaging (fMRI) [14].

Brain connectivity shows dynamic changes in response to various factors such as learning, aging, and disease. Therefore, the analysis of brain connectivity provides a powerful tool for investigating normal brain function and the alterations associated with different neurological and psychiatric conditions, including PD [27].

In PD, there is increasing evidence that the disease affects not only specific regions in the brain but also the connectivity between them. This has led to a shift in focus from a region-based approach to a network-based approach [34] .

### 1.2.2 Structural connectivity and diffusion MRI

#### Diffusion MRI

Diffusion MRI (dMRI) is a neuroimaging technique that studies the movement of water molecules within biological tissues. By tracking the motion of water molecules, dMRI provides insights into the microstructural properties of tissues. This information is crucial for understanding the connectivity and organization of white matter fibers in the brain. Diffusion MRI has paved the way for advanced techniques like Diffusion Tensor Imaging (DTI) and Constrained Spherical Deconvolution (CSD), which enhance the accuracy and resolution of structural connectivity mapping[4].

#### Diffusion Tensor Imaging and Constrained Spherical Deconvolution in Assessing Stuctural Brain Connectivity

Within the brain's white matter, water diffusion is anisotropic due to the constraints imposed by axon membranes, which limit molecular movement across the fibers perpendicular direction. Exploiting this characteristic, diffusion tensor imaging (DTI) generates intricate insights into the microstructure of white matter pathways, thus providing a method to assess structural connectivity[4]. In recent years, DTI has been extensively used to explore the underlying brain changes in Parkinson's disease [20].

While DTI offers valuable information about the average direction of diffusion in each voxel, it can face challenges with complex fiber configurations. More advanced models, such as Constrained Spherical Deconvolution (CSD), provide a better representation of multiple fiber orientations within a single voxel, offering a more detailed view of brain connectivity (figure 1.2).

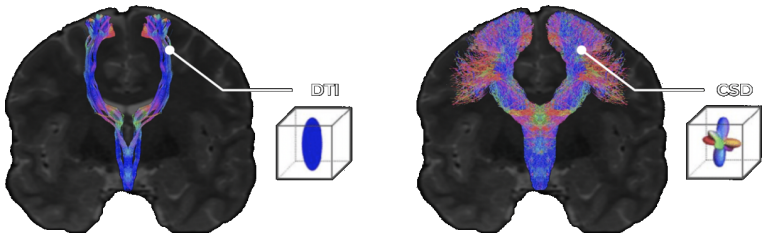



Figure 1.2: DTI model vs CSD model in simulating fiber behavior and reconstructing the Corticospinal Tract.
The DTI method assumes a single predominant fiber direction within a voxel, while the CSD technique deals with multiple intersecting fibers within each voxel, [37]

#### Abnormality in Structural Brain Connectivity in Parkinson's Disease

In the context of PD, these imaging techniques have shown an altered structural connectome. Specific white matter tracts may display reduced integrity, and network analyses might reveal altered patterns of connectivity, reflecting the widespread impact of PD on the brain's intricate wiring.

Multiple DTI studies have consistently reported alterations in the white matter integrity of Parkinsonian patients. Reduced fractional anisotropy (FA), an indicator of fiber density, axonal diameter, and myelination in white matter; increased mean diffusivity (MD), reflecting the overall rate of water diffusion, have been observed in various brain regions including the substantia nigra, corpus callosum, cingulum, and other areas involved in motor and non-motor functions [39]; [16]. The elevated MD suggests potential changes in tissue microstructure, possibly reflecting overall cell loss or other types of neural damage[16] .

Beyond these specific measures, an intriguing pattern has emerged from the analysis of whole-brain network topology using graph theory metrics. Specifically, the "small-world" architecture that normally characterizes brain networks appears to be altered in PD. Tao Wu and collaborators have shown that brain networks in PD shift towards a more regular configuration compared to the healthy balance between integration and segregation. This could imply disruptions in the network connectivity and potentially less efficient information transfer in PD[38].

Notably, these structural alterations have been found to correlate with the severity of motor and cognitive symptoms in PD patients, suggesting their potential as biomarkers for disease progression and cognitive decline [2].

While the aforementioned studies have provided valuable insights into the disease's pathophysiology, there remains a need for more comprehensive and longitudinal studies, especially given the considerable interindividual variability in the clinical presentation and progression of Parkinson's disease.

Chapter 2

# Topological Data Analysis

## 2.1 Introduction to Topological Data Analysis

The development of novel biomarkers for Parkinson's disease prognosis is crucial to enhance our predictive capabilities. In this context, we introduce topological data analysis as a promising approach in investigating brain networks, offering a potential imaging biomarker for advancing our prognosis of the disease.

### 2.1.1 The Concept of Topological Data Analysis

Topological Data Analysis (TDA) is an innovative area in the field of applied mathematics, bridging the gap between data analysis and topology. Topology is the branch of mathematics concerned with properties of space that are preserved under continuous transformations, such as stretching or bending. In simple terms, it focuses on the study of "shape". On the other hand, TDA brings the concepts and principles of topology into the realm of data analysis to extract meaningful insights from complex, multidimensional datasets [8].

The central idea of TDA revolves around identifying the shape inherent in data. In contrast to traditional analytical approaches that may lose essential information due to dimensionality reduction or assumptions about data distribution, TDA seeks to uncover and quantify the hidden, often complex, structure within data. This involves understanding patterns in data from a topological point of view and investigating features like connected components, holes, voids, and their multidimensional analogs in the data[9](figure 2.1).

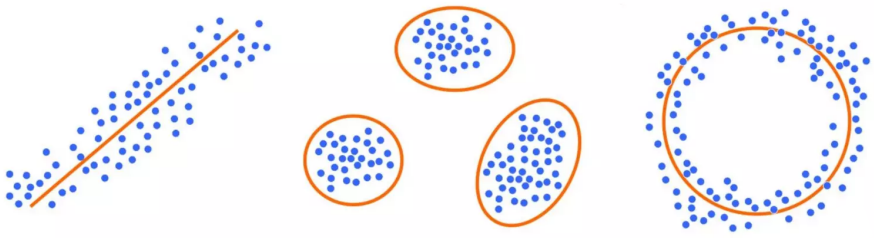


Figure 2.1: Features extraction using TDA
TDA studies the shape of data and investigates features like connected components (left, middle) and holes(right), [7]

A key attribute of TDA is its ability to interpret data at multiple scales and its robustness against noise. Traditional statistical methods often require a fixed scale of observation and are sensitive to

outliers. TDA, however, offers a multiscale lens to view data, capturing persistent structures that exist over a range of scales. This makes it particularly useful for examining complex and high-dimensional datasets [8].

The utility of TDA extends across various fields including but not limited to neuroscience, genetics, image processing, and social network analysis.

### 2.1.2 Applications in Neuroscience and Neuroimaging

The application of TDA to neuroscience and neuroimaging is an emerging and promising research area. As neuroscience data are complex, high-dimensional, and often nonlinear, TDA provides a valuable set of tools to understand the topological and geometric structure underlying this data. Moreover, TDA provides a means to integrate multiple types of neuroscience data : structural, functional, and molecular, into a unified analytical framework. This multi-modal analysis can yield more insights into brain organization and function.

TDA has been particularly valuable in analyzing structural and functional brain networks. In the context of structural brain connectivity, TDA can be employed to investigate white matter tracts derived from diffusion tensor imaging (DTI). It can reveal the "shape" of these intricate structures, such as loops and branches, which may be critical in understanding the organization and routing of brain networks[22].

For functional connectivity, TDA can elucidate the topology of functional brain networks captured through techniques such as functional MRI (fMRI) and electroencephalogram (EEG). By examining the topology of these networks, TDA can help uncover the organizational principles of brain function and identify changes associated with various neurological and psychiatric conditions [31].

TDA helped in characterzing the topological alterations in brain networks that occur due to conditions such as Alzheimer's disease, Parkinson's disease, and schizophrenia. For instance, using TDA, Zhu and al have been able to distinguish between healthy and disease states based on the different topological properties of the brain networks[40] .

#### Use of TDA in Exploring Disease Patterns and Subgroups

One of the interesting applications of TDA is its ability to help understand the structure of the human brain and the changes that occur due to diseases such as Parkinson's. A relevant example is the study by P. Bendich, J. S. Marron and collaborators [5] where TDA was used to examine the structure of brain arteries.

A topological model of the arteries in the brain based on Magnetic Resonance Angiography (MRA) data was contructed using TDA. This approach allowed them to characterize the overall shape of the arterial structure and identify key topological features. Importantly, they demonstrated that TDA could detect subtle, yet critical changes in the vascular structure that could be associated with diseases. Through the analysis, the researchers were able to discern distinct differences in the topological features between healthy individuals and those with certain conditions, such as ischemic stroke. They found that the TDA-derived measures were not only able to distinguish between these groups but also showed a significant correlation with the severity of the stroke symptoms.

These findings underscore the potential of TDA to uncover disease patterns and distinguish between patient subgroups based on the topological properties of their brain imaging data. This suggests that a similar approach could be used to explore the changes in brain connectivity and other neural structures in PD. Given the complexities associated with PD, such as the broad spectrum of motor and non-motor symptoms and the heterogeneous progression of the disease, TDA could provide invalu-

able insights by identifying disease subtypes or patient clusters and by establishing correlations with the disease severity. These insights could lead to more personalized treatment strategies and improved prognostic models.

## 2.2 TDA Techniques and Methods

TDA encompasses a wide array of methods that provide insights into the underlying shapes (or topological structures) of data. Examples include Mapper, persistent homology, zigzag persistence, and Reeb graphs, among others. For this project, we choose to focus on persistent homology.

### 2.2.1 Persistent Homology

#### Definition

Persistent Homology is a mathematical method in TDA that identifies and quantifies topological features of a dataset throughout various scales. The underlying principle of this method is the notion of "persistence", which refers to the lifespan of topological features as one varies the scale of observation.

In a dataset, topological features such as connected components (0-dimensional holes), loops (1-dimensional holes), voids (2-dimensional holes), and their higher-dimensional analogs can be detected. Persistent Homology, as a multiscale tool, systematically monitors these features, tracking their birth (when they first appear) and death (when they vanish) as the scale changes [12] .

The concept of persistence is based on the idea that features which persist over a wide range of scales are likely to be inherent properties of the dataset, whereas features that appear or disappear quickly (at a specific, often smaller scale) might be considered noise or artifacts [15].

Overall, Persistent Homology is a powerful tool for capturing and quantifying the complex topological structure of data, allowing for a comprehensive analysis of the "shape" hidden within the data.

#### Computation and Visualization of Persistent Homology

The computation and visualization of Persistent Homology are critical steps allowing the translation of complex topological information into a more intuitive and accessible format. The computation of Persistent Homology begins with the construction of a mathematical object known as a simplicial complex. This is done by creating a "filtration", a nested family of spaces that are built by gradually adding simplices (vertices, edges, triangles, and their higher-dimensional counterparts) based on a specific rule. Often, this rule is related to a distance metric (like Euclidean distance) between points in the data. As the scale parameter of the filtration changes, new simplices are added, giving birth to topological features or causing others to die (merge). A key concept here is that of homology groups, which algebraically encapsulate the topological features of each space in the filtration [13]. The results of this computation are typically visualized in the form of persistence diagrams or barcodes, which display the "life story" of topological features (figure 2.2).

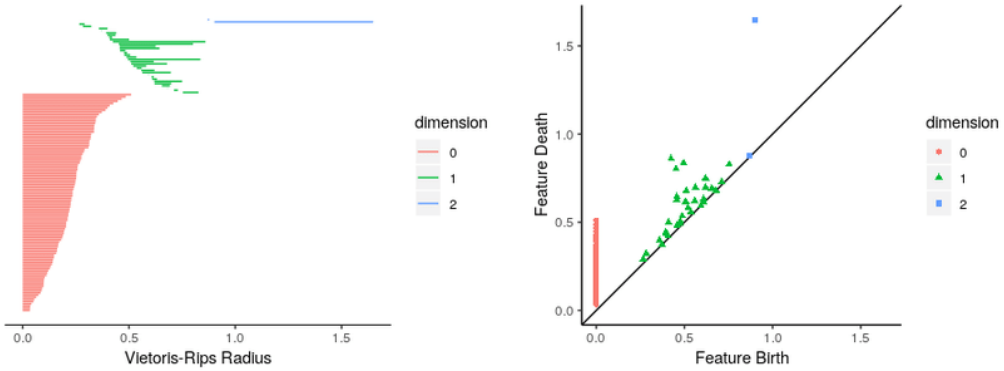



Figure 2.2: Two different representations of persistent homology

Topological barcode representation (on the left) and the corresponding persistence diagram (on the right). The cycles of dimension 0 are depicted in red, those of dimension 1 in green, and dimension 2 cycles are illustrated in blue, [36]

**Persistent diagrams** In this project, we used persistence diagrams to present our persistent homology results. As seen in 2.2 Persistent Diagrams are a primary tool in Persistent Homology for visualizing topological features of a dataset. In these diagrams, each point represents a unique topological feature identified in the data. The x-coordinate of a point corresponds to the "birth" time of the feature, which is the scale at which the feature first appears during the filtration process. The y-coordinate corresponds to the "death" time of the feature, marking the scale at which the feature disappears. Essentially, the persistent diagram is a scatter plot in the plane, where the "persistence" of a feature (the difference between its death and birth times) is represented by the vertical distance of a point from the line y=x (the diagonal) [14].

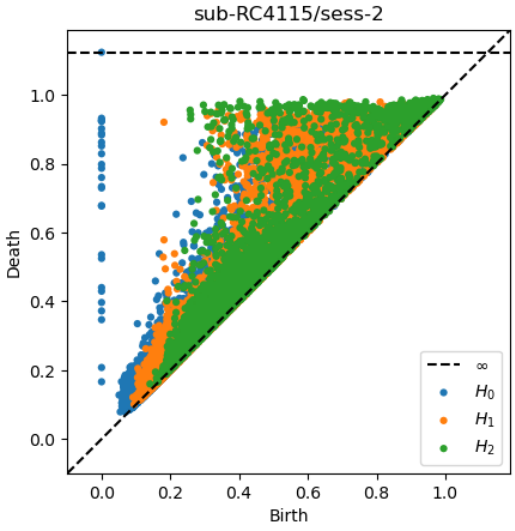



Figure 2.3: Persistence diagram representation of a healthy control/MRI dataset

Generated using GFA (General Fractional Anisotropy) data and the Ripser library in Python, this persistence diagram represents a control participant sourced from the OpenNeuro dataset ds001907. Blue, orange and green points represent respectively cycles of dimension 0, 1 and 2.

**Interpreting Persistent Homology Diagrams** The key principle in interpreting these diagrams lies in the concept of "persistence", the vertical distance from a point to the diagonal line y=x. Features with high persistence, i.e., those that exist over a wide range of scales, are represented by points far from the diagonal. These points are often considered as signal or inherent structure in the data, as they indicate topological features that are robust to changes in scale. Hence, the presence of such features often carries significant implications about the underlying structure of the data [13].

On the other hand, points located close to the diagonal represent features with low persistence, i.e., those that exist only briefly as the scale changes. These points are usually considered as noise or artifacts, as they may be the result of random variations or measurement errors in the data [10].

In general, interpreting persistent diagrams involves recognizing this distinction between noise and signal, which is fundamental in extracting meaningful insights from the data using Persistent Homology.

### Persistent images

**Definition** Persistent Images are another method of visualizing the results of Persistent Homology that transform the scatter plot-like persistence diagrams into a more uniform, image-like representation. The method was developed as a means to address a common challenge in working with persistence diagrams: while they offer a wealth of topological information, their irregular and discrete nature can make subsequent data analysis difficult[1].

The process of creating persistent images involves smearing each point in the persistence diagram with a small Gaussian kernel (a bell-shaped curve centered at the point). Then, a weighted sum of these Gaussian kernels is taken, and the resulting "heat map" is discretized into a regular grid of pixels to create the persistent image. The pixel values in the persistent image represent the presence of topological features with specific birth and death times, essentially creating a density estimate of features in the persistence diagram[1].

One of the primary advantages of persistent images is that they transform the topological information of persistence diagrams into a format that can be easily integrated into standard machine learning algorithms. By converting persistence diagrams into these structured, uniform images, it is possible to feed topological data directly into image processing pipelines and machine learning models, which often require fixed-size inputs[1](figure 2.4).

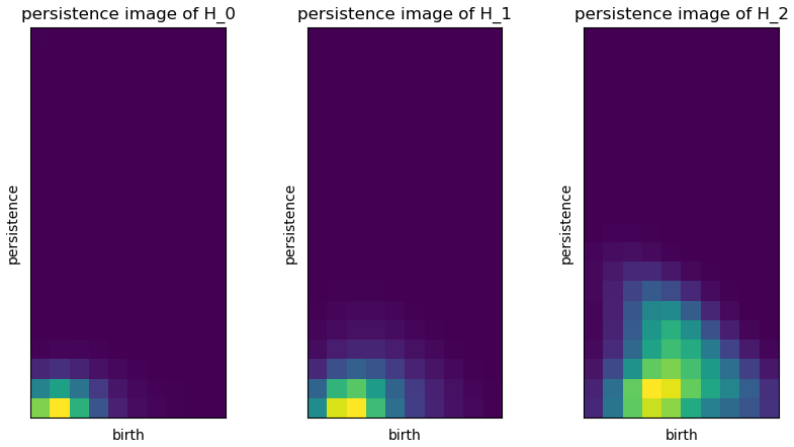



Figure 2.4: Persistence image representation of a healthy control/MRI dataset
Illustrated here are persistence images derived from the earlier persistence diagram (figure2.3), wherein each degree of homology was transformed into a distinct persistence image using the Persim library in Python

**Interpreting Persistent Homology Images** Interpreting Persistent Homology Images requires an understanding of their construction process and what the represented density values mean.

