Linear Inverse Problems on Manifolds with Graph Signal Processing Methods

Thèse de Master - 2023
soumise le 23 Juin 2023
à la Faculté Informatique et Communications
École Polytechnique Fédérale de Lausanne
pour l'obtention partielle du grade de Master ès Sciences
par

Kaan Okumuş

supervisée par:

Dr Matthieu Simeoni, EPFL Center for Imaging, superviseur de thèse
Dr Joan Rué Queralt, EPFL Center for Imaging, co-superviseur de thèse

Lausanne, EPFL, 2023

To my beloved ones...

Acknowledgements

First of all, I would like to express my special thanks to Dr. Matthieu Simeoni. He has been my advisor, always helpful in any situation and guided me with his vision. His constructive criticism and contributions have helped me to make a lot of progress in this thesis. Another special thanks go to Dr. Joan Rue Queralt, who has been my cosupervisor for the whole project in my thesis and has been the biggest contributor to my progress in my thesis with his help and guidance. I met with him almost every week and he helped me to make a lot of progress in different phases of the project. From the acquisition of the real data to the implementation of the operators in Pycsou, he helped me in many subjects. I would like to thank him very much for every comment and guidance he gave me. I would also like to express my thanks to Dr. Hugo Fluhr for accepting to review my thesis as an external expert.

I would like to express my sincere thanks from my heart to my family, especially my mother Sabriye Okumuş, my father Aydın Okumuş and my sister Başak Okumuş. Even though my parents stay in Turkey and my sister lives in Sweden, they have always been there for me and have always been supportive in my most difficult moments. I cannot thank them enough. If I have made it this far in my life, it is entirely due to the path my parents have shown me. My sister's master's thesis and achievements have always been a source of inspiration for me. She has always helped me to stay strong with the advice she gave me during this thesis process.

One of my deepest thanks from my heart goes to my girlfriend Evrim Ecem Saçmalı, who studies architecture in Politecnico di Milano. We both went through a similar thesis process and we have always been there for each other every day of our thesis. We helped each other to stay strong in our most difficult times. I can't thank you enough, my love. I couldn't have gotten this far in my project without you. I am glad you are in my life and we went through this process together with love and passion.

And my last thanks go to my best friend Aybars Karakaş who studies medicine in Turkey. Even though we could not be in a constant contact, we have always supported each other and let our friendships last forever.

Lausanne, 23 Juin 2023 Kaan Okumuş

Abstract

In real-world, when any physical data is measured by an acquisition system, the resulting signal may not be equivalent to the original signal. For example, when an object is photographed, the actual input image is convolved with the camera's point spread function (PSF) and Gaussian noise is added. The problem of obtaining the original data from the output data is called inverse problems, and these can be solved with a Bayesian approach and proximal algorithms for signals in the Euclidean domain. In this thesis, these problems are generalized to manifolds which are irregular Euclidean domains. Two methods are proposed for this purpose. The first one is the tangent plane projection, which allows the discretization of manifold signals in a Euclidean domain from an intrinsic point of view, and the other one is the use of graph signals, which allows a better modeling of the geometric structure from an extrinsic perspective. In this thesis, for the second solution, the theory of graph signal processing is first studied. As a result, graph differential and convolution operators are obtained and integrated into the formulation of inverse problems of graph signals. Then computational aspects of these methods are discussed and Pycsou-GSP is developed as an extension of Pycsou to graph signal processing, where Pycsou is a Python package for the state-of-the-art proximal algorithms. Faster solutions are proposed for graph differential and graph convolution operators in Pycsou-GSP with the use of Numba package. Then, the complete system is used for an application in astronomy. Celestial sphere is chosen as the manifold and HEALPix pixelisation is applied for spherical data. Two proposed solutions are tested and compared for a synthetic data created by ground truth signal. As a result, the graph-based solution gives better results and it is also tested for real-world manifold signal, where Wisconsin H-Alpha Mapper (WHAM) signal is chosen. From the results, the discretization of manifold signals with graph signals and the graph-based solution of linear inverse problems for manifolds are shown to give stronger predictions.

Key words: linear inverse problems, Bayesian solution, proximal algorithms, generalized LASSO problem, manifolds, tangent plane projection, graph signal processing, calculus on graph signals, development of Pycsou-GSP, HEALPix graphs

Résumé

Dans le monde réel, lorsqu'une donnée physique est mesurée par un système d'acquisition, le signal résultant peut ne pas être équivalent au signal original. Par exemple, lorsqu'un objet est photographié, l'image d'entrée réelle est convoluée avec la fonction d'étalement des points de l'appareil photo et un bruit gaussien est ajouté. Le problème de l'obtention des données d'origine à partir des données de sortie est appelé problème inverse. Il peut être résolu par une approche bayésienne et des algorithmes proximaux pour les signaux dans le domaine euclidien. Dans cette thèse, ces problèmes sont généralisés aux manifolds qui sont des domaines euclidiens irréguliers. Deux méthodes sont proposées à cette fin. La première est la projection du plan tangent, qui permet la discrétisation des signaux de manifolds dans un domaine euclidien d'un point de vue intrinsèque, et l'autre est l'utilisation de signaux de graphes, qui permet une meilleure modélisation de la structure géométrique d'un point de vue extrinsèque. Dans cette thèse, pour la deuxième solution, la théorie du traitement des signaux de graphe est d'abord étudiée. En conséquence, des opérateurs différentiels et de convolution de graphes sont obtenus et intégrés dans la formulation de problèmes inverses de signaux de graphes. Ensuite, les aspects informatiques de ces méthodes sont discutés et Pycsou-GSP est développé comme une extension de Pycsou au traitement des signaux de graphe, où Pycsou est un paquetage Python pour les algorithmes proximaux de l'état de l'art. Des solutions plus rapides sont proposées pour les opérateurs différentiels et de convolution de graphes dans Pycsou-GSP avec l'utilisation du paquetage Numba. Le système complet est ensuite utilisé pour une application en astronomie. La sphère céleste est choisie comme collecteur et la pixellisation HEALPix est appliquée aux données sphériques. Les deux solutions proposées sont testées et comparées pour des données synthétiques créées à partir d'un signal de vérité au sol. La solution basée sur les graphes donne de meilleurs résultats et elle est également testée pour le signal du collecteur du monde réel, où le signal du Wisconsin H-Alpha Mapper (WHAM) est choisi. Les résultats montrent que la discrétisation des signaux de collecteurs à l'aide de signaux graphiques et la solution basée sur les graphes des problèmes linéaires inverses pour les collecteurs donnent de meilleures prédictions.