As mentionned previously, the pixel values in a persistent image represent the density of topological features that have specific birth and death times [1].

Areas of the image that are brighter (have higher pixel values) indicate regions where topological features are densely concentrated in the corresponding persistence diagram. These areas could represent robust topological structures that persist across multiple scales in the data[1].

Whereas, darker areas (with lower pixel values) correspond to regions with fewer topological features, suggesting a lack of significant structure at those particular scales[1].

Chapter 3

# Extracting the topological features using fODFs and Persistent Homology

The fiber orientation distribution functions (fODFs), which represent high-dimensional features at each voxel, can be thought of as data points in a high-dimensional space. The goal of TDA is to uncover the "shape" of this data, which represents the topological structure of the brain's white matter connectivity.

## 3.1 Definition and Importance of fODFs

fODFs are crucial elements in the analysis of brain white matter connectivity using diffusion MRI data(dMRI). At the most basic level, an fODF is a 3D function that describes the distribution of fiber orientations within a voxel. This distribution arises from the complex architecture of brain white matter, where axonal fibers can cross, touch, branch or fan within the same voxel (figure 3.1).

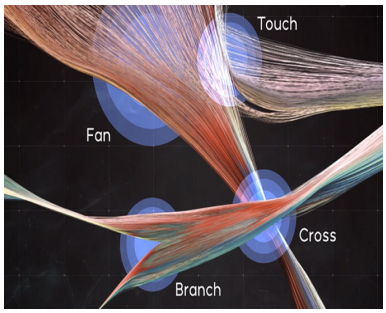



Figure 3.1: Complex axonal fibers configurations
Complex fiber orientations can arise in various situations. Instances involving crossed fibers, branching, and fanning require a more robust fiber orientation model to accurately represent and construct tractography, [37]

In dMRI, the complex configurations of white matter tracts lead to intricate diffusion profiles that are not easily captured by simpler models such as diffusion tensor imaging (DTI). Addressing these challenges, various advanced dMRI techniques, such as Q-ball Imaging (QBI) and Constrained Spherical Deconvolution (CSD) have been developed to estimate the fODF within a voxel [35].

## 3.2 Role of fODFs in Characterizing Brain Connectivity

The understanding of brain connectivity has been fundamentally reshaped with the advent of fiber Orientation Distribution Functions (fODFs) and their applications in dMRI. As described previously, fODFs provide a more comprehensive representation of the complex intra-voxel fiber architectures, offering a significant step forward in modeling and visualizing brain connectivity.

The advantages of advanced techniques that estimate fODFs over simpler models include:

Resolving crossing fibers: Traditional methods like diffusion tensor imaging (DTI) treat each voxel as having a single, dominant fiber direction, which becomes a considerable limitation when we consider that complex fiber configurations which are believed to occur in up to 90% of brain voxels. Newer methods like CSD or QBI, overcome this limitation by representing the diffusion signal as a distribution of fiber orientations, thereby revealing the multiple fiber populations within a voxel [19]. This improved representation enhances the detection and characterization of complex fiber configurations, leading to a better understanding of the intricate connective structure of the brain.

Reduced false positives in tractography: The application of CSD and other advanced techniques for fODFs estimation has profound impacts on tractography e.g. the process of mapping the neuronal fiber pathways in the brain. By resolving the directions of multiple fibers within a voxel, fODFs provide more accurate guidance for tractography algorithms, resulting in more realistic and comprehensive fiber tracking. Consequently, the connectomes generated from fODF-based tractography can capture the brain's wiring more accurately, paving the way for more sophisticated analyses of brain connectivity and its alterations in different diseases, including Parkinson's disease[30].

Stronger correlation with histological measures: Techniques that estimate better fODFs have been shown to have a closer correlation with histological measures than simpler diffusion models, positioning them as more accurate tools for studying brain connectivity and potential pathologies [33].

Informing Microstructural Measures: Another notable role of fODFs in characterizing brain connectivity involves their utility in deriving microstructural measures. For instance, the shape and amplitude of the fODF within a voxel can provide information about the density and arrangement of the fibers, thus offering microstructural insights that are crucial for understanding brain function and pathology [29].

## 3.3 Data Representation and Preprocessing for Persistent Homology in the context of Parkinson's Disease

### 3.3.1 The pipeline

**1. Data acquisition :** The dataset used in this first approach was sourced from the open-access database OpenNeuro, accession number ds001907, a rich multi-modal neuroimaging dataset dedicated to the study of PD.

The ds001907 dataset investigate the multimodal neuroimaging biomarkers of Parkinson's Disease. It includes high-quality structural (T1-weighted), diffusion MRI, and rsfMRI (resting state fMRI) data, and also detailed demographic and clinical information for all subjects, including age, gender, and disease status, along with the Unified Parkinson's Disease Rating Scale (UPDRS) scores.

Specifically, the dMRI data, which is the focus of this internship, was acquired using a multi-shell, high angular resolution diffusion imaging (HARDI) protocol. This type of data is particularly well-suited for advanced diffusion models like the CSD model, offering the opportunity to generate de-

tailed fODF estimates and perform comprehensive tractography analyses.

The dataset includes 3-tesla MRI data from both healthy controls and patients diagnosed with Parkinson's Disease. This allows for direct comparisons of brain connectivity patterns between the two groups, thereby enabling the exploration of connectivity alterations associated with Parkinson's Disease.

**2. Model selection :** Modeling the diffusion signal in brain white matter is a crucial step in studying brain connectivity with diffusion MRI. This process often involves selecting a model that can adequately capture the complex diffusion behavior within brain voxels. For this internship, the model of choice is the RUMBA-SD model, implemented in the open-source software library of Diffusion Imaging in Python (DIPY). Its integration with other tools and functionalities further facilitates comprehensive analyses of brain connectivity.

The RUMBA-SD model is a variant of spherical deconvolution methods, which are specifically designed to estimate Fiber Orientation Distribution Functions from the dMRI signal [21]. These methods aim to "deconvolve" the diffusion signal to reveal the underlying distribution of fiber orientations, hence the name "spherical deconvolution."

RUMBA-SD, in particular, was developed to tackle some of the limitations of earlier spherical deconvolution methods. A key advantage of RUMBA-SD is its ability to handle Rician noise, which is a type of noise inherent in MRI data [23]. By accounting for Rician noise, the RUMBA-SD model can produce more accurate and robust fODF estimates, leading to improved fiber tracking and connectivity analysis.

Furthermore, RUMBA-SD is designed to handle multi-shell diffusion data, meaning data acquired with multiple b-values. This ability to handle multi-shell data allows RUMBA-SD to more fully exploit the rich information available in such datasets, thereby improving the quality of the estimated FODFs. Refer to Appendix RUMBA-SD for details on how i constructed the fODFS.

**3. DATA (fODFs) Visualization :** After the construction of fODFs using the RUMBA-sd model, we proceeded to Data visualization, a key step in our analytical process. Using FURY (an open-source software library designed for visualizing 3D data), the reconstructed fODFs were rendered in a three-dimensional space, allowing for an intuitive understanding of the complex fiber structures within our region of interest.

(a) Isotropic fODF (b) Anisotropic fODF

Figure 3.2: 3D-representation of two main fODF configurations: isotropic and anisotropic. (a) "Isotropic fODF" describes a situation where there is no preferred direction of fiber orientation. Fibers are randomly oriented in all directions, this is represented as a perfect sphere, where the density of fibers is equal in all directions around a central point. (b) "Anisotropic fODF" represents a situation where there is a preferred direction of fiber orientation. The fODF looks elongated in a specific direction, indicating that fibers are predominantly aligned in that direction

Interpretation of a Sample fODF : The central core of the fODF represents the voxel of interest. Each protrusion or "lobe" extending from the central core signifies a predominant fiber orientation within

that voxel. The magnitude (or height) of each lobe indicates the relative density or prominence of fibers in that particular orientation.

In simpler terms, an fODF with multiple distinct lobes suggests that multiple fiber bundles cross or interact within that particular voxel, whereas an fODF with a singular dominant lobe implies a coherent, unidirectional fiber tract.

**4. fODFs Dilation : "Growth Distance"**

We needed a proximity metric to determine how close are two fODFS in a specific direction. The task was then to measure the distance between these fODFs in a way that captures the inherent spatial nature of these orientations. This approach differs from traditional distance measures, such as Euclidean distance, as it takes into account the specific geometric properties of the fODFs.

Inspired by the study "Growth Distances: New Measures for Object Separation and Penetration" by Chong Jin Ong and Elmer Gilbert, we explored a novel method to measure the distances between fODFs, [25]. We adopt the notion of "growth distance" to quantify the difference between various fiber orientations within the brain. To assess this difference we have implemented in Python the concept of "Growth distance" originally introduced in the cited paper.

**Concept of Growth Distance** Growth distance, as introduced by Ong and Gilbert, is a measure that quantifies the amount of uniform enlargement required to make two objects touch [25]. In other words, it assesses the degree of separation between two objects by determining how much these objects need to grow before they intersect. This innovative concept proves particularly useful for evaluating the difference between spatial structures, making it suitable for our fODFs analysis.

**5. Constructing a Growth Distance Matrix :** With the implementation of the growth distance calculation between fODFs, we can build a matrix representing these distances for all pairs of fODFs within our dataset. This growth distance matrix captures the unique topological characteristics of the fODF map and how they compare to others, providing a solid foundation for subsequent topological analysis. Considering the computational complexity involved in measuring the growth distance between fODFs, we opted for a different method to create those distance matrices (chapter 4).

**6. Computation and Visualization of Persistent Homology Diagrams :** The growth distance matrix can be used as a distance matrix to compute persistence homology diagrams.

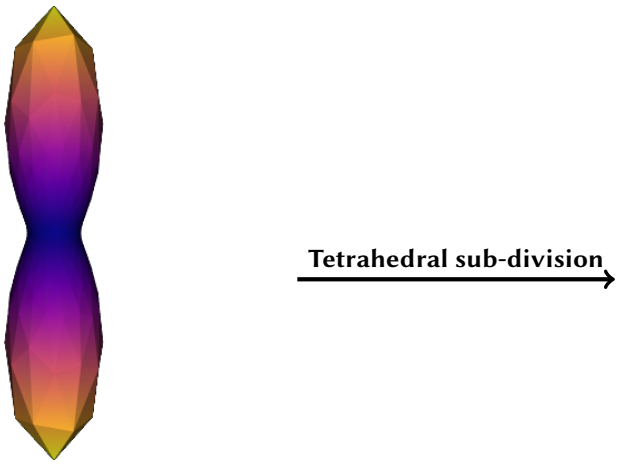
### 3.3.2 Results

**Implementing the Growth Distance for fODFs in Python**

To incorporate the growth distance concept into our study, we used the specific algorithm proposed by Ong and Gilbert in their original research[25] . The algorithm, derived from rigorous algebraic principles, interprets the degree of dilation required for intersection as the "growth distance" between two distinct objects. To serve our research objectives, we translated this abstract algebraic formulation into a concrete Python code. For a detailed look at the Python code used to implement the growth distance algorithm, please refer to the Appendix.
This adapted algorithm was designed to iteratively dilate the fODFs until they intersect. Consequently, the level of dilation necessary for two fODFs to touch is interpreted as the "growth distance" between them, a concept central to our topological characterization of a new biomarker in PD's brain connectivity.

**Tetrahedral subdivision** However, a critical requirement of the growth distance concept is that the objects of study should be convex. Given the complexity of our fODFs, they do not naturally meet this criterion. To solve this issue, we leveraged the way our fODFs data is stored and arranged. In fact, the fODFs data is stored as a collection of interconnected triangles, which can be accessed through the attributes fODF.faces for the triangles and fodf.vertices for the vertices.

Building upon this, we therefore opted to further subdivide each fODF into tetrahedrons by connecting each triangle on the surface to the center of the corresponding fODF (figure 3.3). This division into tetrahedrons essentially transforms each fODF into a series of convex objects, allowing us to use the growth distance concept for our analysis.




(a) fODF Classical representation

(b) fODF representation with Tetrahedral Subdivision

Figure 3.3: Tetrahedral sub-division of an fODF
Illustrated in Fig (a) is a segment of an fODF, which undergoes tetrahedral subdivision by connecting each triangle on the surface to the center of the fODF resulting in the representation shown in Fig (b).

### Computational Complexity of Implementing Growth Distance

The computational cost associated with implementing the growth distance algorithm for our fODFs, once they have been divided into tetrahedrons, is substantial. Here is a brief breakdown:

Each FODF is subdivided into 1444 tetrahedrons. In a single brain scan, there are 128*128*32 = 524 288 voxels. Therefore, the total number of tetrahedrons in a single brain amounts to 524288 * 1444 = 757071872 tetrahedrons. To calculate the growth distance between any two fODFs, we would have to carry out $757071872^2$ calculations.

This computational complexity presents a significant challenge, highlighting the need for efficient algorithmic solutions and high-performance computing resources to process the vast amount of data in a reasonable timeframe.

Chapter 4

# Structural connectome for persistent Homology

Through this second approach, our aim was to determine if there are distinct patterns in the brain's connectivities that might be linked to Parkinson's Disease. And for this purpose we construct the structural connectome using diffusion magnetic resonance imaging data obtained from 5 patients with Parkinson's Disease (PD) - each having undergone 2 sessions - as well as 5 control subjects. The actual data representation that shows the strength of connections between different brain regions is stored in a connectivity matrix. The connectivity matrices are transformed into distance matrices and used to compute persistent homology enabling us to identify the topological variations between the brains of PD subjects and the control group.

## 4.1 Data Representation and Preprocessing for Persistent Homology

### 4.1.1 The pipeline

**1. Data Acquisition:**

We used a small dataset consisting of :

- 5 Parkinson's Disease (PD) subjects from FAIRPARKII(FPII): FPII is a multicenter, phase 2, randomized, double blind international trial (Protocol 2015_22) sponsored by the University Hospital of Lille involving participants with newly diagnosed Parkinson's disease who had never received levodopa and assign to receive oral deferiprone (iron chelator) or matched placebo for 36 weeks. The 5 subjects selected are placebo subjects in order to track the natural progression of PD.
- 5 Controls from the OpenNeuro dataset ds001907 (used in chapter 3).

**2. Preprocessing & Postprocessing of diffusion mri data :**
The preprocessing phase of neuroimaging data analysis is focused on preparing the raw data for further analysis. It involves tasks like noise reduction, data normalization, alignment, and brain extraction. These steps ensure that the data is clean, consistent, and ready for accurate analysis. On the other hand, postprocessing involves using tractography algorithms to trace white matter tracts, constructing a network where brain regions are nodes connected by pathways, and calculating connectivity strengths, resulting in a Connectivity matrix. By using Neuroimaging software like MRtrix, FSL, ANTs, and FreeSurfer, the preprocessing and postprocessing stages were executed. To streamline and automate these processes, we developed dedicated bash scripts. These scripts con-

tain all the necessary bash commands, ensuring the reproducibility and efficiency of our analysis. Refer to the Appendix for the scripts
A detailed breakdown of the steps and methodologies employed during this phase is visually depicted in the accompanying diagram below(figure 4.1).

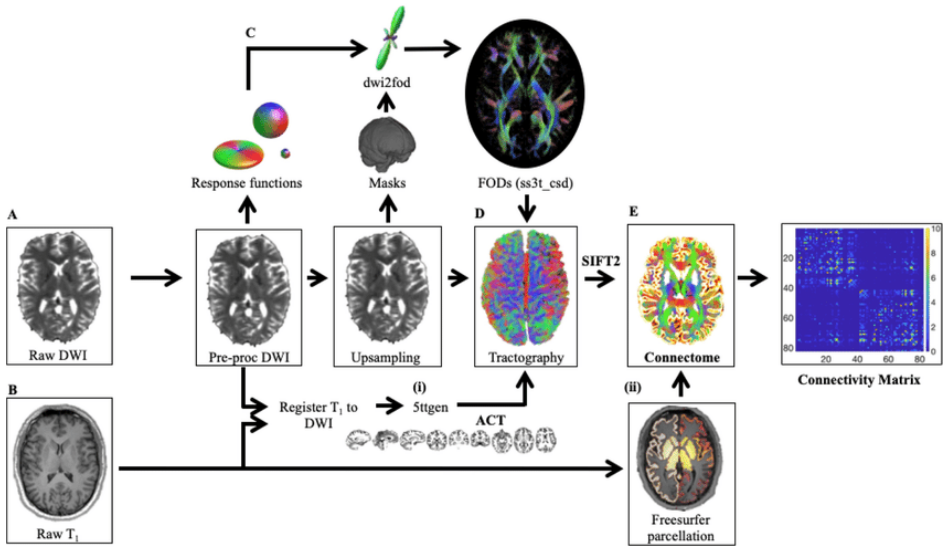
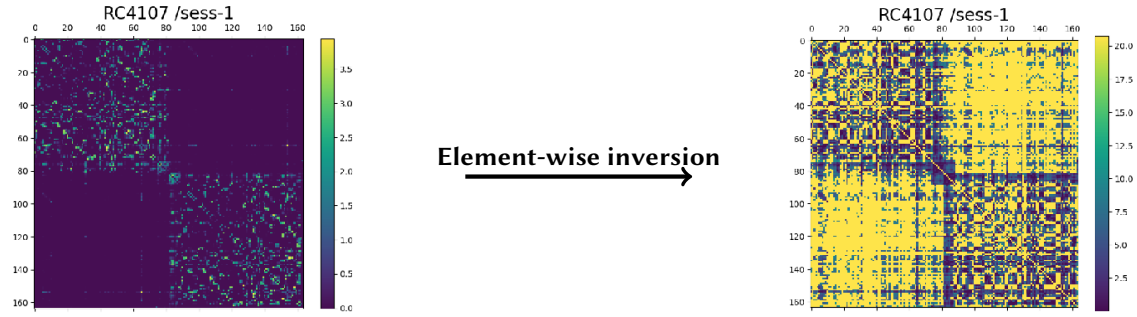



Figure 4.1: Connectome Construction Process

Initial diffusion images undergo preprocessing to eliminate imaging artifacts. This includes removal of noise, Gibbs ringing, motion artifacts, and distortion corrections.b) In parallel, T1 images undergo: i) Registration with diffusion images to generate images for Anatomically Constrained Tractography (ACT) using 5ttgen. ii) Processing through Freesurfer to designate the nodes essential for the subsequent connectome evaluation. c) Fiber Orientation Distributions (FODs) are determined based on the average response functions from the collected data. These images are then up-sampled and normalized for intensity. d) Using the FODs derived in step (c) and the 5ttgen images from step (b(i)), Anatomically Constrained Tractography (ACT) is performed. This is followed by the application of Spherically Informed Filtering of Tractograms (SIFT2) to refine the streamline data. This adjustment ensures the streamline weights align accurately with the detected fiber orientation distributions. e) The finalized connectome is assembled using both the Freesurfer determined divisions and the streamlined and filtered tractogram data,[17]

**3. Conversion to Distance Matrices for TDA:**
Distance matrices are essential in computing Persistent Homology because they provide a quantitative representation of the data's pairwise relationships. These matrices guide the construction of simplicial complexes at different scales, leading to the identification of topological features that persist over a range of parameter values. In the context of brain connectivity matrices, each entry in the matrix typically represents the strength of the connection between two regions in the brain. Thus, one way to convert this connectivity matrix into a format suitable for TDA is to consider each entry as a distance, with stronger connections corresponding to shorter distances. Subsequently, the connectivity matrices underwent coefficient-wise inversion to align with this perspective (figure 4.2).




(a) Connectivity matrix

(b) Distance matrix

Figure 4.2: Connectivity matrix conversion to distance matrix
The connectivity matrix (a) where each voxel representing the strength of the connections between two regions in the brain is converted into a distance matrix (b) through a coefficient-wise inversion, where each coefficient is a distance with stronger connections corresponding to shorter distances.

**4. Topological Feature Extraction with Persistent Homology:** Using the GUDHI library, the distance matrices were used for TDA to compute persistent homology and generate persistent diagrams.

**5. Feature Representation through Persistence Images:** To better suit the format of Machine learning algorithms, persistence diagrams were converted into persistence images, that we performed using the GUDHI library in Python.

**6. Classification and Model Selection:** Binary Classification: The initial aim is to distinguish between PDs and Controls. By categorizing these two groups, we hope to pinpoint the inherent topological differences which might be indicative of the disease's presence.

Multiclass Classification: The subsequent goal broadens the scope to classify the persistence images into three distinct categories, representing different stages of the disease and controls: CC (Control Case), $PD_{W00}$ (Parkinson's Disease at the baseline), and $PD_{W36}$(Parkinson's Disease after 36 weeks). This finer discrimination helps in tracking the evolution of topological features over time and in relation to the disease's progression.

For the classification tasks, given the constraints of our dataset size, we predominantly used two algorithms: Logistic Regression and Support Vector Machines (SVMs). Both were selected for their efficiency and appropriateness for binary and multiclass classification tasks within the context of our data's dimensionality and characteristics.

**7. Model Evaluation :** In this phase, we place a strong emphasis on ensuring the robustness and reliability of our classification models. Model accuracy serves as a primary metric, indicating how well our models differentiate between the groups in our study. Additionally, to optimize the predictive power of our classifiers, we used parameter tuning. This iterative process allows us to refine model parameters, resulting in enhanced accuracy and performance.

### 4.1.2 Results

#### Identification of Topological Features and Their Persistence

Using Matplotlib library in python we visualized the connectivity/distances matrices generated for the 5 PDs (Figure 4.3) and 5 Controls (Figure 4.4)

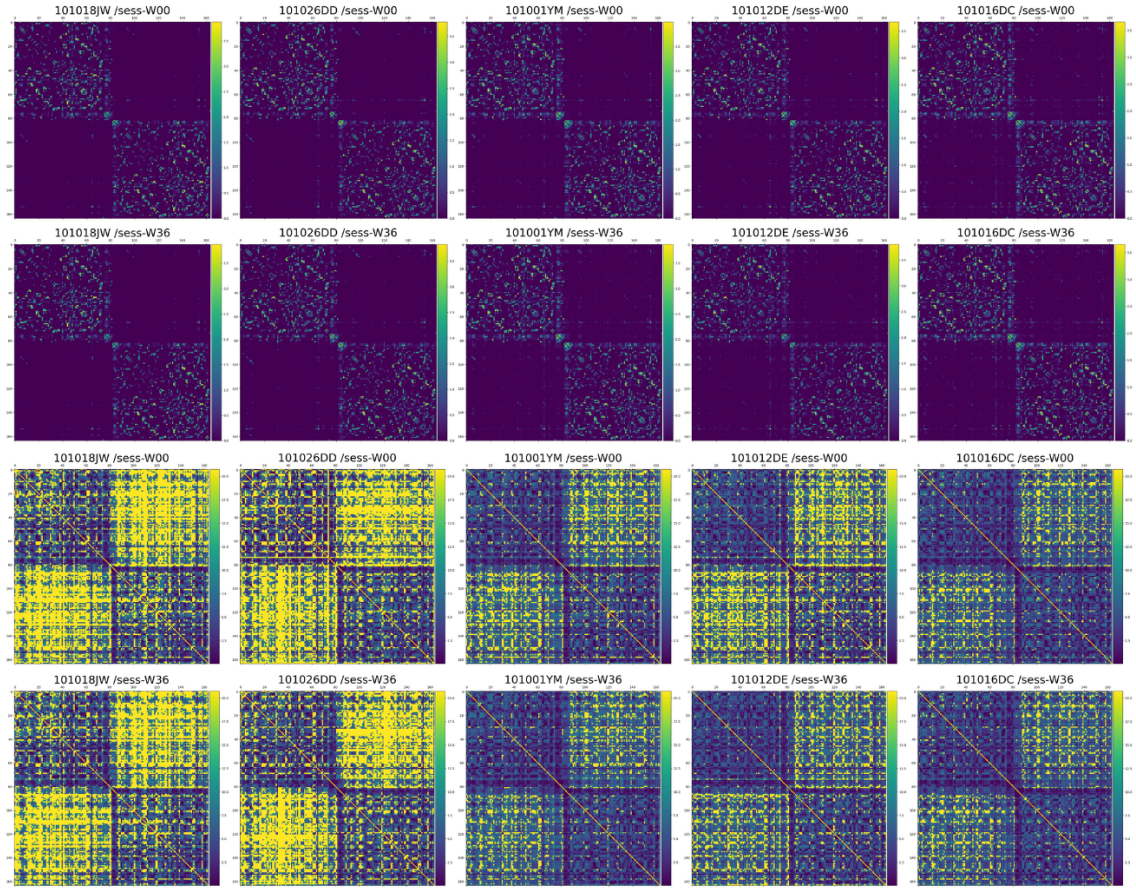



Figure 4.3: Representation of Connectivity and Distance Matrices for the 5 PDs from the FPII dataset
The first two rows illustrate the connectivity matrices for the PDs : the baseline session (W00) in the first row and the session after 9 months of disease progression (W36) in the second row. The third and fourth rows present the distance matrices derived from element-wise inversion of the respective connectivity matrices.

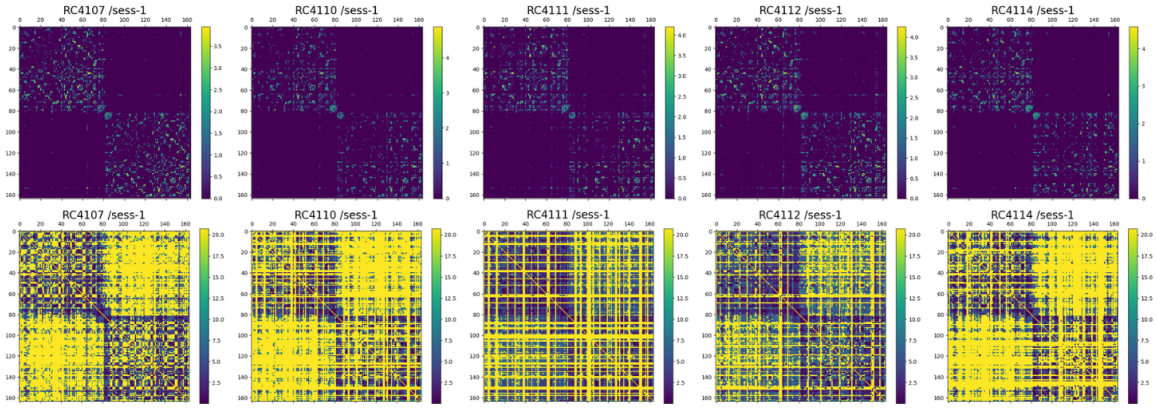



Figure 4.4: Representation of Connectivity and Distance Matrices for the 5 Controls from the Openneuro dataset
The figure illustrates the connectivity matrices for the Controls in the first row, and the distance matrices derived from element-wise inversion of the respective connectivity matrices in the second row

Using the GUDHI library, we created persistence diagrams from the distance matrices. These diagrams capture the birth and death of topological features across different scales. The results highlight a variety of persistent topological features. Notably, we observe distinctions in these features within the persistence diagrams across different homology degrees when comparing PDs to controls, as well as between the PD sessions themselves (Figure 4.5 and Figure 4.6).

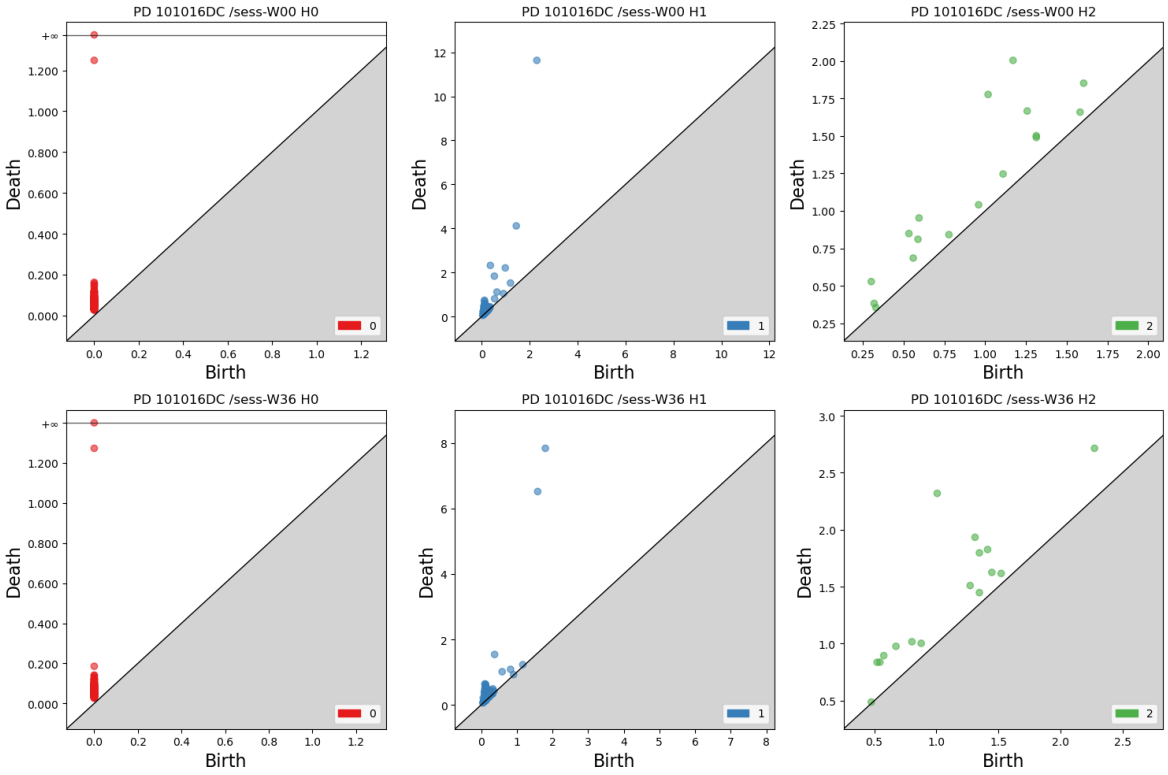



Figure 4.5: Persistence diagrams for the 3 homology degrees for one patient from FPII dataset for his two sessions, at the baseline and 36 weeks after

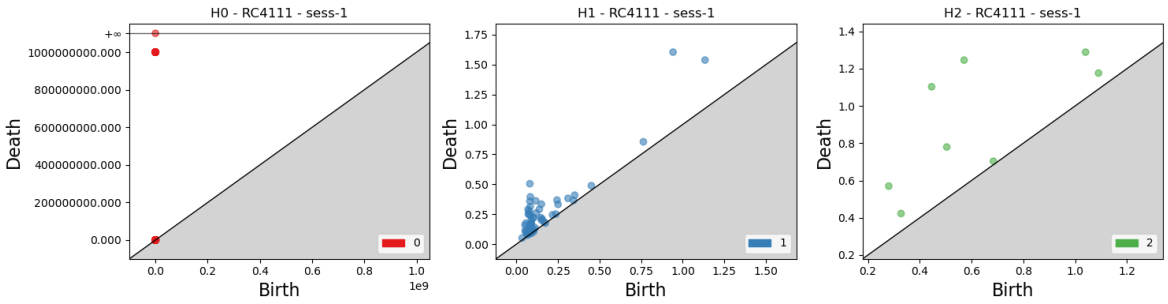



Figure 4.6: Persistence diagrams for the 3 homology degrees for a Control from the OpenNeuro dataset

Persistent diagram for homology degree 0 (left panel, red dots), homology degree 1 (middle panel, blue dots), homology degree 2 (right panel, green dots)

In order to simplify the comparison between persistence diagrams, we converted them into persistence images. This conversion enables encoding the topological information into a format suitable for our next step : Classification using Machine learning algorithms. The resulting persistence im-

ages for PDs and Controls are available in the subsequent figures. A detailed interpretation of these results, both qualitative and quantitative, is presented in the following section.

### Qualitative and Quantitative assessment of Persistent Homology Results

### Classification using Machine Learning algorithms

- Binary Classification PD vs Control : In this classification control vs. PD groups, we obtained varying classifier performance across different hypotheses. For both homology degrees 0 and 1, the Logistic Regression and SVM classifiers achieved an accuracy of 1.0, indicating perfect classification. However, for homogy of degree 2, both classifiers only reached an accuracy of 0.5, suggesting no clear differentiation between the groups. These results highlight the need for careful data analysis and model selection based on the specific hypothesis being tested (figure 4.7 and 4.8). Refer to Appendix for the code
- Binary Classification $PD_W00$ vs $PD_W36$ : In this classification (PDs) at first session (W00) vs PDs after 9 months of disease progression (W36) using Logistic Regression and Support Vector Machines (SVM), we found that both models had a perfect accuracy of 1.0 for homology degree 2. This means they successfully distinguished PDs from the baseline and progressed sessions. However, for homologies 0 and 1, the accuracy of both models was only 0.5, indicating chance-level results. The accuracy in $H_2$ highlights the models ability in spotting complex patterns due to disease-related changes. This suggests the classifiers could be valuable in telling apart PDs linked to the initial and progressed sessions, potentially enhancing our understanding of disease-related differences through quantitative analysis (figure 4.9).
- Classification into three different classes (CC, $PD_W00$, $PD_W36$) for each homology degree: In our analysis, we evaluated the accuracy of the SVM classifier in distinguishing the persistence images into three distinct classes: CC, $PD_W00$, and $PD_W36$, across different homology degrees in order to track the natural progression of the disease. Initially, for the homology degree H0, the SVM achieved an accuracy of 33.33% while for $H_1$, it was considerably higher at 66.67%. The classifier struggled to categorize images under the $H_2$ homology degree, showing an accuracy of 0.0%. However, after the process of parameter tuning, there was a notable improvement in the performance for the $H_2$ homology degree, elevating its accuracy to 33.33%. The accuracies for $H_0$ and $H_1$ remained unchanged post-tuning. This underscores the importance of parameter optimization, especially in instances where initial model performance is unsatisfactory.