Mots clefs : problèmes linéaires inverses, solution bayésienne, algorithmes proximaux, problème LASSO généralisé, manifolds, projection du plan tangent, traitement des signaux de graphes, calcul sur les signaux de graphes, développement de Pycsou-GSP, graphes HEALPix

Contents

Acknowledgements i

Abstract (English/Français) iii

List of Figures ix

List of Tables, Algorithms and Codes xi

Nomenclature xiii

1 Introduction 1

2 Linear Inverse Problems on Manifolds 5
- 2.1 Linear Inverse Problems on Euclidean Spaces . . . 5
- 2.2 Linear Inverse Problems on Manifolds . . . 11
  - 2.2.1 What are Manifolds? . . . 11
  - 2.2.2 Manifold Signals . . . 13
  - 2.2.3 Problem Definition . . . 13
  - 2.2.4 Proposed Solutions . . . 14

3 Theory of Graph Signal Processing 17
- 3.1 Graph Theory . . . 17
- 3.2 Graph Signals . . . 21
- 3.3 Calculus for Graph Signals . . . 24
  - 3.3.1 Graph Differential Operators . . . 24
  - 3.3.2 Graph Signal Smoothness . . . 26
- 3.4 Graph Spectral Theory . . . 28
  - 3.4.1 Eigen-Decomposition of Graph Laplacian . . . 28
  - 3.4.2 Graph Fourier Transform . . . 30
- 3.5 Generalized Operators for Graph Signals . . . 31
  - 3.5.1 Convolution . . . 31
  - 3.5.2 Filtering . . . 32
  - 3.5.3 Shifting . . . 33
  - 3.5.4 Modulation . . . 34
  - 3.5.5 Dilation . . . 34

3.6 Integration of GSP Methods to Inverse Problems on Manifolds 35

4 Computational Aspects of Graph Signal Processing 37
4.1 Challenges of Implementing GSP Methods 37
4.2 Proposed Implementations in GSP Methods 39
4.2.1 General Strategy 39
4.2.2 Graph Differential Operators 40
4.2.3 Graph Filtering Operators 45
4.3 Development of Pycsou-GSP 48
4.3.1 Overall Structure 48
4.3.2 Integration of Differential and Convolution Methods 50
4.3.3 Features, Usages, Limitations and Future Works 52

5 Application in Astronomy 55
5.1 Problem Definition 55
5.2 HEALPix Graph Construction 58
5.3 Comparison of Methods on Synthetic Data 59
5.4 Implementation of Proposed Graph-Based Method on Real-World Signal 62

6 Conclusion 67

A Appendix: Validation of Calculus for Graph Signals 71
A.1 Proof of Adjointness of Graph Gradient and Graph Divergence 71
A.2 Proof of Graph Hessian Being the Trace of Graph Laplacian 71
A.3 Proof of Graph Laplacian Being Positive Semi-Definite 72
A.4 Proof of Mean Conservation in Shift Operator 72

B Appendix: Theory of Classical Signal Processing 73
B.1 Differential Operators in Classical Calculus 73
B.1.1 Gradient Operator 73
B.1.2 Divergence Operator 73
B.1.3 Laplacian Operator 73
B.2 Definition of the Spectrum 74
B.2.1 Eigen-Decomposition of Laplacian 74
B.2.2 Fourier Transform 74

C Graph Filtering Design Examples 75
C.1 Diffusion Problem 75
C.2 Tikhonov Regularization 75
C.3 Edge Detection by Ideal High Pass Filter 78

Bibliography 85

Curriculum Vitae 87

List of Figures

2.1 Block diagram of linear inverse problem for continuous domain. 6
2.2 Block diagram of discretised linear inverse problem. 7
2.3 Different examples of simple 2D manifolds. 11
2.4 Example of 2D manifold with two tangent spaces. 12
2.5 Example of tangent plane projection of earth manifold signal. 14
2.6 Discretisation of continuous Stanford bunny manifold: obtaining the point clouds and triangular meshes of continuous manifold. 16

3.1 Topological visualization of a simple undirected graph and its weight adjacency matrix. 18
3.2 Different graph types. 20
3.3 Different graph construction examples. 22
3.4 Temperature signal for Swiss cities and visualization of signals for classical and graph signal processing. 23
3.5 Image grid 2d graph signal and Bunny graph signal examples. 24
3.6 Graphs with the same signal values and vertices with different edges. 28
3.7 Eigenvectors of the combinatorial graph Laplacian corresponded to eigenvalues λ_0, λ_1, λ_2 and λ_{50} and zero crossing values for different number of eigenvalues. 29
3.8 Graphs of heat kernels for rates $\tau = 1$ and $\tau = 20$ in vertex and graph spectral domain. 31

4.1 An example of RCD format. 40
4.2 Performance time comparison of different Numba arguments for matrix-free jit-compiled solution of graph gradient. 42
4.3 Performance time comparison of Numba jit-compiled gradient with sparse dot implementation for four different graph types. 42
4.4 Performance time comparison of Numba jit-compiled divergence with sparse dot implementation for four different graph types. 44
4.5 Performance time comparison of Numba jit-compiled Laplacian of combinatorial and normalized with sparse dot implementation for two different graph types. 45

4.6 Chebyshev approximation of heat kernel spectrum $\hat{g}(\lambda_l) = e^{-\lambda_l}$ with performance analysis for different orders. 47
4.7 Chebyshev approximation of discrete sampled heat kernel spectrum $\hat{g}(\lambda_l) = e^{-\lambda_l}$ for order $m = 3,5,7$. 48
4.8 Structure of Pycsou-GSP 49
4.9 Block diagram of the algorithm presented in GraphConvolution operator with method of chebyshev. 52

5.1 Celestial and Earth spheres. 56
5.2 Orthographic front view and gnomonic view with $(0,90)$ degree rotation of synthetic signal on celestial manifold. 57
5.3 Orthographic view from front and back of Hydrogen-alpha ions intensity signal on celestial manifold measured by Wisconsin H-Alpha Mapper (WHAM) telescope. 57
5.4 Block diagram of graph-based and tangent plane projection solutions for the application of inverse problems on celestial manifold signals. 58
5.5 Block diagram of HEALPix graph construction. 58
5.6 Orthographic view of the HEALPix pixelisation of the sphere for $N_{\text{side}} = 1,2,4,8$. 59
5.7 Gnomonic views of ground truth, noisy output and reconstructed synthetic signals for linear inverse problem on manifold solved by graph signal processing methods and tangent plane projection, where output signal is resulted from graph convolution of ground truth with heat kernel of rate $\tau = 5$ with addition of Gaussian noise. 61
5.8 Mollweide views of ground truth, noisy output and reconstructed WHAM signals for linear inverse problem on manifold solved by graph signal processing, where output signal is resulted from heat kernel filtered input with addition of Gaussian noise. 64
5.9 Mollweide views of ground truth, noisy output and reconstructed WHAM signals for linear inverse problem on manifold solved by graph signal processing, where output signal is resulted from spherical downsampled input with addition of Gaussian noise. 65

C.1 Heat kernel on manifolds with rate $\tau = 1,10$. 76
C.2 Heat kernel on graph signals of rates $\tau = 1,10$ with their spectrum. 76
C.3 Tiknonov problem solution by kernel on bunny graph signal with rate $\gamma = 10$. . 78
C.4 High pass filtering on bunny graph signal. 79

List of Tables, Algorithms and Codes

Tables:

2.1 Choices of cost functional based on noise modeling. 8
2.2 Choices of regularising functional based on apriori knowledge. 9

3.1 Graph smoothness comparison table. 28

4.1 Different choices of Numba arguments, which are nopython, parallel, fastmath, nogil and cache. 41

Algorithms:

1 Accelerated Proximal Gradient Descent. 10
2 Primal Dual Splitting Algorithm. 11

Codes:

4.1 Codes for implementation of sparse dot solution of graph gradient. 40
4.2 Codes for implementation of vectorized solution in RCD format of graph gradient. 41
4.3 Codes for implementation of Numba jit-compiled matrix-free solution of graph gradient. 41
4.4 Codes for implementation of sparse dot solution of graph gradient. 43
4.5 Codes for implementation of Numba jit-compiled matrix-free solution of graph divergence. 43
4.6 Codes for implementation of sparse dot solution of graph Laplacian for types combinatorial or normalized according to input data “lap_type“. 44
4.7 Codes for implementation of Numba jit-compiled matrix-free solution of graph Laplacian for types combinatorial or normalized according to input data “lap_type“. 44
4.8 Sample codes for the creation of Pycsou-GSP graph object. 49
4.9 Sample codes for conversion to RCD format. 50
4.10 Example code for integration of matrix-free jit-compiled solutions to GraphGradient as Pycsou.abc.LinOp. 50
4.11 Example code for conversion to RCD format. 51

4.12 Sample code for usages of Pysou-GSP differential and convolution operators with testing by comparison with Pygsp solution as well. 53
4.13 Output of the code in Listing 4.12. 53

Nomenclature

2D	Two dimension
3D	Three dimension
α	Discretized input signal in $\mathbb{R}^N$ if f is continuous manifold signal
$\alpha^\star$	Solution of the estimation of input discrete signal α
exp	Exponential function
$\hat{f}$	Graph Fourier transform of graph signal f
$\imath$	Indicator function
$\mathbb{R}$	Real numbers set
$\mathbb{R}^N$	N dimensional real vector space
Λ	Diagonal matrix with diagonal elements of eigenvalues of Laplacian operator
$\mathbf{D}$	Degree matrix of graph $\mathscr{G}$
$\mathbf{G}$	Linear operator $\mathbb{R}^M \to \mathbb{R}^N$ of acquisiton system
$\mathbf{L}$	Laplacian matrix of graph $\mathscr{G}$
$\mathbf{U}$	Eigenvector matrix of Laplacian operator, also inverse graph Fourier transform operator
$\mathbf{U}^T$	Transpose of $\mathbf{U}$, also graph Fourier transform operator
$\mathbf{W}$	Weight adjacency matrix of graph $\mathscr{G}$
$\mathbf{Y}$	Random vector of Measured output data
$\mathbf{y}$	Measured output data in $\mathbb{R}^M$
$\mathscr{D}_s$	Dilation operator by scale s

$\mathscr{E}$ Set of edges of graph $\mathscr{G}$ with number of edges N_e

$\mathscr{G}$ Graph

$\mathscr{H}(\mathscr{X})$ Hilbert space of manifold signals $f : \mathscr{X} \to \mathbb{R}$

$\mathscr{L}^2(\mathbb{R})$ Space of square integrable functions defined on real numbers

$\mathscr{M}_k$ Modulation operator by modulation k

$\mathscr{N}(a;b)$ Gaussian random variable with mean a and standard deviation b

$\mathscr{N}(i,K)$ Neighborhood of vertex i with path less than K

$\mathscr{T}_k$ Shifting operator by shift k

$\mathscr{V}$ Set of vertices of graph $\mathscr{G}$ with number of nodes N

$\mathscr{X}$ Continuous manifold domain

∇ Gradient operator either for signal in Euclidean space, manifold or graph signal.

$\overline{T}_k$ Scaled Chebyshev polynomial

Φ Analysis operator of basis $\{\varphi\}_{i=1}^N$

Φ^* Synthesis operator of basis $\{\varphi\}_{i=1}^N$, adjoint of Φ

div Divergence operator either for signal in Euclidean space, manifold or graph edge signal.

$\tilde{\mathbf{L}}$ Normalized Laplacian matrix of graph $\mathscr{G}$

$\tilde{L}$ Normalized Laplacian operator either for signal in Euclidean space, manifold or graph edge signal.

$\{\lambda_l\}_{l=0}^{N-1}$ Eigenvalues of Laplacian operator

$\{\theta_k\}$ Monomial polynomial coefficients

$\{\varphi_i\}_{i=1}^N$ Set of basis vectors for $\mathbb{R}^N$

$\{c_k\}$ Chebyshev polynomial coefficients

$\{u\}_{l=0}^{N-1}$ Eigenvectors of Laplacian operator

$d_{\mathscr{G}}(i,j)$ Geodesic distance from vertex i to vertex j

F Either data fidelity function in Bayesian approach solution of inverse problems or graph edge signal

f Input continuous manifold signal or discrete graph signal

f_p	Projected signal in $\mathbb{R}^N$ Euclidean space obtained from manifold signal
H	Hessian operator either for signal in Euclidean space, manifold or graph edge signal.
H	Linear operator $\mathscr{L}^2(\mathbb{R}) \rightarrow \mathbb{R}^N$ of acquisiton system
L	Laplacian operator either for signal in Euclidean space, manifold or graph edge signal.
M	Number of measurements in output
N	Number of sample points to input signal
O	Big O notation
P	Projection operator to manifold signal to $\mathbb{R}^N$
$p_{\mathbf{n}}$	Probability distributed function of $\mathbf{n}$
R	Regularization function in Bayesian approach solution of inverse problems
$T\mathscr{X}$	Tangent bundle on manifold $\mathscr{X}$
T_k	Chebyshev polynomial
$T_x\mathscr{X}$	Tangent space of point x on manifold $\mathscr{X}$
Jit	Just-in Time compilation

1 Introduction

In real life, many problems can be solved with mathematical modeling. Many features and results can be extracted from any data with certain modeling. The field of signal processing can propose many problems in this regard. One of the most basic application areas is the effort to access the real data of any output data that we measure or observe. For example, when we take a picture of an object with a digital camera, we cannot get the real image exactly due to certain filtering characteristic of the camera we have, and we get a noisy signal. The effort to mathematically model the digital camera and predict the real signal from the output signal that we obtain is actually an example of inverse problem solving. In this problem, the signal is a 2D image signal and we can describe it in Euclidean space.