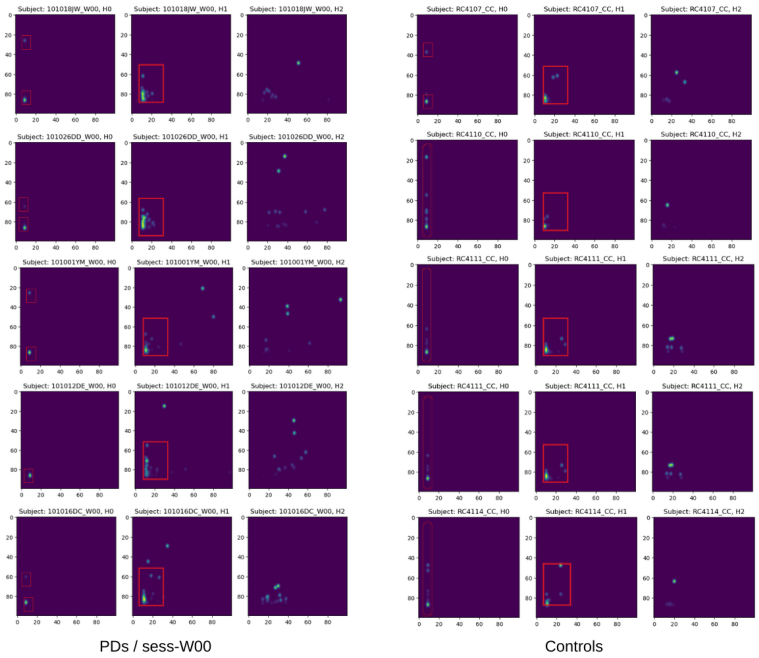



Figure 4.7: Persistence images PDs/sess-W00 Vs Controls
These persistence images of homology degree 0 and 1 help us understand the features in the data of PDs/sess-W00 compared to that of Controls. We have outlined a consistent pattern in each group with a red frame.

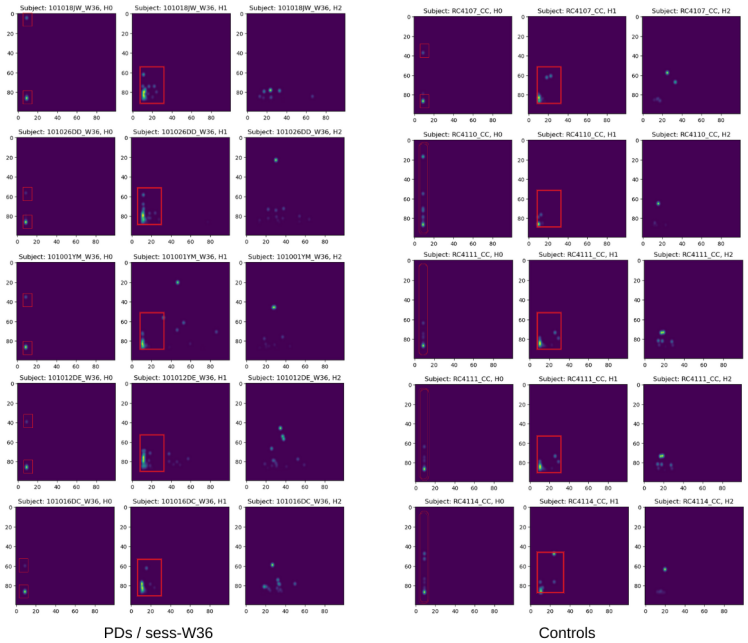



Figure 4.8: Persistence images PDs/sess-W36 Vs Controls
These persistence images of homology degree 0 and 1 help us understand the features in the data of PDs/sess-W36 compared to that of Controls. We have outlined a consistent pattern in each group with a red frame.

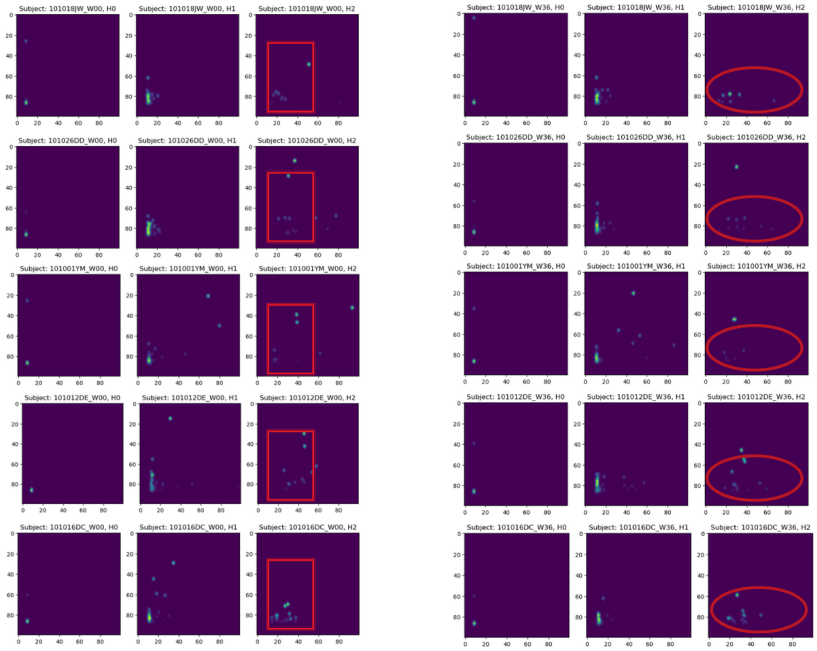



Figure 4.9: Persistence images PDs/sess-W36 Vs PDs/sess-W00
These persistence images of homology degree 2 help us understand the features in the data of PDs/sess-W00 compared to that of PDs/sess-W36. We have outlined a consistent pattern in each group with a red frame.

# Discussion

One of the prominent limitations encountered in this study is the computational complexity associated with fiber Orientation Distribution Functions. Processing and analyzing these distributions demand significant computational resources, which can restrict real-time or large-scale applications. To address this issue we can consider Code Optimization :

- Parallel Computing: Some loops can benefit from parallelization. Tools like "multiprocessing" in Python can distribute tasks across multiple CPUs, speeding up execution.
- Algorithmic Refinement: seeking faster linear programming solutions or using GPU-accelerated libraries such as "CuPy" might enhance performance.
- Collaborative and Cloud-based Computing also could be a good way to tackle this : Given the computational demands of topological data analysis, especially with fODFs, leveraging collaborative and cloud-based computational resources can help distribute the processing load, allowing for more complex analyses without the limitations of local computational capacity.

We can also consider optimizing the fODFs data given that the complexity we are dealing with arises from the nature of the problem itself (computationally expensive due to high volume of comparisons). There are methods or approximations that could potentially reduce the number of tetrahedrons we need to compare and reduce the resolution of the fODFs in a way that would still provide sufficient detail for our purposes. The Generalized Fractional Anisotropy helps identify the primary directions of the Fiber Orientation Distribution Function. By setting a threshold on GFA, we can selectively focus on key parts of the fODFs, thereby simplifying our analysis and reducing computational demand. This approach emphasizes vital fiber orientations and omits less significant ones, balancing efficiency with result quality.

Another limitation that became apparent was the size of the dataset used for the structural connectome approach. A smaller dataset may not capture the entire spectrum of variability and nuances inherent in broader cohorts. This might affect the generalizability and robustness of the results. For this approach, expanding the dataset is crucial. A larger, more varied dataset will ensure a more comprehensive and generalizable analysis. The scalability of the implemented method is a subject of further optimization work, to ensure the feasibility of this approach for larger datasets and facilitate its integration into broader studies on brain fiber networks in Parkinson's disease.

Another fundamental limitation faced during the study was the accuracy of the classification process. While the methods employed provided some level of distinction between different data sets or categories, they did not always produce consistent or highly reliable results. Factors that could contribute to these accuracy limitations include noise in the data, insufficient training data for the classifier, or the inherent complexity of the patterns we are trying to classify. Improved classification models or more sophisticated feature extraction methods may be required to enhance accuracy.

# Conclusion

Throughout this research, we have used topological data analysis on 3-tesla diffusion MRI images to uncover a new imaging biomarker in Parkinson's disease. Its application combined with Machine learning classification algorithms, revealed distinct connectivity patterns in both control and Parkinsonian patients demonstrating the potential of topological features to offer valuable insights into Parkinson's disease. Remarkably, these methodologies have enabled the differentiation of newly diagnosed subjects from those with nine months of disease progression, suggesting a promising imaging biomarker for the prognosis of Parkinson's disease.

# Bibliography


[1] Henry Adams et al. "Persistence Images: A Stable Vector Representation of Persistent Homology". In: *Journal of Machine Learning Research* 18.8 (2017), pp. 1–35. ISSN: 1533-7928. URL: http://jmlr.org/papers/v18/16-337.html (visited on 07/09/2023).

[2] Federica Agosta et al. "Mild cognitive impairment in Parkinson's disease is associated with a distributed pattern of brain white matter damage". eng. In: *Human Brain Mapping* 35.5 (May 2014), pp. 1921–1929. ISSN: 1097-0193. DOI: 10.1002/hbm.22302.

[3] Alberto Ascherio and Michael A. Schwarzschild. "The epidemiology of Parkinson's disease: risk factors and prevention". eng. In: *The Lancet. Neurology* 15.12 (Nov. 2016), pp. 1257–1272. ISSN: 1474-4465. DOI: 10.1016/S1474-4422(16)30230-7.

[4] Vinit Baliyan et al. "Diffusion weighted imaging: Technique and applications". In: *World Journal of Radiology* 8.9 (Sept. 2016), pp. 785–798. ISSN: 1949-8470. DOI: 10.4329/wjr.v8.i9.785. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5039674/ (visited on 08/14/2023).

[5] Paul Bendich et al. "Persistent Homology Analysis of Brain Artery Trees". In: *The annals of applied statistics* 10.1 (2016), pp. 198–218. ISSN: 1932-6157. DOI: 10.1214/15-AOAS886. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5026243/ (visited on 07/09/2023).

[6] Ed Bullmore and Olaf Sporns. "Complex brain networks: graph theoretical analysis of structural and functional systems". en. In: *Nature Reviews Neuroscience* 10.3 (Mar. 2009). Number: 3 Publisher: Nature Publishing Group, pp. 186–198. ISSN: 1471-0048. DOI: 10.1038/nrn2575. URL: https://www.nature.com/articles/nrn2575 (visited on 07/09/2023).

[7] Carla Federica Melia. *Topological Data Analysis and Persistent Homology*. en. URL: https://www.slideshare.net/CarlaFedericaMelia/topological-data-analysis-and-persistent-homology-126895845 (visited on 08/11/2023).

[8] Gunnar Carlsson. "Topology and Data". In: *Bulletin of The American Mathematical Society - BULL AMER MATH SOC* 46 (Apr. 2009), pp. 255–308. DOI: 10.1090/S0273-0979-09-01249-X.

[9] Frédéric Chazal and Bertrand Michel. *An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists*. arXiv:1710.04019 [cs, math, stat]. Feb. 2021. DOI: 10.48550/arXiv.1710.04019. URL: http://arxiv.org/abs/1710.04019 (visited on 07/09/2023).

[10] Frédéric Chazal et al. "Proximity of persistence modules and their diagrams". en. In: *Proceedings of the twenty-fifth annual symposium on Computational geometry*. Aarhus Denmark: ACM, June 2009, pp. 237–246. ISBN: 978-1-60558-501-7. DOI: 10.1145/1542362.1542407. URL: https://dl.acm.org/doi/10.1145/1542362.1542407 (visited on 07/09/2023).

[11] William Dauer and Serge Przedborski. "Parkinson's disease: mechanisms and models". eng. In: *Neuron* 39.6 (Sept. 2003), pp. 889–909. ISSN: 0896-6273. DOI: 10.1016/s0896-6273(03)00568-3.

[12] Edelsbrunner, Letscher, and Zomorodian. "Topological Persistence and Simplification". en. In: *Discrete & Computational Geometry* 28.4 (Nov. 2002), pp. 511–533. ISSN: 1432-0444. DOI: 10.1007/s00454-002-2885-2. URL: https://doi.org/10.1007/s00454-002-2885-2 (visited on 07/09/2023).

[13] Herbert Edelsbrunner and John Harer. "Departments of Computer Science and Mathematics Duke University". en. In: ().

[14] Karl J. Friston. "Functional and effective connectivity: a review". eng. In: *Brain Connectivity* 1.1 (2011), pp. 13–36. ISSN: 2158-0022. DOI: 10.1089/brain.2011.0008.

[15] Robert Ghrist. "Barcodes: The persistent topology of data". en. In: *Bulletin of the American Mathematical Society* 45.1 (2008), pp. 61–75. ISSN: 0273-0979, 1088-9485. DOI: 10.1090/S0273-0979-07-01191-3. URL: https://www.ams.org/bull/2008-45-01/S0273-0979-07-01191-3/ (visited on 07/09/2023).

[16] Julie M. Hall et al. "Diffusion alterations associated with Parkinson's disease symptomatology: A review of the literature". eng. In: *Parkinsonism & Related Disorders* 33 (Dec. 2016), pp. 12–26. ISSN: 1873-5126. DOI: 10.1016/j.parkreldis.2016.09.026.

[17] Phoebe Imms et al. "Navigating the link between processing speed and network communication in the human brain". In: *Brain Structure and Function* 226 (May 2021). DOI: 10.1007/s00429-021-02241-8.

[18] J. Jankovic. "Parkinson's disease: clinical features and diagnosis". eng. In: *Journal of Neurology, Neurosurgery, and Psychiatry* 79.4 (Apr. 2008), pp. 368–376. ISSN: 1468-330X. DOI: 10.1136/jnnp.2007.131045.

[19] Ben Jeurissen et al. "Investigating the prevalence of complex fiber configurations in white matter tissue with diffusion magnetic resonance imaging". eng. In: *Human Brain Mapping* 34.11 (Nov. 2013), pp. 2747–2766. ISSN: 1097-0193. DOI: 10.1002/hbm.22099.

[20] Derek K. Jones, Thomas R. Knösche, and Robert Turner. "White matter integrity, fiber count, and other fallacies: the do's and don'ts of diffusion MRI". eng. In: *NeuroImage* 73 (June 2013), pp. 239–254. ISSN: 1095-9572. DOI: 10.1016/j.neuroimage.2012.06.081.

[21] Enrico Kaden et al. "Multi-compartment microscopic diffusion imaging". eng. In: *NeuroImage* 139 (Oct. 2016), pp. 346–359. ISSN: 1095-9572. DOI: 10.1016/j.neuroimage.2016.06.002.

[22] Hyekyoung Lee et al. "Discriminative persistent homology of brain networks: 2011 8th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, ISBI'11". In: *2011 8th IEEE International Symposium on Biomedical Imaging*. Proceedings - International Symposium on Biomedical Imaging (2011), pp. 841–844. ISSN: 9781424441280. DOI: 10.1109/ISBI.2011.5872535. URL: http://www.scopus.com/inward/record.url?scp=80055057822&partnerID=8YFLogxK (visited on 07/09/2023).

[23] Sylvain L. Merlet and Rachid Deriche. "Continuous diffusion signal, EAP and ODF estimation via Compressive Sensing in diffusion MRI". eng. In: *Medical Image Analysis* 17.5 (July 2013), pp. 556–572. ISSN: 1361-8423. DOI: 10.1016/j.media.2013.02.010.

[24] J. A. Obeso et al. “Past, present, and future of Parkinson’s disease: A special essay on the 200th Anniversary of the Shaking Palsy”. eng. In: *Movement Disorders: Official Journal of the Movement Disorder Society* 32.9 (Sept. 2017), pp. 1264–1310. ISSN: 1531-8257. DOI: 10.1002/mds.27115.

[25] Chong Jin Ong and Elmer G Gilbert. “lEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 12, NO. 6, DECEMBER 1996”. en. In: ().

[26] Werner Poewe et al. “Parkinson disease”. en. In: *Nature Reviews Disease Primers* 3.1 (Mar. 2017). Number: 1 Publisher: Nature Publishing Group, pp. 1–21. ISSN: 2056-676X. DOI: 10.1038/nrdp.2017.13. URL: https://www.nature.com/articles/nrdp201713 (visited on 07/09/2023).

[27] Russell A. Poldrack et al. “Long-term neural and physiological phenotyping of a single human”. en. In: *Nature Communications* 6.1 (Dec. 2015). Number: 1 Publisher: Nature Publishing Group, p. 8885. ISSN: 2041-1723. DOI: 10.1038/ncomms9885. URL: https://www.nature.com/articles/ncomms9885 (visited on 07/09/2023).

[28] Tamara Pringsheim et al. “The prevalence of Parkinson’s disease: a systematic review and meta-analysis”. eng. In: *Movement Disorders: Official Journal of the Movement Disorder Society* 29.13 (Nov. 2014), pp. 1583–1590. ISSN: 1531-8257. DOI: 10.1002/mds.25945.

[29] David Raffelt et al. “Apparent Fibre Density: a novel measure for the analysis of diffusion-weighted magnetic resonance images”. eng. In: *NeuroImage* 59.4 (Feb. 2012), pp. 3976–3994. ISSN: 1095-9572. DOI: 10.1016/j.neuroimage.2011.10.045.

[30] Marco Reisert et al. “Global fiber reconstruction becomes practical”. eng. In: *NeuroImage* 54.2 (Jan. 2011), pp. 955–962. ISSN: 1095-9572. DOI: 10.1016/j.neuroimage.2010.09.016.