In this way, we can apply many existing solution methods in the literature for linear inverse problems in the Euclidean domain. In general, this method is ill-posed. A Bayesian approach can be adopted to solve this problem. In this approach, the relationship between the original and the acquired signals is modeled probabilistically. In this model, many assumptions can be made that can be used in real practice. In general, the real signal is modeled by filtering it with a linear system and adding noise. Then we try to maximize the posterior probability, which gives the probability of the input signal when we have the output signal. This probability is obtained by Bayes' theorem from the likelihood function that forms the fitting term and the apriori probability function of the input signal that forms the regularization term. Thus, the maximum aposterior probability problem becomes an optimization problem where we estimate the true signal.

Different problems can be obtained with different apriori information and noise models. These problems can also be solved by proximal algorithms for signals in the Euclidean domain. This solution requires discretization of continuous signals. However, such problems do not only apply to signals in the Euclidean domain. Some signals can be defined in more complex domains. For example, any geometric object can be represented by two-dimensional surfaces. For example, a human body, a spherical model of the world or an organ can be represented by such surfaces. Manifolds that are locally in the Euclidean domain but not globally in the Euclidean domain can be used to represent such data. One can define manifold signals

with signals defined on these structures. In order to apply inverse problems to such signals, discretization is required and this cannot be done with a classical sampling method. Therefore, this thesis proposes two different discretization methods for manifolds. First, a traditional method, the projection method, can be used. Here, any point on the manifold is chosen and the projection method is applied on the Euclidean space at that point. In this method, manifolds are treated from an intrinsic point of view and the manifold structure is ignored. As a result, a signal is obtained in Euclidean space and the solution of this signal for inverse problems is done as mentioned. Finally, the estimated signal is interpolated again and the original manifold signal is estimated. The second method, and the main one proposed in this thesis, is the representation of manifolds by graphs. Graphs are simply a set of nodes and edges and have the potential to enable extrinsic modeling of manifolds.

The second method, and the main one proposed in this thesis, is the representation of manifolds by graphs. Graphs are simply a set of nodes and edges and have the potential to model manifolds both extrinsically and intrinsically. This again results in graph signals that are not in the Euclidean domain. In this case, inverse problems need to be redefined for graph signals. At this point in the project, a graph signal processing method needs to be investigated and studied. This requires the study of graph theory and the mathematical definition of graph signals. Here, Hilbert fields and the associated inner product and norm metrics are defined for graph signals. On top of these definitions, calculus methods for graph signals are studied. Differential operators such as gradient divergence and Laplacian Hessian are defined. The gradient operator can be particularly useful for first order differencing. This feature can help to define smoothness metrics and different metrics can be defined. The Laplacian operator plays an important role in providing a spectrum description for graphical signals. Just like in classical signal processing, eigenfunctions form a basis for graphical signals and the analysis operator of this basis defines the graphical Fourier transform. Eigenvalues represent the concept of frequency, allowing the spectrum to be analyzed in a different domain for graphical signals. On top of this, many general operators defined in classical signal processing such as shifting, convolution, dilation, modulation are defined for graphic signals. In particular, it is not possible to define a change such as shifting in the vertex domain of graphic signals. This definition can be used in graphic signal processing by using the equivalents in the frequency domain of classical signal processing. As a consequence, graph convolution is defined as the product of input and filter signals in the graph spectral domain. Similarly, graph filtering is also specified using the definition of graph convolution. Thus, low-pass filters with effects such as blurring and downsampling can be applied with this definition. As a result, graph filtering can be used directly for modeling inverse problems. In Generalized LASSO and Generalized Tikhonov, the norm of the difference operator of the signal is taken in the regularization function. For graph signals, this operator can be preferred as graph gradient. Thus, this inverse problem is formulated in a discrete form and can be solved with similar proximal algorithms.

In order to implement the proposed graph signal processing method, it is necessary to analyze these methods computationally and implement them in a computer environment. Although there are specific libraries designed for this purpose, such as Pygsp, in order to use the Pycsou

library rich in proximal algorithms, it is necessary to develop a separate library suitable for this environment. Therefore, in this project, a library called Pycsou-GSP is developed as an extension of Pycsou for graphical signal processing. It basically implements the visualization of graphical signals and is heavily inspired by pygsp. The main center of this package is the differential and convolution operators. The main strategy in developing these modules is to implement them as a Pycsou.abc.operator.LinOp class, which is both fast and in line with the principles of Pycsou version 2. In order to provide the fastest solution, the jit compilation of the Numba package is used. This method optimizes the Python language's code switching, allowing code with for loops to run faster. Thus, this method is used in gradient, divergence and Laplacian and is shown to be faster than the sparse matrix multiplication used in Pygsp. For convolution, the spectral filtering method is found to be computationally expensive and the Chebyshev approximation method is applied. This also provides a fast implementation of graph filtering. The integration of these methods into Pycsou's LinOp class also takes into account the fact that Pycsou is module agnostic and precision agnostic for Cupy, Dask.Array and Numpy. This results in a library that is practically ready for graphical signal processing solutions.

In order to bring all these solutions together and test them, an application in astronomy is chosen. Here the data of the celestial map obtained by telescopes on Earth is used. The data is considered to be obtained from the surface of an imaginary sphere enclosing the earth. Here the manifold represents the surface of this celestial sphere. Thus, the manifold signal is measured by passing the manifold signal through the point spread function of the telescope and the signal is measured in a noisy way. To obtain the graphic signal, the HEALPix algorithm for pixelization of spheres is used. The two methods proposed here as solutions to the inverse problem are tested and the results are compared. As a result, we can see that the graphical signal processing method gives better results. In particular, it is concluded that graphical signals are much more suitable for modeling the geometric structure of manifolds than tangent plane projection.

In this thesis, there are 6 chapters in total, including an introduction. First, in chapter 2, linear inverse problems are studied on manifolds. Here we describe the data model of this problem in Euclidean space, its formulation, the Bayesian approach solution, the resulting optimization problem and how this problem can be solved by proximal algorithms. Then manifolds are defined and the mathematical properties of manifold signals are discussed. Then the inverse problems are reformulated in manifold signals and two solutions are proposed: tangent plane projection method and graph signal processing method. To explore the details of the second solution, the theory of graph signal processing is studied in Chapter 3. First, graph theory and graph signals are defined. Then calculus methods for graph signals are defined and the graph spectral domain is studied. Inspired by classical signal processing, classical operators such as shifting, dilation, modulation and convolution are generalized to graph signals. As a result, graph filtering is defined and the integration of methods for the reformulation of inverse problems is explained. Thus, the computational aspects of graph signal processing are described in Chapter 5 for the implementation of this method in a computer environment.