[31] Manish Saggar et al. “Towards a new approach to reveal dynamical organization of the brain using topological data analysis”. en. In: *Nature Communications* 9.1 (Apr. 2018). Number: 1 Publisher: Nature Publishing Group, p. 1399. ISSN: 2041-1723. DOI: 10.1038/s41467-018-03664-4. URL: https://www.nature.com/articles/s41467-018-03664-4 (visited on 07/09/2023).

[32] Anthony H. V. Schapira, K. Ray Chaudhuri, and Peter Jenner. “Non-motor features of Parkinson disease”. eng. In: *Nature Reviews. Neuroscience* 18.7 (July 2017), pp. 435–450. ISSN: 1471-0048. DOI: 10.1038/nrn.2017.62.

[33] Kiran Seunarine and Daniel Alexander. “Multiple Fibers. Beyond the Diffusion Tensor.” In: *Diffusion MRI: From Quantitative Measurement to In Vivo Neuroanatomy* (Nov. 2013), pp. 56–74. ISSN: 9780123964601. DOI: 10.1016/B978-0-12-396460-1.00006-8.

[34] Sule Tinaz, Maureen G. Courtney, and Chantal E. Stern. “Focal cortical and subcortical atrophy in early Parkinson’s disease”. eng. In: *Movement Disorders: Official Journal of the Movement Disorder Society* 26.3 (Feb. 2011), pp. 436–441. ISSN: 1531-8257. DOI: 10.1002/mds.23453.

[35] Jacques-Donald Tournier, Susumu Mori, and Alexander Leemans. “Diffusion tensor imaging and beyond”. eng. In: *Magnetic Resonance in Medicine* 65.6 (June 2011), pp. 1532–1556. ISSN: 1522-2594. DOI: 10.1002/mrm.22924.

[36] Raoul Wadhwa et al. “TDAstats: R pipeline for computing persistent homology in topological data analysis”. In: *Journal of Open Source Software* 3 (Aug. 2018), p. 860. DOI: 10.21105/joss.00860.

[37] *What is tractography?* en. URL: https://www.o8t.com/blog/tractography (visited on 08/11/2023).

[38] Tao Wu et al. “Changes of functional connectivity of the motor network in the resting state in Parkinson’s disease”. eng. In: *Neuroscience Letters* 460.1 (Aug. 2009), pp. 6–10. ISSN: 1872-7972. DOI: 10.1016/j.neulet.2009.05.046.

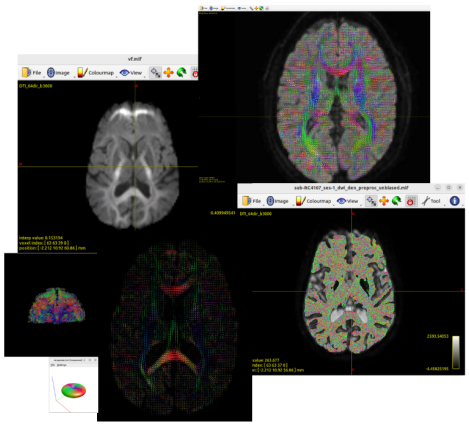
[39] Jiuquan Zhang et al. “Characterizing iron deposition in Parkinson’s disease using susceptibility-weighted imaging: an in vivo MR study”. eng. In: *Brain Research* 1330 (May 2010), pp. 124–130. ISSN: 1872-6240. DOI: 10.1016/j.brainres.2010.03.036.

[40] Hongru Zhu et al. “Altered Topological Properties of Brain Networks in Social Anxiety Disorder: A Resting-state Functional MRI Study”. en. In: *Scientific Reports* 7.1 (Mar. 2017). Number: 1 Publisher: Nature Publishing Group, p. 43089. ISSN: 2045-2322. DOI: 10.1038/srep43089. URL: https://www.nature.com/articles/srep43089 (visited on 08/15/2023).

# Appendix

## I Additional Figures and Visualizations

Preprocessing



Postprocessing

Preprocessing and Postprocessing of diffusion MRI data using Neuroimaging software Mrtrix, FSL, ANTS and Free surfer

## II Code and Algorithms Used

# Reconstruction based on a robust and unbiased spherical deconvolution model (RUMBA-SD)

In [ ]:
```
#%pip install bids
```

In [10]:
```
#importing necessary libraries
import numpy as np
from dipy.io.image import load_nifti, save_nifti
from dipy.io.gradients import read_bvals_bvecs
from dipy.core.gradients import gradient_table
from bids.layout import BIDSLayout
import dipy.reconst.dti as dti
from dipy.data import get_fnames
from dipy.io.image import load_nifti_data, load_nifti, save_nifti
from dipy.data import get_fnames, get_sphere
from dipy.core.sphere import HemiSphere
import nibabel as nib
from dipy.reconst.csdeconv import auto_response_ssst
from dipy.reconst.rumba import RumbaSDModel
```

In [20]:
```
gradient_layout = BIDSLayout("/media/imane/DATA/openneuro/ds001907/sub-RC4201/ses-1/", validate=False)
#accessing the openneuro dataset

subj = 'RC4201'

dwi_fname = gradient_layout.get(subject=subj, suffix='dwi', extension='.nii.gz', return_type='file')[0]
bvec_fname = gradient_layout.get( extension='.bvec', return_type='file')[0]
bval_fname = gradient_layout.get( extension='.bval', return_type='file')[0]

dwi_img = nib.load(dwi_fname)
affine = dwi_img.affine

bvals, bvecs = read_bvals_bvecs(bval_fname, bvec_fname)
gtab = gradient_table(bvals, bvecs)

sphere = get_sphere('repulsion724')
```

In [21]:
```
from dipy.segment.mask import median_otsu
import dipy.reconst.dti as dti
from dipy.segment.mask import median_otsu

dwi_data = dwi_img.get_fdata() #diffusion data

maskdata, mask = median_otsu(dwi_data, vol_idx=range(10, 50), median_radius=3,
                             numpass=1, autocrop=True, dilate=2)
print('maskdata.shape (%d, %d, %d, %d)' % maskdata.shape)
print('data.shape (%d, %d, %d, %d)' % dwi_data.shape)
```

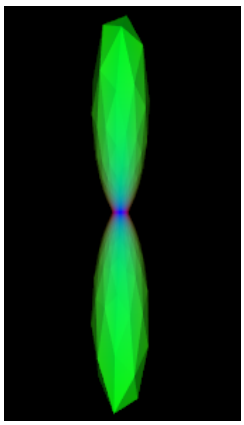
```
maskdata.shape (81, 108, 69, 129)
data.shape (128, 128, 72, 129)
```

In [22]:
```
from dipy.reconst.rumba import RumbaSDModel

rumba = RumbaSDModel(gtab)
print(f"wm_response: {rumba.wm_response}, " +
      f"csf_response: {rumba.csf_response}, " +
      f"gm_response: {rumba.gm_response}")
```

```
wm_response: [0.0017 0.0002 0.0002], csf_response: 0.003, gm_response: 0.0008
```

Fiber response function estimation :

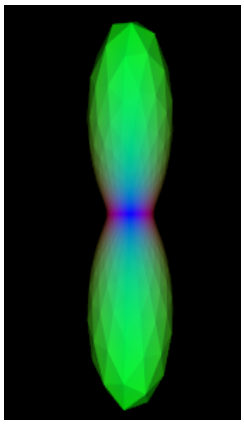
In [23]:
```
#Visualization of the response function
from dipy.sims.voxel import single_tensor_odf
from fury import window, actor

# Enables/disables interactive visualization
interactive = False

scene = window.Scene()

evals = rumba.wm_response
evecs = np.array([[0, 1, 0], [0, 0, 1], [1, 0, 0]]).T

response_odf = single_tensor_odf(sphere.vertices, evals, evecs)
# Transform our data from 1D to 4D
response_odf = response_odf[None, None, None, :]
response_actor = actor.odf_slicer(response_odf, sphere=sphere
                                  )

scene.add(response_actor)
#print('Saving illustration as default_response.png')
#window.record(scene, out_path='default_response.png', size=(200, 200))


window.show(scene)
```



In [11]:
```
scene.rm(response_actor)
```

In [24]:
```
#Another method : Estimation of the fiber response function from the local brain region.
from dipy.sims.voxel import single_tensor_odf
from dipy.reconst.csdeconv import auto_response_ssst
from fury import window, actor

response, _ = auto_response_ssst(gtab, dwi_data, roi_radii=10, fa_thr=0.7)
print(response)
evals = response[0]
evecs = np.array([[0, 1, 0], [0, 0, 1], [1, 0, 0]]).T

response_odf = single_tensor_odf(sphere.vertices, evals, evecs)
# transform our data from 1D to 4D
response_odf = response_odf[None, None, None, :]
response_actor = actor.odf_slicer(response_odf, sphere=sphere

                                  )
scene = window.Scene()
scene.add(response_actor)
#print('Saving illustration as estimated_response.png')
#window.record(scene, out_path='estimated_response.png', size=(200, 200))
window.show(scene)
```

```
(array([0.00165202, 0.00039875, 0.00039875]), 69502.95412529839)
```



In [25]:
```
scene.rm(response_actor)
```

Reconstruction of the fODF (Fiber Orientation Distribution Function)

In [26]:
```
response, _ = auto_response_ssst(gtab, dwi_data, roi_radii=10, fa_thr=0.7)
print(response)
```

```
(array([0.00165202, 0.00039875, 0.00039875]), 69502.95412529839)
```

In [28]:
```
#global fit
rumba = RumbaSDModel(gtab, wm_response=response[0], gm_response=None,
                     voxelwise=False, use_tv=False, sphere=sphere)
data_tv = dwi_data[10:50, 40:85, 20:40] #small part of the data

#voxel = dwi_data[29:30,59:60,38:39]
```

In [29]:
```
rumba_fit = rumba.fit(data_tv)

odf = rumba_fit.odf() #odf tensor
```

In [27]:
```
odf.shape
```

Out[27]:
```
(40, 45, 20, 724)
```

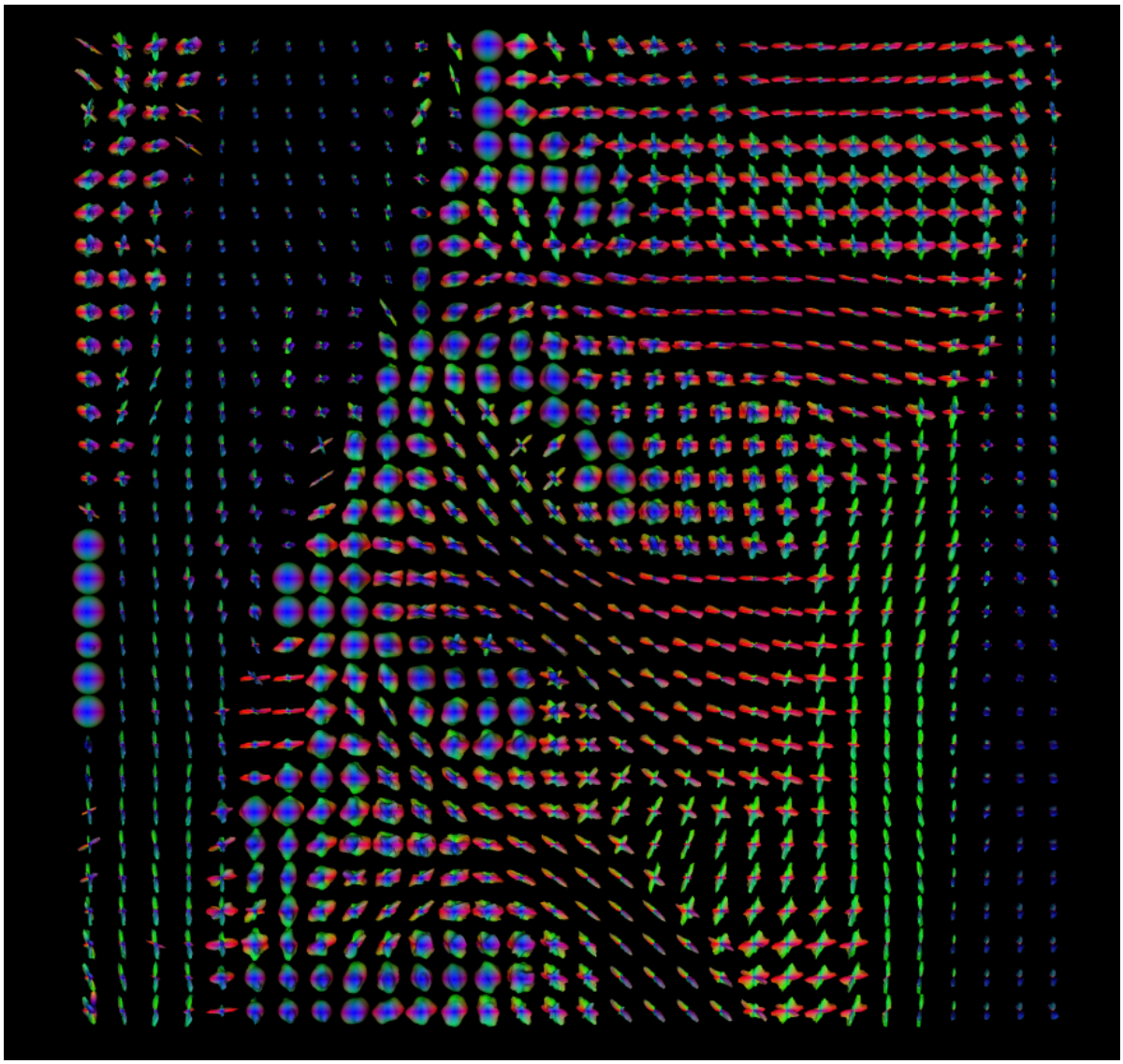
In [31]:
```
#To visualize the fODFs, i combined the fODF with the isotropic components.
combined = rumba_fit.combined_odf_iso
```

In [32]:
```
combined.shape
```

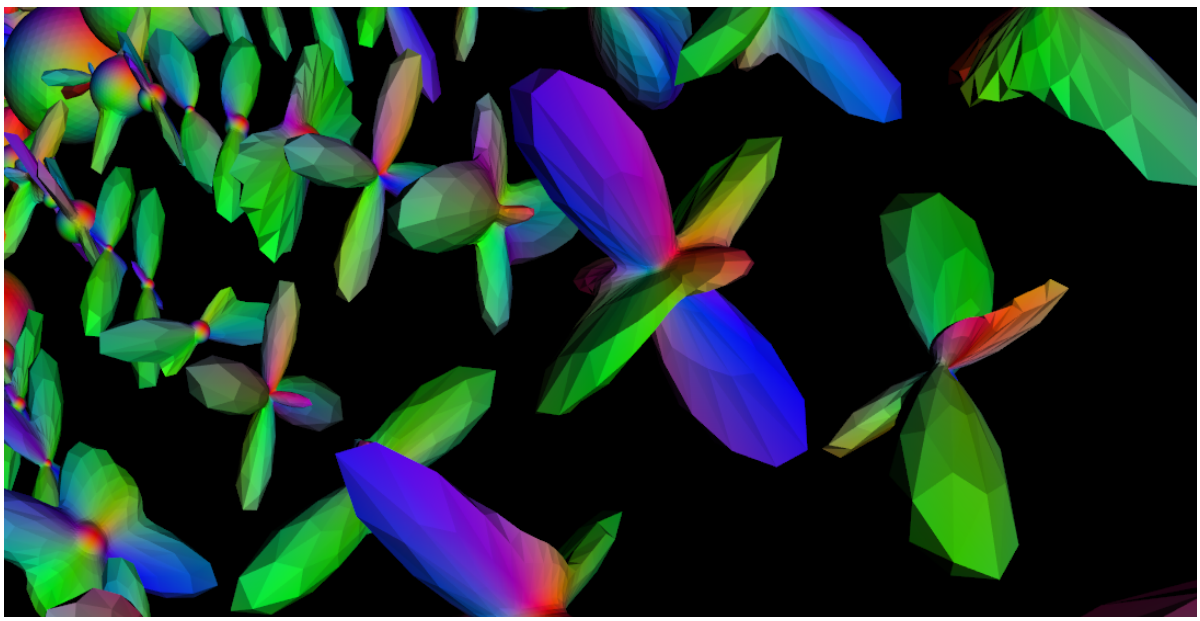
Out[32]:
```
(40, 45, 20, 724)
```

In [37]:
```
from fury import window, actor
combined = rumba_fit.combined_odf_iso
fodf_spheres = actor.odf_slicer(combined, sphere=sphere, norm=True,
                                scale=0.5, colormap=None)
scene = window.Scene()
scene.add(fodf_spheres)
window.show(scene)
```

Visualization of the fODF map: RUMBA-SD



Zoom ++



In [38]:
```
fodf_spheres.indices
```

Out[38]:
```
(array([ 0,  0,  0, ..., 39, 39, 39]),
 array([ 0,  0,  0, ..., 44, 44, 44]),
 array([ 0,  1,  2, ..., 17, 18, 19]))
```

In [39]:
```
fodf_spheres.faces
```

Out[39]:
```
array([[  2, 363,   0],
       [  0, 363, 365],
       [366, 364,   1],
       ...,
       [723, 706, 689],
       [723, 719, 685],
       [702, 719, 723]], dtype=uint16)
```

In [40]:
```
fodf_spheres.vertices
```

Out[40]:
```
array([[-0.0914682 ,  0.99569081,  0.01527661],
       [-0.03198756, -0.99848131,  0.04485394],
       [ 0.00657908,  0.99435208,  0.10592761],
       ...,
       [-0.73806889,  0.044762  , -0.67323894],
       [ 0.99918788,  0.03855745, -0.01170055],
       [ 0.71569262, -0.0094363 , -0.69835165]])
```

# Growth distance implementation

In [12]:
```
#Tetrahedron subdivision
import numpy as np

odf_cartesian_faces_dict = {}  # Dictionary to store tetrahedrons for each ODF center

for i in range(odf.shape[0]):
    for j in range(odf.shape[1]):
        for k in range(odf.shape[2]):
            voxel_center = np.array([i, j, k])  # Center of the current voxel

            for n in range(odf.shape[3]):
                voxel_faces = []  # List to store Cartesian coordinates for each face in the voxel
                face_indices = sphere.faces[n]  # Face indices corresponding to the current direction

                face_vertices = [voxel_center]  # Add the center of the voxel as the first vertex

                for idx in face_indices:
                    cartesian_coords = voxel_center + odf[i, j, k, n] * sphere.vertices[idx]

                    face_vertices.append(cartesian_coords)

                voxel_faces.append(face_vertices)

                # Add tetrahedrons to the corresponding ODF center in the dictionary
                odf_center_key = tuple(voxel_center)  # Use the voxel_center as the key
                if odf_center_key not in odf_cartesian_faces_dict:
                    odf_cartesian_faces_dict[odf_center_key] = []
                odf_cartesian_faces_dict[odf_center_key].extend(voxel_faces)
# odf_cartesian_faces_dict contains tetrahedrons for each ODF center
```

In [ ]:
```
#the tetrahedrons of the first ODF with center [0, 0, 0]
odf_cartesian_faces_dict[(0, 0, 0)]
```

In [39]:
```
#Implementing the growth distance algorithm
Aeq = np.column_stack(A)
ones_row = np.ones((1, Aeq.shape[1]))
Aeq = np.concatenate((Aeq, ones_row), axis=0)
Aeq
```

Out[39]:
```
array([[ 0.00000000e+00, -4.45509482e-05,  1.55800182e-05,
         3.20443748e-06],
       [ 0.00000000e+00,  4.84966050e-04,  4.86325202e-04,
         4.84314000e-04],
       [ 0.00000000e+00,  7.44070161e-06, -2.18467780e-05,
         5.15936194e-05],
       [ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00,
         1.00000000e+00]])
```

In [41]:
```
Beq = np.column_stack(B)
minus_ones_row = np.ones((1, Beq.shape[1])) * -1
Beq = np.concatenate((Beq, minus_ones_row), axis=0)
Beq
```

Out[41]:
```
array([[ 0.00000000e+00, -2.38126082e-04, -2.98936667e-04,
        -3.11225567e-04],
       [ 0.00000000e+00, -6.00621932e-04, -5.74394913e-04,
        -5.62277423e-04],
       [ 1.00000000e+00,  1.00004530e+00,  9.99985505e-01,
         1.00008053e+00],
       [-1.00000000e+00, -1.00000000e+00, -1.00000000e+00,
        -1.00000000e+00]])
```

In [43]:
```
combined_matrix = np.hstack((Aeq, Beq))
combined_matrix
```

Out[43]:
```
array([[ 0.00000000e+00, -4.45509482e-05,  1.55800182e-05,
         3.20443748e-06,  0.00000000e+00, -2.38126082e-04,
        -2.98936667e-04, -3.11225567e-04],
       [ 0.00000000e+00,  4.84966050e-04,  4.86325202e-04,
         4.84314000e-04,  0.00000000e+00, -6.00621932e-04,
        -5.74394913e-04, -5.62277423e-04],
       [ 0.00000000e+00,  7.44070161e-06, -2.18467780e-05,
         5.15936194e-05,  1.00000000e+00,  1.00004530e+00,
         9.99985505e-01,  1.00008053e+00],
       [ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00,
         1.00000000e+00, -1.00000000e+00, -1.00000000e+00,
        -1.00000000e+00, -1.00000000e+00]])
```

In [50]:
```
pB_minus_pA = B[0] - A[0]
pB_minus_pA
```

Out[50]:
```
array([0, 0, 1])
```

In [51]:
```
b = np.concatenate((pB_minus_pA, [0]))
b
```

Out[51]:
```
array([0, 0, 1, 0])
```

In [31]:
```
#Exemple of the growth distance between 2 tetrahedrons of 2 different fodfs
import numpy as np
from scipy.spatial import ConvexHull
from scipy.optimize import linprog

def calculate_convex_hull(vertices):
    hull = ConvexHull(vertices)
    return np.array(vertices)[hull.vertices]

def solve_lp_problem(bar_A, bar_B):

    num_A = len(bar_A)
    num_B = len(bar_B)

    # Construct the constraint matrix combined_matrix
    Aeq = np.column_stack(bar_A)
    ones_row = np.ones((1, Aeq.shape[1]))
    Aeq = np.concatenate((Aeq, ones_row), axis=0)

    Beq = np.column_stack(bar_B)
    minus_ones_row = np.ones((1, Beq.shape[1])) * -1
    Beq = np.concatenate((Beq, minus_ones_row), axis=0)

    combined_matrix = np.hstack((Aeq, Beq))
    # Coefficients of the objective function
    c = np.ones(num_A+num_B)

    pB_minus_pA = bar_B[0] - bar_A[0]

    # Construct the right-hand side vector b_eq
    b_eq = np.concatenate((pB_minus_pA, [0]))

    bounds = [(0, None)] * (num_A + num_B)  # Bounds for alpha_A and alpha_B

    res = linprog(c, A_eq=combined_matrix, b_eq=b_eq, bounds=bounds)
    sigma_star = res.fun
    alpha_A = res.x[:num_A]
    alpha_B = res.x[num_A:]
    return sigma_star, alpha_A, alpha_B
```

In [33]:
```
# Calculate convex hulls of A and B
A = odf_cartesian_faces[0]   #tetrahedron1
B = odf_cartesian_faces[800]   #tetrahedron2
bar_A = calculate_convex_hull(A)
bar_B = calculate_convex_hull(B)


sigma_star, alpha_A, alpha_B = solve_lp_problem(bar_A, bar_B)

print("sigma_star:", sigma_star)
print("alpha_A:", alpha_A)
print("alpha_B:", alpha_B)
```

```
sigma_star: 2.0000000035223438
alpha_A: [1.00000000e+00 8.50020595e-10 5.04347761e-10 5.87846481e-10]
alpha_B: [1.00000000e+00 1.33106094e-09 1.31830763e-10 4.13307623e-10]
```

The growth distance is given by the factor sigma star = 2.0000000035223438

In [17]:
```
#Exemple of the growth distance between 2 different fodfs
# Retrieve the tetrahedrons for ODF centers (0, 0, 0) and (0, 0, 1)
tetrahedrons_A = odf_cartesian_faces_dict[(0, 0, 0)]
tetrahedrons_B = odf_cartesian_faces_dict[(0, 0, 1)]
```

In [24]:
```python
import numpy as np
from scipy.spatial import ConvexHull
from scipy.optimize import linprog
import time

def calculate_convex_hull(vertices):
    hull = ConvexHull(vertices)
    return np.array(vertices)[hull.vertices]

def solve_lp_problem(bar_A, bar_B):
    num_A = len(bar_A)
    num_B = len(bar_B)

    # Construct the constraint matrix A_eq
    Aeq = np.column_stack(bar_A)
    ones_row = np.ones((1, Aeq.shape[1]))
    Aeq = np.concatenate((Aeq, ones_row), axis=0)

    Beq = np.column_stack(bar_B)
    minus_ones_row = np.ones((1, Beq.shape[1])) * -1
    Beq = np.concatenate((Beq, minus_ones_row), axis=0)

    combined_matrix = np.hstack((Aeq, Beq))
    # Coefficients of the objective function
    c = np.ones(num_A + num_B)

    pB_minus_pA = bar_B[0] - bar_A[0]

    # Construct the right-hand side vector b_eq
    b_eq = np.concatenate((pB_minus_pA, [0]))

    bounds = [(0, None)] * (num_A + num_B)  # Bounds for alpha_A and alpha_B

    res = linprog(c, A_eq=combined_matrix, b_eq=b_eq, bounds=bounds)
    sigma_star = res.fun
    alpha_A = res.x[:num_A]
    alpha_B = res.x[num_A:]
    return sigma_star
```

In [ ]:
```python
def calculate_optimized_sigma_star(tetrahedrons_A, tetrahedrons_B):
    optimized_sigma_star = float('inf')  # Initialize with a large value

    # Loop over the tetrahedrons of ODF centers (0, 0, 0) and (0, 0, 1)
    for tetrahedron_A in tetrahedrons_A:
        for tetrahedron_B in tetrahedrons_B:
            # Calculate the convex hulls for each pair of tetrahedrons
            convex_hull_A = calculate_convex_hull(tetrahedron_A)
            convex_hull_B = calculate_convex_hull(tetrahedron_B)

            # Solve the linear programming problem between the convex hulls
            sigma_star = solve_lp_problem(convex_hull_A, convex_hull_B)

            # Update the optimized_sigma_star if a smaller value is found
            optimized_sigma_star = min(optimized_sigma_star, sigma_star)

    return optimized_sigma_star
```

In [ ]:
```python
def run_optimization(tetrahedrons_A, tetrahedrons_B):
    start_time = time.time()
    optimized_sigma_star = calculate_optimized_sigma_star(tetrahedrons_A, tetrahedrons_B)
    end_time = time.time()
    runtime_seconds = end_time - start_time
    runtime_minutes = runtime_seconds / 60
    # Print runtime in minutes
    print("Runtime of the code is:", runtime_minutes, "minutes.")

    # Print the optimized sigma_star
    print("Optimized sigma_star:", optimized_sigma_star)
```

In [ ]:
```python
run_optimization(tetrahedrons_A, tetrahedrons_B)
```

Runtime of the code is: 9.171941713492076 minutes. Optimized sigma_star: 1.9809936553587417

The optimized sigma_star value of 1.9809936553587417 indicates the minimum scaling factor needed for the two FODFs to intersect, based on the linear programming solution. This value plays a crucial role in the computation of the growth distance between two FODFs, which is a key step in the topological data analysis process.

As for the runtime, 9 minutes per pair of FODFs may be time-consuming, especially since we are dealing with a large number of FODFs.

# Loading and Visualizing structural connectome Openneuro/FPII

In [1]:
```python
#Loading FairparkII matrices
import os
import numpy as np

def load_pd_matrices(directory, subjects):
    """
    Load the matrices from "parcels_coreg.csv" for specific subjects in the specified directory.

    Args:
    directory (str): The directory where your datasets are located.
    subjects (list): The list of subjects to load matrices for.
    """
    # Create an empty dictionary to store the matrices
    matrices = {}

    # Iterate over the specified subjects
    for subject_id in subjects:
        # Construct the path to the subject's directory
        subject_directory = os.path.join(directory, subject_id)

        # Check if the directory exists for the current subject
        if os.path.isdir(subject_directory):
            # Create a dictionary to hold the matrices for each session
            session_matrices = {}

            # Define the session directories
            sessions = ['W00', 'W36']

            # Iterate over the session directories within the subject's directory
            for session in sessions:
                # Construct the path to the session's directory
                session_directory = os.path.join(subject_directory, session)

                # Construct the path to the .csv file
                csv_file = os.path.join(session_directory, 'parcels_coreg.csv')

                # Check if the .csv file exists
                if os.path.isfile(csv_file):
                    # Load the matrix from the .csv file
                    matrix = np.loadtxt(open(csv_file, "rb"), delimiter=",", skiprows=0)

                    # Add the matrix to the dictionary using the session as the key
                    session_matrices[session] = matrix

            # Add the session matrices to the main dictionary using the subject ID as the key
            matrices[subject_id] = session_matrices

    return matrices
```

In [2]:
```python
# Loading the Pd's matrices
pd_list = ["101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
pd_matrices = load_pd_matrices("/media/imane/DATA/FPII/Fairparkpreprocessed/Lille/", pd_list)
pd_matrices['101018JW']
```

Out[2]:
```
{'W00': array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 7.35907096e-04],
        ...,
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 6.12915969e+00, 3.11312096e-03],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         6.12915969e+00, 0.00000000e+00, 1.39777430e-04],
        [0.00000000e+00, 0.00000000e+00, 7.35907096e-04, ...,
         3.11312096e-03, 1.39777430e-04, 0.00000000e+00]]),
 'W36': array([[0.00000000e+00, 0.00000000e+00, 1.28486910e-04, ...,
         5.86121818e-05, 0.00000000e+00, 2.14683935e-05],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [1.28486910e-04, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 8.63461999e-04],
        ...,
        [5.86121818e-05, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 6.31223792e+00, 2.60735966e-03],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         6.31223792e+00, 0.00000000e+00, 1.65842975e-04],
        [2.14683935e-05, 0.00000000e+00, 8.63461999e-04, ...,
         2.60735966e-03, 1.65842975e-04, 0.00000000e+00]])}
```

In [3]:
```python
import os
import numpy as np

#loading openneuro matrices
def load_control_matrices(bids_directory, subjects, sessions):
    """
    Load the subject matrices from "parcels_coreg.csv" in the 'dwi' folder of each specified subject's session 

    Args:
    bids_directory (str): The directory where your BIDS datasets are located.
    subjects (list): The list of subjects.
    sessions (list): The list of sessions.
    """
    matrices = {}

    # Iterate over the specified subjects and sessions
    for subject in subjects:
        for session in sessions:
            subject_path = os.path.join(bids_directory, "sub-" + subject, "ses-" + session, "dwi")

            # Check if the 'dwi' directory exists in the current subject directory
            if os.path.isdir(subject_path):

                # File name of the subject matrix
                file_name = "parcels_coreg.csv"

                file_path = os.path.join(subject_path, file_name)

                # Check if the 'parcels_coreg.csv' file exists
                if os.path.exists(file_path):
                    # Load the matrix from the .csv file
                    matrix = np.loadtxt(open(file_path, "rb"), delimiter=",", skiprows=0)

                    # Add the matrix to the dictionary using the subject and session as the key
                    matrices[(subject, session)] = matrix
                else:
                    print(f"File '{file_name}' doesn't exist in the directory {subject_path}")
            else:
                print(f"'dwi' directory doesn't exist in the directory {subject_path}")

    return matrices
```

In [4]:
```python
#Loading the control's matrices
control_list = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
sessions = ["1"]
control_matrices = load_control_matrices('/media/imane/DATA/openneuro/ds001907/', control_list, sessions)
control_matrices[('RC4107','1')] #matrix of subject RC4107 session 1,
```

Out[4]:
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 1.80605806e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 1.80605806e-04, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        0.00000000e+00, 9.47638568e+00, 2.13186839e-04],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        9.47638568e+00, 0.00000000e+00, 7.94117874e-05],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        2.13186839e-04, 7.94117874e-05, 0.00000000e+00]])
```

In [5]:
```python
#Visualizing PD matrices
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.axes_grid1 import make_axes_locatable

def visualize_pd_matrices(matrices, subjects_to_plot, sessions_to_plot):
    # Define the colormap range
    vmin = None
    vmax = None

    # Define the number of columns and rows for the subplots
    n_cols = len(subjects_to_plot)
    n_rows = len(sessions_to_plot)

    # Create a new figure with specified size
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(10 * n_cols, 10 * n_rows))  # Adjust size as needed

    for i, session in enumerate(sessions_to_plot):
        for j, subject_id in enumerate(subjects_to_plot):
            # Retrieve the matrix if it exists
            if subject_id in matrices and session in matrices[subject_id]:
                matrix = matrices[subject_id][session]

                # Plot the matrix on the subplot
                ax = axs[i, j]
                im = ax.matshow(np.log1p(matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
                ax.set_title(f'{subject_id} /sess-{session}', fontsize=35)

                # Create a new axes with a smaller height and add the colorbar to this axes
                divider = make_axes_locatable(ax)
                cax = divider.append_axes("right", size="5%", pad=0.05)
                plt.colorbar(im, cax=cax)

    # Show the figure
    plt.tight_layout()
    plt.show()
```

In [ ]:
```python
#plotting pd's matrices
sessions_to_plot = ["W00", "W36"]
subjects_to_plot = ["101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
visualize_pd_matrices(pd_matrices, pd_list, sessions_to_plot)
```

In [ ]:
```python
def visualize_and_collect_pd_inverse_matrices_separately(matrices, subjects_to_plot, sessions_to_plot):
    # Define the colormap range
    vmin = None
    vmax = None

    # Define the number of columns and rows for the subplots
    n_cols = len(subjects_to_plot)
    n_rows = len(sessions_to_plot)

    # Create a new figure with specified size
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(10 * n_cols, 10 * n_rows))  # Adjust size as needed

    # Dictionaries to store the inverse matrices of each session
    spd_matrices_W00 = []
    spd_matrices_W36 = []

    for i, session in enumerate(sessions_to_plot):
        for j, subject_id in enumerate(subjects_to_plot):
            # Retrieve the matrix if it exists
            if subject_id in matrices and session in matrices[subject_id]:
                matrix = matrices[subject_id][session]

                # Calculate the inverse of the matrix
                inverse_matrix = 1 / (matrix + 1e-9)

                # Add the inverse matrix to the corresponding list
                if session == "W00":
                    spd_matrices_W00.append(inverse_matrix)
                elif session == "W36":
                    spd_matrices_W36.append(inverse_matrix)