First, specific difficulties in the implementation are listed. Then the proposed methods for graph differential and convolution operators are studied and compared with other methods. Finally, the development of a Python package called Pycsou-GSP is described. The overall structure of the package, the integration of the differential and convolution operators and the resulting package's features, usage, limitations and future work are discussed. Then, in chapter 5, all the proposed methods are tested with an application in astronomy. First, the application is described and HEALPix, the graph generation method for the application, is explained. Then the graph-based solution is tested with real data. Finally, the results are discussed and compared with the tangent plane projection method on synthetic data. Finally, the conclusion of the thesis is made in Chapter 6.

2 Linear Inverse Problems on Manifolds

Inverse problems deal with the recovery of the input signal from measured output data through a noisy acquisition system. An example is an imaging system where the input is the actual image of the captured object and the output is the noisy image. Traditionally, such signals are defined in the Euclidean domain and there are many solution techniques in the literature [1]–[4] for the inverse problems in Euclidean spaces. However, such signals can be more appropriately modeled by a non-Euclidean domain such as a manifold. In this case, the problem must also be solved for manifold signals that are defined in irregular non-Euclidean domain. Therefore, this chapter explores how to adapt linear inverse problem techniques for manifolds. First of all, techniques for the problem in Euclidean spaces are presented such as the formulation of signal modeling, reformulation into an optimisation problem by Bayesian approach and the computational solution by proximal algorithms. Then, mathematical description of manifold signals are given along with a discussion of why they are useful for better modeling. Using this, linear inverse problems are redefined for such signals. Finally, two different methods for solving inverse problems on manifolds are proposed. The first one is the traditional solution of projection of signals into Euclidean space by ignoring the structure of the manifold, while the second one offers sampling of manifold by graph, which results in the main proposal of the thesis, which requires further analysis in the following chapters.

2.1 Linear Inverse Problems on Euclidean Spaces

Recovery of the input causal signals from a set of noisy measured output data is the focus of an inverse problem. In order to investigate the problem in a general manner, the unknown input signal to be estimated is denoted as f and assumed to be continuous in Euclidean space, $\mathscr{L}^2(\mathbb{R})$. Also, it is assumed that there are M number of measurements for the sensed data, which can be denoted as $\mathbf{y} \in \mathbb{R}^M$.

The Bayesian approach, which models the whole system in a probabilistic manner, can be utilized to solve this problem as explained by Dashti et al in their article [5] and Stuart in his

book [6] in detail. With this method, the noisy measurements from the sensing device are assumed to be the outcomes of the random variable $\mathbf{Y}: \Omega \to \mathbb{R}^M$. As stated by Simeoni's lecture notes [1], the mean of this random variable $\mathbb{E}[\mathbf{Y}]$ is to be approximated by its one-sample empirical estimate $\mathbf{y}$: $\mathbb{E}[\mathbf{Y}] = \mathbf{y}$. Let the sensing noise be denoted as $\mathbf{n}: \Omega \to \mathbb{R}^M$, the model becomes as follows

$$\mathbf{Y} = \mathbf{y} + \mathbf{n} \tag{2.1}$$

Another assumption for the problem is that the measurements are unbiased and linear. To model the acquisition system of the sensing device, a linear operator denoted as H is proposed: $\mathbf{y} = Hf$. This operator can be seen as an analysis operator associated to some basis $\{\phi_i\}_{i=1}^M$:

$$H: \begin{cases} \mathscr{L}^2(\mathbb{R}^d) \to \mathbb{R}^M \\ f \mapsto \begin{bmatrix} < f, \phi_1 > & \dots & < f, \phi_M > \end{bmatrix}^T \end{cases} \tag{2.2}$$

where the details about analysis and synthesis operators with basis definitions can be seen in [7]. Here Φ could be one of the sampling functionals in real-world problems. As Simeoni proposes in his lecture notes [1], some of the candidates are spatial sampling, subsampling, Fourier/Radon sampling, filtering, mean-pooling and so on.

From all of the assumptions, the complete model is as follows

$$\begin{aligned} \mathbf{Y} &= Hf + \mathbf{n} \\ \mathbb{E}[\mathbf{Y}] &= Hf \end{aligned} \tag{2.3}$$

Based on this Bayesian approach, the linear inverse problems attempt to estimate the original input signal f from $y = Hf$. The depiction of this problem can be seen in Figure 2.1.

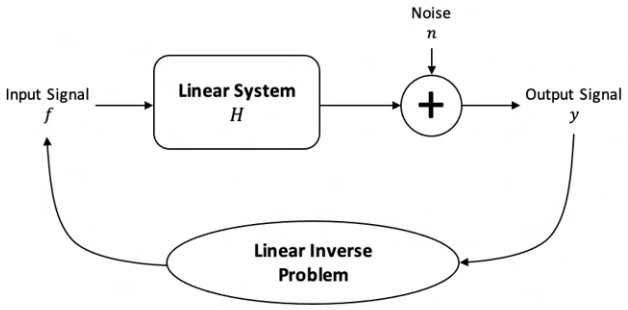


Figure 2.1: Block diagram of linear inverse problem for continuous domain.

Discretisation of the Problem:

In Figure 2.1, the signal to be estimated is continuous, which makes the solution impossible with a feasible iterative algorithm, i.e. a computational solution in a computer environment

with limited storage space. Therefore, as proposed by Gupta, Fageot and Unser in their article [8], the traditional solution is to discretize the problem by representing the input continuous signal in discrete space. For this purpose, the pixelisation is introduced as a natural candidate, which is a linear operator that can be denoted as Φ^* as stated in the lecture notes of Simeoni [1]. This can be seen as an analysis operator associated to some other basis denoted as $\{\varphi_i\}_{i=1}^N$ as the details can be seen in [7]. In order to obtain the continuous signal again, the adjoint of the operator Φ can be applied as an interpolation as stated in [1], which has the following definition:

$$\Phi : \begin{cases} \mathbb{R}^M \to \mathscr{L}^2(\mathbb{R}^d) \\ \alpha \mapsto \sum_{i=1}^N \alpha_i \varphi_i \end{cases} \tag{2.4}$$

Then, the main problem becomes as follows

$$\mathbf{Y} = Hf + \mathbf{n} = H\Phi\alpha + \mathbf{n} = \mathbf{G}\alpha + \mathbf{n} \tag{2.5}$$

Here, the signal to be estimated denoted as α is discrete and the whole system is depicted in Figure 2.2. The input continuous signal can be obtained again by using the synthesis operator Φ as stated in [1].