                # Plot the matrix on the subplot
                ax = axs[i, j]
                im = ax.matshow(np.log1p(inverse_matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
                ax.set_title(f'{subject_id} /sess-{session}', fontsize=35)

                # Create a new axes with a smaller height and add the colorbar to this axes
                divider = make_axes_locatable(ax)
                cax = divider.append_axes("right", size="5%", pad=0.05)
                plt.colorbar(im, cax=cax)

    # Show the figure
    plt.tight_layout()
    plt.show()

    return spd_matrices_W00, spd_matrices_W36

subjects_to_plot = [ "101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
sessions_to_plot = ["W00", "W36"]
pd_inversematrices_W00, pd_inversematrices_W36 = visualize_and_collect_pd_inverse_matrices_separately(pd_matrice
```

In [ ]:

```python
#Visualizing control matrices
import matplotlib.pyplot as plt
import numpy as np

def visualize_control_matrices(matrices, subjects_to_plot):
    # Define the number of columns and rows for the subplots
    n_cols = 5
    n_rows = 1

    # Initialize a figure and axes object
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(30, 5))
    axs = axs.flatten()  # Flatten to easily iterate over

    # Define the colormap range
    vmin = None
    vmax = None

    # Iterate over the matrices
    i = 0
    for (subject_id, session), matrix in matrices.items():
        # Skip subjects not in the subjects_to_plot list
        if subject_id not in subjects_to_plot:
            continue

        # Plot the matrix on the subplot
        cax = axs[i].matshow(np.log1p(matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
        axs[i].set_title(f'{subject_id} /sess-{session}', fontsize=20)

        # Add a colorbar to the subplot
        fig.colorbar(cax, ax=axs[i])

        # Increment the subplot index
        i += 1

    # If there are more axes than matrices, remove the unused axes
    if i < len(axs):
        for j in range(i, len(axs)):
            fig.delaxes(axs[j])

    # Show the figure
    plt.tight_layout()
    plt.show()

#plotting control's matrices

subjects_to_plot = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
visualize_control_matrices(control_matrices, subjects_to_plot)
```

In [ ]:
```python
#Visualizing control element-wise inverse matrices
import matplotlib.pyplot as plt
import numpy as np

def visualize_and_collect_control_inverse_matrices(matrices, subjects_to_plot):
    # Define the number of columns and rows for the subplots
    n_cols = 5
    n_rows = 1

    # Initialize a figure and axes object
    fig, axs = plt.subplots(n_rows, n_cols, figsize=(30, 5))
    axs = axs.flatten()  # Flatten to easily iterate over

    # Define the colormap range
    vmin = None
    vmax = None

    # List to store the inverse matrices
    icc_matrices = []

    # Iterate over the matrices
    i = 0
    for (subject_id, session), matrix in matrices.items():
        # Skip subjects not in the subjects_to_plot list
        if subject_id not in subjects_to_plot:
            continue

        # Calculate the inverse of the matrix
        inverse_matrix = 1 / (matrix + 1e-9)

        # Plot the matrix on the subplot
        cax = axs[i].matshow(np.log1p(inverse_matrix), vmin=vmin, vmax=vmax, interpolation='nearest')
        axs[i].set_title(f'{subject_id} /sess-{session}', fontsize=20)

        # Add a colorbar to the subplot
        fig.colorbar(cax, ax=axs[i])

        # Add the inverse matrix to the list
        icc_matrices.append(inverse_matrix)

        # Increment the subplot index
        i += 1

    # If there are more axes than matrices, remove the unused axes
    if i < len(axs):
        for j in range(i, len(axs)):
            fig.delaxes(axs[j])

    # Show the figure
    plt.tight_layout()
    plt.show()

    return icc_matrices

subjects_to_plot = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
icc_matrices = visualize_and_collect_control_inverse_matrices(control_matrices, subjects_to_plot)
```

# Topological data analysis : persistent homology

In [ ]:
```python
import matplotlib.pyplot as plt
import gudhi
import numpy as np
## compute separate lists of pd diagrams for each homology degree gudhi:
def compute_and_plot_pd_persistence_diagrams_gudhi(matrices_W00, matrices_W36, subjects_to_plot):
    H0_pd_diagrams_W00 = []
    H1_pd_diagrams_W00 = []
    H2_pd_diagrams_W00 = []
    H0_pd_diagrams_W36 = []
    H1_pd_diagrams_W36 = []
    H2_pd_diagrams_W36 = []

    diagrams_per_row = 3

    for i, (matrix_W00, matrix_W36) in enumerate(zip(matrices_W00, matrices_W36)):
        subject_id = subjects_to_plot[i]

        # Create a figure with subplots for each homology dimension (H0, H1, H2) and session (W00, W36)
        fig, axs = plt.subplots(nrows=2, ncols=diagrams_per_row, figsize=(15, 10))

        # Compute and plot the persistence diagrams for session W00
        compute_and_plot_session(matrix_W00, subject_id, "W00", axs[0, :], H0_pd_diagrams_W00, H1_pd_diagrams_W0

        # Compute and plot the persistence diagrams for session W36
        compute_and_plot_session(matrix_W36, subject_id, "W36", axs[1, :], H0_pd_diagrams_W36, H1_pd_diagrams_W3

        plt.tight_layout()
        plt.show()

    return (H0_pd_diagrams_W00, H1_pd_diagrams_W00, H2_pd_diagrams_W00), (H0_pd_diagrams_W36, H1_pd_diagrams_W36

def compute_and_plot_session(matrix, subject_id, session, axs, H0_pd_diagrams, H1_pd_diagrams, H2_pd_diagrams):
    rips_complex = gudhi.RipsComplex(distance_matrix=matrix, max_edge_length=np.inf)
    simplex_tree = rips_complex.create_simplex_tree(max_dimension=3)
    diagrams = simplex_tree.persistence()

    for dim in range(3):
        diagram_dim = [point for point in diagrams if point[0] == dim]
        gudhi.plot_persistence_diagram(diagram_dim, axes=axs[dim])
        axs[dim].set_title(f'PD {subject_id} /sess-{session} H{dim}')

        # Store diagrams in respective lists
        if dim == 0:
            H0_pd_diagrams.append(diagram_dim)
        elif dim == 1:
            H1_pd_diagrams.append(diagram_dim)
        else:  # dim == 2
            H2_pd_diagrams.append(diagram_dim)

subjects_to_plot = ["101018JW", "101026DD", "101001YM", "101012DE", "101016DC"]
(H0_pd_diagrams_W00, H1_pd_diagrams_W00, H2_pd_diagrams_W00), (H0_pd_diagrams_W36, H1_pd_diagrams_W36, H2_pd_dia
```

In [ ]:
```python
import gudhi as gd
import matplotlib.pyplot as plt
## compute separate lists of control diagrams for each homology degree with gudhi:
def plot_and_store_control_persistence_diagrams(distance_matrices, subjects):
    H0_cc_diagrams = []
    H1_cc_diagrams = []
    H2_cc_diagrams = []

    for i, (distance_matrix, subject) in enumerate(zip(distance_matrices, subjects)):
        fig, axs = plt.subplots(1, 3, figsize=(15, 4))

        # Compute the Rips complex
        rips_complex = gd.RipsComplex(distance_matrix=distance_matrix)

        # Compute the simplex tree
        simplex_tree = rips_complex.create_simplex_tree(max_dimension=3)

        # Compute the persistence diagram
        diag = simplex_tree.persistence()

        # Separate the diagrams by dimension
        diag_H0 = [p for p in diag if p[0] == 0]
        diag_H1 = [p for p in diag if p[0] == 1]
        diag_H2 = [p for p in diag if p[0] == 2]

        # Add diagrams to respective lists
        H0_cc_diagrams.append(diag_H0)
        H1_cc_diagrams.append(diag_H1)
        H2_cc_diagrams.append(diag_H2)

        # Plot H0 diagram
        gd.plot_persistence_diagram(diag_H0, axes=axs[0])
        axs[0].set_title(f'H0 - {subject} - sess-1')

        # Plot H1 diagram
        gd.plot_persistence_diagram(diag_H1, axes=axs[1])
        axs[1].set_title(f'H1 - {subject} - sess-1')

        # Plot H2 diagram
        gd.plot_persistence_diagram(diag_H2, axes=axs[2])
        axs[2].set_title(f'H2 - {subject} - sess-1')

        # Adjust the layout
        plt.tight_layout()
        plt.show()

    return H0_cc_diagrams, H1_cc_diagrams, H2_cc_diagrams

subjects = ["RC4107", "RC4110", "RC4111", "RC4112", "RC4114"]
H0_cc_diagrams, H1_cc_diagrams, H2_cc_diagrams = plot_and_store_control_persistence_diagrams(icc_matrices, subje
```

In [ ]:
```python
#example of a persistence diagram data
H1_cc_diagrams[0]
```

## Conversion of Persistence Diagrams into Persistence Images

In [ ]:
```
#compute and plot persistence images
import numpy as np
import gudhi.representations
import matplotlib.pyplot as plt
import gudhi as gd

def filter_infinite_points(diagram):
    return [(birth, death) for dim, (birth, death) in diagram if np.isfinite(death) and death != 999999999.9999

PI = gd.representations.PersistenceImage(bandwidth=5*1e-2, weight=lambda x: x[1]**1, im_range=[-.5,5,-.5,4],reso

# Transform diagrams to images
H0_pd_images_W00 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H0_pd_diagrams_W00]
H1_pd_images_W00 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H1_pd_diagrams_W00]
H2_pd_images_W00 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H2_pd_diagrams_W00]

H0_pd_images_W36 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H0_pd_diagrams_W36]
H1_pd_images_W36 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H1_pd_diagrams_W36]
H2_pd_images_W36 = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H2_pd_diagrams_W36]

H0_cc_images = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H0_cc_diagrams]
H1_cc_images = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H1_cc_diagrams]
H2_cc_images = [PI.fit_transform(np.array([filter_infinite_points(pd)])) for pd in H2_cc_diagrams]


# Construct a dictionary
subjects_dict = {
    "101018JW_W00": [H0_pd_images_W00[0], H1_pd_images_W00[0], H2_pd_images_W00[0]],
    "101026DD_W00": [H0_pd_images_W00[1], H1_pd_images_W00[1], H2_pd_images_W00[1]],
    "101001YM_W00": [H0_pd_images_W00[2], H1_pd_images_W00[2], H2_pd_images_W00[2]],
    "101012DE_W00": [H0_pd_images_W00[3], H1_pd_images_W00[3], H2_pd_images_W00[3]],
    "101016DC_W00": [H0_pd_images_W00[4], H1_pd_images_W00[4], H2_pd_images_W00[4]],
    "101018JW_W36": [H0_pd_images_W36[0], H1_pd_images_W36[0], H2_pd_images_W36[0]],
    "101026DD_W36": [H0_pd_images_W36[1], H1_pd_images_W36[1], H2_pd_images_W36[1]],
    "101001YM_W36": [H0_pd_images_W36[2], H1_pd_images_W36[2], H2_pd_images_W36[2]],
    "101012DE_W36": [H0_pd_images_W36[3], H1_pd_images_W36[3], H2_pd_images_W36[3]],
    "101016DC_W36": [H0_pd_images_W36[4], H1_pd_images_W36[4], H2_pd_images_W36[4]],
    "RC4107_CC": [H0_cc_images[0], H1_cc_images[0], H2_cc_images[0]],
    "RC4110_CC": [H0_cc_images[1], H1_cc_images[1], H2_cc_images[1]],
    "RC4111_CC": [H0_cc_images[2], H1_cc_images[2], H2_cc_images[2]],
    "RC4112_CC": [H0_cc_images[3], H1_cc_images[3], H2_cc_images[3]],
    "RC4114_CC": [H0_cc_images[4], H1_cc_images[4], H2_cc_images[4]],
}


# Calculate number of columns for subplot
n_images_per_subject = 3  # H0, H1, H2
n_cols = n_images_per_subject

# Define homologies
homologies = ['0', '1', '2']

# Iterate over subjects and display all images
for i, (subject, images) in enumerate(subjects_dict.items()):
    # Create a new figure for each subject
    fig, axs = plt.subplots(1, n_cols, figsize=(3*n_cols,5))

    for j, image in enumerate(images, start=0):
        # Reshape the image to 2D (as per your PI resolution), and flip it vertically
        reshaped_image = image[0].reshape(100,100)
        flipped_image = np.flip(reshaped_image, 0)

        # Add a subplot for the image
        ax = axs[j]
        ax.imshow(flipped_image)

        # Add title with subject and homology information
        ax.set_title(f"Subject: {subject}, H{homologies[j]}")

    # Show the plot
    plt.tight_layout()
    plt.show()
```

# Classification

In [14]:
```
#Discriminating the subjects pd vs control
from sklearn.model_selection import train_test_split

def create_dataset(images_cc, images_pd):
    X = np.concatenate((images_cc, images_pd))
    y = np.concatenate((np.zeros(len(images_cc)), np.ones(len(images_pd)))) # 0 for CC, 1 for PD
    X = [img[0].reshape(-1) for img in X] # Flatten the images
    return train_test_split(X, y, test_size=0.2, random_state=42) # Split into training and test sets
```

In [15]:
```
#training classifiers for each homology to discriminate PD vs Control
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

def train_logistic_regression(X_train, y_train, X_test, y_test):
    clf = LogisticRegression()
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print("Logistic Regression Accuracy:", accuracy)
```

In [16]:
```
from sklearn.svm import SVC

def train_svm(X_train, y_train, X_test, y_test):
    clf = SVC(kernel='linear')
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print("SVM Accuracy:", accuracy)
```

In [17]:
```
for homology, images_cc, images_pd in zip(['H0', 'H1', 'H2'], [H0_cc_images, H1_cc_images, H2_cc_images], [H0_p
    print(f"Training classifiers for {homology}")
    X_train, X_test, y_train, y_test = create_dataset(images_cc, images_pd)
    train_logistic_regression(X_train, y_train, X_test, y_test)
    train_svm(X_train, y_train, X_test, y_test)
```

```
Training classifiers for H0
Logistic Regression Accuracy: 1.0
SVM Accuracy: 1.0
Training classifiers for H1
Logistic Regression Accuracy: 1.0
SVM Accuracy: 1.0
Training classifiers for H2
Logistic Regression Accuracy: 0.5
SVM Accuracy: 0.5
```

In [18]:
```
#Discriminating between pd sessions (W00 vs W36)
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

def train_classifiers(images_W00, images_W36, homology_name):
    # Combine images and labels
    images_combined = images_W00 + images_W36
    labels = [0] * len(images_W00) + [1] * len(images_W36)

    # Reshape images
    images_flatten = [img.flatten() for img in images_combined]

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(images_flatten, labels, test_size=0.2, random_state=42)

    # Logistic Regression
    lr = LogisticRegression()
    lr.fit(X_train, y_train)
    lr_accuracy = accuracy_score(y_test, lr.predict(X_test))

    # SVM
    svm = SVC()
    svm.fit(X_train, y_train)
    svm_accuracy = accuracy_score(y_test, svm.predict(X_test))

    print(f"Training classifiers for {homology_name}")
    print(f"Logistic Regression Accuracy: {lr_accuracy}")
    print(f"SVM Accuracy: {svm_accuracy}")

# Call the function for H0, H1, and H2 homologies
train_classifiers(H0_pd_images_W00, H0_pd_images_W36, "H0")
train_classifiers(H1_pd_images_W00, H1_pd_images_W36, "H1")
train_classifiers(H2_pd_images_W00, H2_pd_images_W36, "H2")
```

```
Training classifiers for H0
Logistic Regression Accuracy: 0.5
SVM Accuracy: 0.5
Training classifiers for H1
Logistic Regression Accuracy: 0.5
SVM Accuracy: 0.5
Training classifiers for H2
Logistic Regression Accuracy: 1.0
SVM Accuracy: 1.0
```

In [19]:
```python
#Discriminating between 3 classes : pds W00 VS pds W36 VS controls
#before paramater tunning
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import numpy as np

def train_SVM(cc_images, pd_images_W00, pd_images_W36):
    # Concatenate the images and create labels
    all_images = np.concatenate([cc_images, pd_images_W00, pd_images_W36])
    all_images = all_images.reshape(all_images.shape[0], -1) # Flatten each image
    labels = [0] * len(cc_images) + [1] * len(pd_images_W00) + [2] * len(pd_images_W36)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(all_images, labels, test_size=0.2, random_state=42)

    # Apply scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Create and fit the SVM model
    clf = svm.SVC(kernel='linear')
    clf.fit(X_train, y_train)

    # Predict the test set results
    y_pred = clf.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy

# Training the classifiers for each homology degree
accuracy_H0 = train_SVM(H0_cc_images, H0_pd_images_W00, H0_pd_images_W36)
accuracy_H1 = train_SVM(H1_cc_images, H1_pd_images_W00, H1_pd_images_W36)
accuracy_H2 = train_SVM(H2_cc_images, H2_pd_images_W00, H2_pd_images_W36)

print("SVM Accuracy for H0:", accuracy_H0)
print("SVM Accuracy for H1:", accuracy_H1)
print("SVM Accuracy for H2:", accuracy_H2)
```