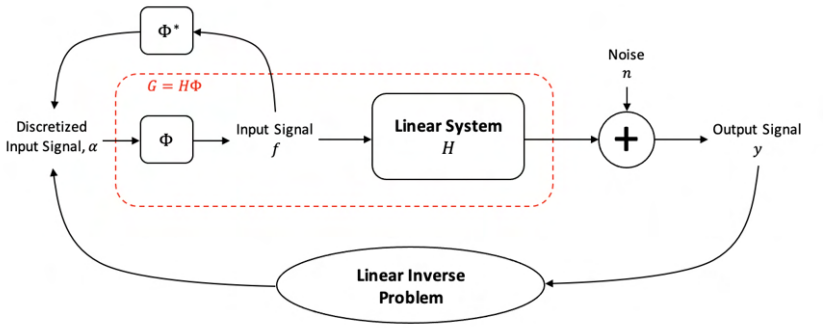


Figure 2.2: Block diagram of discretised linear inverse problem.

In general, the problem $\mathbf{y} = \mathbf{G}\alpha$ is ill-posed as proposed by Simeoni [1], Hansen [9]. As discussed by Simeoni [1], first of all, the problem may have no solution in case G is not surjective, which results in the data vector $\mathbf{y}$ not to belong to the range space of G. Secondly, more than one solution may exist in case G is not injective. This is possible when the null space of G is not equal to the set of null vector. Finally, the solutions may be numerically unstable. For surjective $\mathbf{G}$, the minimal norm solution to $\mathbf{y} = \mathbf{G}\alpha$ results in virtually unbounded reconstruction map. In this situation, small changes have a significant impact on the solution.

Because of the ill-posed characteristic of the problem, Bayesian approach is proposed for the system as described in Equation 2.5 and Figure 2.2, which is proposed by [6] as well. In this approach, some prior information for the noise $\mathbf{n}$ and input discrete signal α from Equation 2.5 are suggested. These are probability distribution functions of noise $p_{\mathbf{n}}(y)$ and input signal $p(\alpha)$

and choices of these functions are listed in Table 2.1 and 2.2. Having this model, conditional probability distribution function of the input discrete signal given output signal, in other words *a posteriori probability*, denoted as $p(\alpha|\mathbf{y})$, is attempted to be maximized for the optimal estimation of the input signal. This is called as *Maximum A Posteriori Probability (MAP)* estimation and explained in detail by Pishro-Nik [10] in detail. The formulation is as follows

$$\begin{aligned}\alpha^{\star} &= \arg\max_{\alpha\in\mathbb{R}^N} p(\alpha|\mathbf{y}) \equiv \arg\max_{\alpha\in\mathbb{R}^N} p(\mathbf{y}|\alpha).p(\alpha) \\ &\equiv \arg\min_{\alpha\in\mathbb{R}^N} -\log p(\mathbf{y}|\alpha) - \log p(\alpha)\end{aligned} \tag{2.6}$$

Here, Bayes' theorem, $p(\alpha|\mathbf{y}) = p(\mathbf{y}|\alpha)p(\alpha)/p(\mathbf{y})$, is applied for the derivation as done in [5] and [6] as well. As stated in the article [11] of Vidal et al, MAP estimation can be seen as a regularization of maximum likelihood estimation, where $\log p(\mathbf{y}|\alpha)$ is called as *likelihood function* and $\log p(\alpha)$ is related to the regularization of the input signal as presented. Then, this problem can be denoted by the following minimisation problem in general:

$$\alpha^{\star} = \arg\min_{\alpha\in\mathbb{R}^N} F(\mathbf{y},\mathbf{G}\alpha) + \lambda R(\alpha) \tag{2.7}$$

where $F : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}_+ \cup \{0\}$ is data fidelity or cost term, while $R : \mathbb{R}^N \to \mathbb{R}_+ \cup \{0\}$ is regularization term and λ is called as regularization constant. The latter is related to penalizing the solution that is far away from the output signal. The optimal choice of the cost functional is dependent on the apriori knowledge on the noise and the corresponding results can be seen in Table 2.1. On the other hand, the former term is related to favoring the parsimonous solution, which can be interpreted as applying Occam's razor principle. This is related to the choice of simplest one until further evidence is presented as explained in [12]. Some of the most common examples of regularizing functions explained in [1], [9] are listed in Table 2.2.

Table 2.1: Choice of cost functional based on noise modeling.

Noise Type	Noise PDF	Cost Functional, $F(\mathbf{y},\mathbf{Gx})$
Noiseless	$p_{\mathbf{n}}(y) = 0$	$\imath(\mathbf{y}-\mathbf{Gx}) = \begin{cases} 0, & \text{if } \mathbf{y}=\mathbf{Gx} \\ \infty, & \text{o.w.} \end{cases}$
Gaussian Noise	$p_{\mathbf{n}}(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\mathbf{y}^T\mathbf{y}\right)$	$\|\|\mathbf{y}-\mathbf{Gx}\|\|_2^2$
Laplacian Noise	$p_{\mathbf{n}}(y) = \frac{1}{2\sigma} \exp\left(-\frac{\|x\|}{\sigma}\right)$	$\|\|\mathbf{y}-\mathbf{G}\alpha\|\|_1$

Example - Generalised LASSO Problem:

As explained by Ali et al in their article [13], in generalised LASSO problem, the noise $\mathbf{n}$ from the model described in Equation 2.5 is Gaussian from Table 2.1. Then, from $(\mathbf{y}|\alpha) \sim \mathcal{N}(\mathbf{G}\alpha;\sigma)$, the likelihood function $p(\mathbf{y}|\alpha)$ can be determined. The regularization type is total variation from Table 2.2, which comes from the Laplacian a priori probability with a difference operator

Table 2.2: Choice of regularising functional based on a priori knowledge where **D** is a finite difference operator.

Regularization Type	Apriori Information	Regularising Functional, $R(\alpha)$
Tikhonov	$p_\alpha(\alpha) \propto \exp\left(-\|\|\alpha\|\|_2^2\right)$	$\|\|\alpha\|\|_2$
Generalised Tikhonov	$p_\alpha(\alpha) \propto \exp\left(-\|\|\mathbf{D}\alpha\|\|_2^2\right)$	$\|\|\mathbf{D}\alpha\|\|_2$
ℓ_1	$p_\alpha(\alpha) \propto \exp\left(-\|\|\alpha\|\|_1\right)$	$\|\|\alpha\|\|_1$
Total Variation	$p_\alpha(\alpha) \propto \exp\left(-\|\|\mathbf{D}\alpha\|\|_1\right)$	$\|\|\mathbf{D}\alpha\|\|_1$

D. Then, the maximum a posteriori probability (MAP) problem results in as follows

$$\begin{aligned}
&\arg\max_{\alpha\in\mathbb{R}^N} p(\alpha|y) \equiv \arg\max_{\alpha\in\mathbb{R}^N} p(y|\alpha)p(\alpha) \\
&\equiv \arg\min_{\alpha\in\mathbb{R}^N} -\log(p(y|\alpha)) - \log(p(\alpha)) \\
&\equiv \arg\min_{\alpha\in\mathbb{R}^N} -\ln(\exp\left((\mathbf{y}-\mathbf{G}\alpha)^T(\mathbf{y}-\mathbf{G}\alpha)\right)) - \ln(\exp\left(-||\mathbf{D}\alpha||_1\right)) \\
&\equiv \arg\min_{\alpha\in\mathbb{R}^N} ||\mathbf{y}-\mathbf{G}\alpha||_2^2 + \lambda||\mathbf{D}\alpha||_1
\end{aligned} \tag{2.8}$$

where the result has the same form as Equation 2.7.

Example - Tikhonov Problem:

Similar approach is applied to define this type of problem as presented in [1]. Here, the noise **n** is Gaussian from Table 2.1 and the regularisation type is also generalised Tikhonov from Table 2.2, which is led by Gaussian a priori information. Following the similar steps, MAP problem results in as follows

$$\arg\min_{\alpha\in\mathbb{R}^N} ||\mathbf{y}-\mathbf{G}\alpha||_2^2 + \lambda||\alpha||_2^2 \tag{2.9}$$

Solution of the Optimisation Problem:

As stated by Simeoni's lecture notes [1], most of the optimisation problems from linear inverse systems have the following form in practice

$$\arg\min_{\alpha\in\mathbb{R}^N} F(\alpha) + G(\alpha) + H(\mathbf{K}\alpha) \tag{2.10}$$

under the assumption that F is convex and differentiable functional with β-Lipschitz continuous gradient, G and H are two proper, lower semicontinuous and convex functions with simple proximal operators and **K** is a linear operator. All of these assumptions enable the existence and unicity of the solution with an iterative algorithm from the mathematical proofs presented in [1].

Under the given assumption, the minimisation of $G(\alpha)$ with respect to α can be solved efficiently by *proximal minimisation* algorithm as proposed by [1]. This is because $G(\alpha)$ can

be represented by simple proximal operator, which is explicitly formulated as $\text{prox}_{\tau,G}(\mathbf{x}) = \arg\min_{\alpha\in\mathbb{R}^N} G(\alpha)+\frac{1}{2\tau}||\alpha-\mathbf{x}||_2^2$ as in [2]. This proximal operator gives the solution of $\arg\min_{\alpha\in\mathbb{R}^N} G(\alpha)$ when input of the proximal operator is equal to α and this is achieved by iteratively updating the estimated signal as $\alpha_n = \text{prox}_{\tau,G}(\alpha_{n-1})$, where n is the iteration index and τ is related to convergence rate. This is what is done in proximal minimisation. On the other hand, *gradient descent* is a better method for minimising differentiable functions like $F(\alpha)$ with respect to α as stated by Chong [14]. In this algorithm, the estimated function is iteratively updated by $\alpha_n = \alpha_{n-1} - \tau\nabla F(x_{n-1})$, where ∇ is gradient operator and τ is convergence rate. When it's compared to proximal minimisation, gradient descent can handle a wide range of differentiable functions, whereas proximal minimisation specializes in handling non-smooth functions.

Accelerated Proximal Gradient Descent (APGD) is the combination of these two strategies with the inclusion of Nesterov acceleration as explained in detail by Li et al [15]. This algorithm provides a solution of the problem $\arg\min_{\alpha\in\mathbb{R}^N} F(\alpha) + G(\alpha)$. The algorithm can be seen in Algorithm 1.

Algorithm 1 Accelerated Proximal Gradient Descent.

Require: τ, σ, $\alpha_0 = z_0$
  for all $n \geq 1$ **do**
    $z_n \leftarrow \text{prox}_{\tau,G}(\alpha_{n-1} - \tau\nabla F(\alpha_{n-1}))$ ▷ Proximal Minimisation + Gradient Descent
    $\alpha_n \leftarrow z_n + \frac{n-1}{n+\sigma}(z_n - z_{n-1})$ ▷ Nesterov Acceleration Part
  end for
  return $\{\alpha_n\}_{n=1}^N$

APGD is suitable for solving Tikhonov and LASSO problem, where $F(\alpha) = ||\mathbf{y} - \mathbf{G}\alpha||_2^2$ and $G(\alpha) = \lambda||\alpha||_2$ or $G(\alpha) = \lambda||\alpha||_1$. For LASSO, by obtaining $\nabla F(\alpha) = \mathbf{G}^T(\mathbf{G}\alpha - \mathbf{y})$ and $\text{prox}_{\lambda||.||_1}(\alpha) = \text{soft}_\lambda(\alpha)$, this yields *Fast Iterative Soft Thresholding Algorithm (FISTA)* as presented by Beck et al [3].

The optimisation problem in Equation 2.10 cannot be solved by APGD. To solve this, the problem is converted into a saddle point problem, which is $\arg\min_{\alpha\in\mathbb{R}^N}\max_{\mathbf{z}\in\mathbb{R}^M} F(\alpha)+G(\alpha)-H^*(\mathbf{z})+\mathbf{z}^T\mathbf{K}\alpha$, where H^* is Fenchel conjugate of H, where the definitions of saddle-point problem can be seen in [16]. The definition and properties of Fenchel conjugate can be seen in the article [17] by Bergmann et al. It's seen that the saddle point problem can be achieved by variable splitting method and addition of Lagrange multipliers as suggested by [1]. As a result, this problem can be divided into the iterative solution of two variables α and $\mathbf{z}$ and *Proximal Gradient Descent* algorithm is suitable for the update of these parameters. Then, with the choice of momentum term $\rho > 0$, the update for the next iteration are controlled. The overall algorithm is called as *Primal Dual Splitting (PDS)* and from Condat's analysis [18], the formulation is given in Algorithm 2.

PDS is suitable for generalised Tikhonov and generalised LASSO problem, where $F(\alpha) = ||\mathbf{y} - \mathbf{G}\alpha||_2^2$, $G(\alpha) = 0$ and $H(\alpha) = ||\alpha||_2$ or $H(\alpha) = ||\alpha||_1$ with $\mathbf{K} = \mathbf{D}$ which is the difference

operator.

Algorithm 2 Primal Dual Splitting Algorithm.