```
SVM Accuracy for H0: 0.3333333333333333
SVM Accuracy for H1: 0.6666666666666666
SVM Accuracy for H2: 0.0
```

In [ ]:
```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
#parameter tunning
def tune_hyperparameters(cc_images, pd_images_W00, pd_images_W36):
    # Concatenate images and labels
    X = np.concatenate([cc_images, pd_images_W00, pd_images_W36], axis=0)
    y = np.array([0] * len(cc_images) + [1] * (len(pd_images_W00) + len(pd_images_W36)))

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Apply scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train.reshape(X_train.shape[0], -1))
    X_test = scaler.transform(X_test.reshape(X_test.shape[0], -1))

    # Define the parameter grid
    param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'linear']}

    # Create a GridSearchCV object
    grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=2, cv=min(3, len(np.unique(y))))

    # Fit the model
    grid.fit(X_train, y_train)

    # Print the best parameters
    print("Best Parameters: ", grid.best_params_)

    # Print the test accuracy
    print("Test Accuracy: ", grid.score(X_test, y_test))

# Tuning hyperparameters for each homology degree
print("Tuning for H0:")
tune_hyperparameters(H0_cc_images, H0_pd_images_W00, H0_pd_images_W36)

print("\nTuning for H1:")
tune_hyperparameters(H1_cc_images, H1_pd_images_W00, H1_pd_images_W36)

print("\nTuning for H2:")
tune_hyperparameters(H2_cc_images, H2_pd_images_W00, H2_pd_images_W36)
```

In [21]:
```python
#Classification after parameter tunning
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import numpy as np

def train_SVM(cc_images, pd_images_W00, pd_images_W36, kernel, C, gamma):
    # Concatenate the images and create labels
    all_images = np.concatenate([cc_images, pd_images_W00, pd_images_W36])
    all_images = all_images.reshape(all_images.shape[0], -1) # Flatten each image
    labels = [0] * len(cc_images) + [1] * len(pd_images_W00) + [2] * len(pd_images_W36)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(all_images, labels, test_size=0.2, random_state=42)

    # Apply scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Create and fit the SVM model with the given kernel, C, and gamma values
    clf = svm.SVC(kernel=kernel, C=C, gamma=gamma)
    clf.fit(X_train, y_train)

    # Predict the test set results
    y_pred = clf.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy

# Training the classifiers for each homology degree with the best parameters
accuracy_H0 = train_SVM(H0_cc_images, H0_pd_images_W00, H0_pd_images_W36, kernel='linear', C=0.1, gamma=1)
accuracy_H1 = train_SVM(H1_cc_images, H1_pd_images_W00, H1_pd_images_W36, kernel='linear', C=0.1, gamma=1)
accuracy_H2 = train_SVM(H2_cc_images, H2_pd_images_W00, H2_pd_images_W36, kernel='rbf', C=0.1, gamma=1)

print("SVM Accuracy for H0:", accuracy_H0)
print("SVM Accuracy for H1:", accuracy_H1)
print("SVM Accuracy for H2:", accuracy_H2)
```

```
SVM Accuracy for H0: 0.3333333333333333
SVM Accuracy for H1: 0.6666666666666666
SVM Accuracy for H2: 0.3333333333333333
```

```
#FPII postprocessing
ConvertSubject(){
    subject=$1
    session=$2
    fileNiiGZ="sub-${subject}_ses-${session}_acq-DWIconcat_run-1_dwi_preproc.nii.gz"
    fileBvec="sub-${subject}_ses-${session}_acq-
DWIconcat_run-1_dwi_preproc.eddy_rotated_bvecs"
    fileBval="sub-${subject}_ses-${session}_acq-DWIconcat_run-1_dwi.bval"
    fileT1="sub-${subject}_ses-${session}_acq-3DT1_rec-uBCnGC_run-1_T1w.nii.gz"

    # Convert NIFTI to .mif and include gradient information
    mrconvert  $fileNiiGZ "dwi.mif" -fslgrad $fileBvec $fileBval

    # Generate mask
    dwi2mask dwi.mif mask.mif

    # Estimating the basis function using Tournier algorithm
    dwi2response tournier dwi.mif response.txt -voxels voxels.mif
    # Viewing the basis functions
    # mrview dwi.mif -overlay.load voxels.mif
    # shview response.txt


    # Applying the basis functions to the diffusion data
    dwi2fod csd dwi.mif -mask mask.mif response.txt fod.mif

    # Viewing the FODs
    # mrview fod.mif

    # Normalizing the FODs
    mtnormalise fod.mif fod_norm.mif -mask mask.mif

    # Convert the anatomical image to MRtrix format
    mrconvert $fileT1 T1.mif

    # Segment the anatomical image with FSL's FAST
    5ttgen fsl T1.mif 5tt_nocoreg.mif

    # Extract the b0 images
    dwiextract dwi.mif - -bzero | mrmath - mean mean_b0.mif -axis 3

    # Convert the b0 and 5tt images
    mrconvert mean_b0.mif mean_b0.nii.gz
    mrconvert 5tt_nocoreg.mif 5tt_nocoreg.nii.gz

    # Extract the grey matter segmentation
    fslroi 5tt_nocoreg.nii.gz 5tt_vol0.nii.gz 0 1

    # Coregister the anatomical and diffusion datasets
    flirt -in mean_b0.nii.gz -ref 5tt_vol0.nii.gz -interp nearestneighbour -dof 6 -omat
diff2struct_fsl.mat

    # Convert the transformation matrix to MRtrix format
    transformconvert diff2struct_fsl.mat mean_b0.nii.gz 5tt_nocoreg.nii.gz flirt_import
diff2struct_mrtrix.txt

    # Apply the transformation matrix to the non-coregistered segmentation data
    mrtransform 5tt_nocoreg.mif -linear diff2struct_mrtrix.txt -inverse 5tt_coreg.mif

    # View the coregistration in mrview
    # mrview dwi.mif -overlay.load 5tt_nocoreg.mif -overlay.colourmap 2 -overlay.load
5tt_coreg.mif -overlay.colourmap 1

    # Create the grey matter / white matter boundary
    5tt2gmwmi 5tt_coreg.mif gmwmSeed_coreg.mif

    # View the GM/WM boundary
    # mrview dwi.mif -overlay.load gmwmSeed_coreg.mif
```

```
    # Create streamlines with tckgen
    tckgen -act 5tt_coreg.mif -backtrack -seed_gmwmi gmwmSeed_coreg.mif -nthreads 16 -
maxlength 250 -cutoff 0.06 -select 40000000 fod_norm.mif tracks_40M.tck

    # Extract a subset of tracks
    tckedit tracks_40M.tck -number 200k smallerTracks_200k.tck

    # View the tracks in mrview
    # mrview dwi.mif -tractography.load smallerTracks_200k.tck

    # Sift the tracks with tcksift2
    tcksift2 -act 5tt_coreg.mif -out_mu sift_mu.txt -out_coeffs sift_coeffs.txt -nthreads
16 tracks_40M.tck fod_norm.mif sift_1M.txt

    #Running recon-all
    export SUBJECTS_DIR=$(pwd)
    recon-all -i $fileT1 -s recon -all

    #Converting the labels
    fsLocation="/home/imane/mambaforge/envs/myenv/share/mrtrix3/labelconvert/
fs_a2009s.txt"
    labelconvert recon/mri/aparc.a2009s+aseg.mgz $FREESURFER_HOME/FreeSurferColorLUT.txt
$fsLocation parcels.mif

    #Coregistering the parcellation:
    mrtransform parcels.mif -interp nearest -linear diff2struct_mrtrix.txt -inverse -
datatype uint32 parcels_coreg.mif

    #Creating the connectome WITH coregistration
    tck2connectome -symmetric -zero_diagonal -scale_invnodevol -tck_weights_in sift_1M.txt
tracks_40M.tck parcels_coreg.mif parcels_coreg.csv -out_assignment
assignments_parcels_coreg.csv

}

# ---------------------- END OF FUNCTION DEFINITION -----------------------------------


subjects=( "101018JW", "101026DD", "101001YM", "101012DE", "101016DC")
sessions=( "W00", "W36" )

# Add your free surfer install to home.
export FREESURFER_HOME=/home/imane/freesurfer/usr/local/freesurfer/7.4.1
source $FREESURFER_HOME/SetUpFreeSurfer.sh


for subj in "${subjects[@]}"
do
    for sess in "${sessions[@]}"
    do
        # Move to Subject Path.
        cd $subj
        cd $sess

        echo "Currently working on Subject $subj, session $sess"
        ConvertSubject $subj $sess
        printf "\n"

        # Go out to base path
        cd ..
        cd ..
    done
done
```

```
# Openneuro post+preprocessing
ConvertSubject(){
    subject=$1
    session=$2
    fileNiiGZ="sub-${subject}_ses-${session}_dwi.nii.gz"
    fileBvec="sub-${subject}_ses-${session}_dwi.bvec"
    fileBval="sub-${subject}_ses-${session}_dwi.bval"

    # nii, bvec, bval
    #Preprocessing
    # Convert NIFTI to .mif and include gradient information
    mrconvert  $fileNiiGZ "dwi.mif" -fslgrad $fileBvec $fileBval

    # Denoising
    dwidenoise dwi.mif dwi_den.mif -noise noise.mif

    # Calculate residual
    mrcalc dwi.mif dwi_den.mif -subtract residual.mif


    dwifslpreproc dwi_den.mif dwi_den_preproc.mif -rpe_none -nocleanup -pe_dir AP -
eddy_options " --slm=linear"


    # Bias correction
    dwibiascorrect ants dwi_den_preproc.mif dwi_den_preproc_unbiased.mif -bias bias.mif

    #Post processing
    # Generate mask
    dwi2mask dwi_den_preproc_unbiased.mif mask.mif

    # Estimating the basis function using Tournier algorithm
    dwi2response tournier dwi_den_preproc_unbiased.mif response.txt -voxels voxels.mif
    # Viewing the basis functions
    # mrview dwi.mif -overlay.load voxels.mif
    # shview response.txt


    # Applying the basis functions to the diffusion data
    dwi2fod csd dwi_den_preproc_unbiased.mif -mask mask.mif response.txt fod.mif

    # Viewing the FODs
    # mrview fod.mif

    # Normalizing the FODs
    mtnormalise fod.mif fod_norm.mif -mask mask.mif

    # Convert the anatomical image to MRtrix format
    mrconvert ${ANAT_PATH}/sub-${subject}_ses-${session}_T1wbrain.nii.gz T1.mif

    # Segment the anatomical image with FSL's FAST
    5ttgen fsl T1.mif 5tt_nocoreg.mif

    # Extract the b0 images
    dwiextract dwi.mif - -bzero | mrmath - mean mean_b0.mif -axis 3

    # Convert the b0 and 5tt images
    mrconvert mean_b0.mif mean_b0.nii.gz
    mrconvert 5tt_nocoreg.mif 5tt_nocoreg.nii.gz

    # Extract the grey matter segmentation
    fslroi 5tt_nocoreg.nii.gz 5tt_vol0.nii.gz 0 1

    # Coregister the anatomical and diffusion datasets
    flirt -in mean_b0.nii.gz -ref 5tt_vol0.nii.gz -interp nearestneighbour -dof 6 -omat
diff2struct_fsl.mat

    # Convert the transformation matrix to MRtrix format
```

```
    transformconvert diff2struct_fsl.mat mean_b0.nii.gz 5tt_nocoreg.nii.gz flirt_import
diff2struct_mrtrix.txt

    # Apply the transformation matrix to the non-coregistered segmentation data
    mrtransform 5tt_nocoreg.mif -linear diff2struct_mrtrix.txt -inverse 5tt_coreg.mif

    # View the coregistration in mrview
    # mrview dwi.mif -overlay.load 5tt_nocoreg.mif -overlay.colourmap 2 -overlay.load
5tt_coreg.mif -overlay.colourmap 1

    # Create the grey matter / white matter boundary
    5tt2gmwmi 5tt_coreg.mif gmwmSeed_coreg.mif

    # View the GM/WM boundary
    # mrview dwi.mif -overlay.load gmwmSeed_coreg.mif

    # Create streamlines with tckgen
    tckgen -act 5tt_coreg.mif -backtrack -seed_gmwmi gmwmSeed_coreg.mif -nthreads 16 -
maxlength 250 -cutoff 0.06 -select 40000000 fod_norm.mif tracks_40M.tck

    # Extract a subset of tracks
    tckedit tracks_40M.tck -number 200k smallerTracks_200k.tck

    # View the tracks in mrview
    # mrview dwi.mif -tractography.load smallerTracks_200k.tck

    # Sift the tracks with tcksift2
    tcksift2 -act 5tt_coreg.mif -out_mu sift_mu.txt -out_coeffs sift_coeffs.txt -nthreads
16 tracks_40M.tck fod_norm.mif sift_1M.txt

    #Running recon-all
    export SUBJECTS_DIR=$(pwd)
    recon-all -i ${ANAT_PATH}/sub-${subject}_ses-${session}_T1wbrain.nii.gz -s recon -all

    #Converting the labels
    fsLocation="/home/imane/mambaforge/envs/myenv/share/mrtrix3/labelconvert/
fs_a2009s.txt"
    labelconvert recon/mri/aparc.a2009s+aseg.mgz $FREESURFER_HOME/FreeSurferColorLUT.txt
$fsLocation parcels.mif

    #Coregistering the parcellation:
    mrtransform parcels.mif -interp nearest -linear diff2struct_mrtrix.txt -inverse -
datatype uint32 parcels_coreg.mif

    #Creating the connectome WITH coregistration
    tck2connectome -symmetric -zero_diagonal -scale_invnodevol -tck_weights_in sift_1M.txt
tracks_40M.tck parcels_coreg.mif parcels_coreg.csv -out_assignment
assignments_parcels_coreg.csv



}

# ---------------------- END OF FUNCTION DEFINITION -----------------------------------


subjects=("RC4115" "RC4110" "RC4111" "RC4112" "RC4113" "RC4114" )
sessions=("1")

# Add your free surfer install to home.
DATASET_PATH="/media/imane/DATA/openneuro/ds001907"
#set freesurfer home
export FREESURFER_HOME=/home/imane/freesurfer/usr/local/freesurfer/7.4.1
source $FREESURFER_HOME/SetUpFreeSurfer.sh
#set ants path
export ANTSPATH=/opt/ANTs/bin/
export PATH=${ANTSPATH}:$PATH
```

```
for subject in "${subjects[@]}"
do
    for session in "${sessions[@]}"
    do
         # Define paths
        DWI_PATH="${DATASET_PATH}/sub-${subject}/ses-${session}/dwi"
        ANAT_PATH="${DATASET_PATH}/derivatives/sub-${subject}/ses-${session}/anat"

         # Navigate to the DWI directory
        cd ${DWI_PATH}


        echo "Currently working on Subject $subject, session $session"
        ConvertSubject $subject $session
        printf "\n"



    done
done
```

EL Missaoui Imane

## Uncovering The Topological Features For The Characterisation Of A New Biomarker In Parkinson's Disease

### Abstract:

The development of new biomarkers for **Parkinson's disease** prognosis is crucial to enhance our predictive capabilities. In this context, we introduce **topological data analysis**, specifically the method of **persistent homology** as a promising approach in investigating brain networks, offering a potential imaging biomarker . The main objective is to identify disease-specific topological patterns using **machine learning**. Two distinct approaches both derived from **diffusion MRI** data are used: **fiber Orientation Distribution Functions** (fODFs), used to capture the geometric properties of white matter fibers, and the **structural connectome**, used to map connections between brain regions. The results obtained from the analysis of persistent homology of the structural connectome have enabled the identification of significant changes in structural connectivity that may be linked to disease progression. In conclusion, this innovative approach, which combines Topological Data Analysis, machine learning, fODFs, and the structural connectome, offers promising prospects for a better understanding of Parkinson's disease progression.

**Keywords:** Topological Data Analysis, Persistent homology, diffusion MRI, Parkinson's disease, fiber Orientation Distribution Function, Structural Connectome, Machine Learning.

## Mise En Évidence Des Caractéristiques Topologiques Pour La Caractérisation D'un Nouveau Biomarqueur Dans La Maladie de Parkinson.

### Résumé:

Le développement de nouveaux biomarqueurs pour le pronostic de la **maladie de Parkinson** est crucial pour améliorer nos capacités prédictives. Dans ce contexte, nous introduisons **l'analyse de données topologiques**, plus précisément la méthode d'**homologie persistante**, en tant qu'approche prometteuse pour l'exploration des réseaux cérébraux, offrant un potentiel biomarqueur d'imagerie. L'objectif principal est d'identifier des schémas topologiques spécifiques à la maladie en utilisant **l'apprentissage automatique**. Deux approches distinctes, toutes deux dérivées des données d' **IRM de diffusion**, sont utilisées : les **fonctions de distribution d'orientation des fibres** (fODFs), utilisées pour capturer les propriétés géométriques des fibres de matière blanche, et le **connectome structurel**, utilisé pour cartographier les connexions entre les régions cérébrales. Les résultats obtenus à partir de l'analyse de l'homologie persistante du connectome structurel ont permis d'identifier des changements significatifs dans la connectivité structurelle qui pourraient être liés à la progression de la maladie. En conclusion, cette approche innovante, qui combine l'analyse de données topologiques, l'apprentissage automatique, les fODFs et le connectome structurel, offre des perspectives prometteuses pour une meilleure compréhension de la progression de la maladie de Parkinson.

**Mots clés:** Analyse topologique des données, homologie persistante, IRM de diffusion, maladie de Parkinson, fonction de distribution de l'orientation des fibres, connectome structurel, apprentissage automatique.