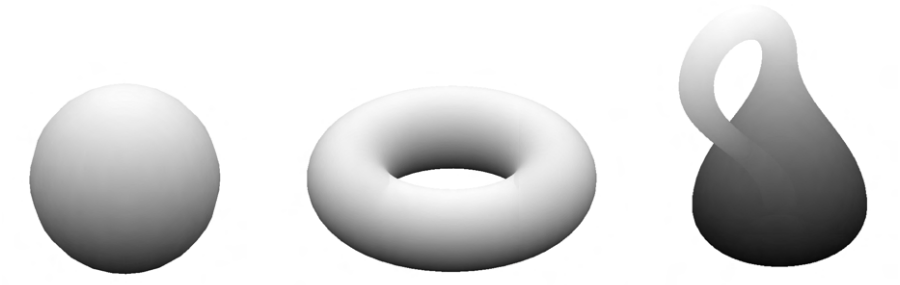
Require: $\tau, \sigma, \rho, \alpha_0, z_0$
  for all $n \geq 1$ **do**
    $\tilde{\alpha}_n \leftarrow \text{prox}_{\tau,G}(\alpha_{n-1} - \tau\nabla F(\alpha_{n-1}) - \tau\mathbf{K}^* z_{n-1})$ ▷ PGD for $\arg\min_{\alpha\in\mathbb{R}^N} F(\alpha) + G(\alpha) - \mathbf{z}^T K\alpha$
    $\tilde{z}_n \leftarrow \text{prox}_{\sigma,H^*}(z_{n-1} + \sigma\mathbf{K}[2\tilde{\alpha}_n - \alpha_{n-1}])$ ▷ PGD for $\arg\max_{\mathbf{z}\in\mathbb{R}^M} \mathbf{z}^T K\alpha - H^*(z)$
    $\alpha_n \leftarrow \rho\tilde{\alpha}_n + (1-\rho)\alpha_{n-1}$ ▷ Momentum for the Update
    $z_n \leftarrow \rho\tilde{z}_n + (1-\rho)z_{n-1}$ ▷ Momentum for the Update
  end for
  return $\{\alpha_n\}_{n=1}^N$

2.2 Linear Inverse Problems on Manifolds

In this section, an attempt is made to solve linear inverse problems to manifolds, a useful mathematical object for representing geometric structural data. The main issue arises from the fact that the manifolds lie in the non-Euclidean domain. Therefore, we first give mathematical definitions of manifolds and manifold signals. Then, different methods for the linear inverse problem on manifolds are proposed for this thesis.

2.2.1 What are Manifolds?

A manifold is a topological space that is locally Euclidean near each point, while being globally non-Euclidean. This phenomena can be understood simply by the example on spherical surface modeling of Earth. For the perspective of a human staying on the surface of Earth, the planet seems to be planar, which is Euclidean. However, it has spherical structure from the perspective of a human in the space, that makes it globally non-Euclidean. A few simple examples of 2D manifolds as a surface of 3D objects like sphere, toroid and Klein bottle can be seen in Figure 2.3.



(a) Sphere. (b) Toroid. (c) Klein bottle.

Figure 2.3: Different examples of simple 2D manifolds for the surface of 3D geometric object.

Mathematically, a manifold is d-dimensional where each point on the manifold has a neighborhood that is topologically equivalent to d-dimensional Euclidean space as stated by Bronstein et al in their article [19]. Such spaces are called as *tangent space*, which can be denoted as $T_x\mathscr{X}$. In Figure 2.4, two tangent spaces of two different points on 2D manifold can be seen. The infinite set of tangent spaces of each points on the manifold is called as *tangent bundle*, which can be denoted as $T\mathscr{X}$.

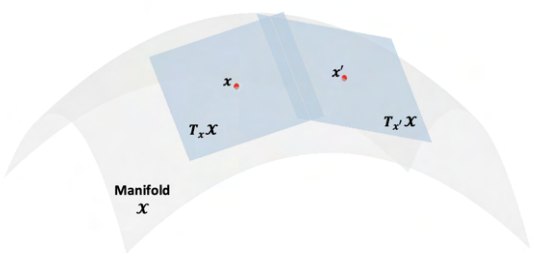


Figure 2.4: Example of 2D manifold with two different tangent spaces.

In the article [19] by Bronstein et al, on each tangent space, an inner product $< .,. >: T_x\mathscr{X} \times T_x\mathscr{X} \rightarrow \mathbb{R}$ is defined and called as *Riemannian metric*. A manifold with this metric is also called as *Riemannian manifold* as stated by Chavel in his book [20]. This type of manifold can be realized as a subset of a Euclidean space. According to the Nash embedding theorem stated in [21], any sufficiently smooth Riemannian manifold can be represented in an appropriately large-dimensional Euclidean space. As an example of an embedding, 3D geometric objects can be modeled by 2D manifolds, which are embedded in 3D Euclidean space. Such manifolds are frequently used in computer graphics and computer vision for a variety of applications. In this thesis, we mostly focus on such 2D manifolds that represent the surface of 3D geometric object.

The embeddings of the manifold by Euclidean spaces do not need to be unique. Different realizations of the same Riemannian metric are called as *isometries* as defined in Bronstein et al article [19]. Related to that concept, there are two specific types of properties for manifolds, which are *intrinsic* and *extrinsic*, as stated by do Carmo et al's book [22]. The former is the properties that do not change the isometries of the manifold, i.e. the Riemannian metrics such as curvature, geodesics and topology. The latter is related to the specific realization of the manifold in the Euclidean space, which facilitates visualization, analysis and computation. Intrinsic and extrinsic analyses offer complementary insights into the geometry of a manifold. Neglecting either analysis can result in an incomplete understanding of the manifold's geometry as stated by Bhattacharya et al [23]. As an example, neglecting the extrinsic analysis can result in flawed embeddings and visualizations, potentially distorting the manifold's structure, on the other hand, erroneous characterizations of shape is led by ignoring the intrinsic analysis.

2.2.2 Manifold Signals

A continuous signal can be defined on the points of manifolds as well. Let $f : \mathscr{X} \to \mathbb{R}$ denote the manifold signal. It should be noted that the signal is not defined in Euclidean domain, while it has Euclidean tangent space for each point. Another type of function for manifolds is tangent vector field $F : \mathscr{X} \to T\mathscr{X}$ that attaches tangent vectors to each point: $F(x) \in T_x\mathscr{X}$. The Hilbert spaces of these two functions can be defined as $\mathscr{H}(\mathscr{X})$ and $\mathscr{H}(T\mathscr{X})$ as presented by Bronstein et al's article [19].

The inner product of $\mathscr{H}(\mathscr{X})$ can also be defined as

$$< f, g >_{\mathscr{H}(\mathscr{X})} = \int_{\mathscr{X}} f(x)g(x)dx \tag{2.11}$$

and the inner product of $\mathscr{H}(T\mathscr{X})$ can be defined as

$$< F, G >_{\mathscr{H}(T\mathscr{X})} = \int_{\mathscr{X}} < F(x)G(x) >_{T_x\mathscr{X}} dx \tag{2.12}$$

The definitions of Hilbert spaces and signals provide the generalizations of calculus methods. The difference operators such as gradient, divergence, Laplacian as given in Appendix A.1 for the classical calculus can be attempted to be defined for the manifolds. However, it's problematic to do so due to the continuous non-Euclidean nature of the manifolds. According to Bronstein et al's article [19], the intrinsic gradient denoted as $\nabla : \mathscr{H}(\mathscr{X}) \to \mathscr{H}(T\mathscr{X})$ and intrinsic divergence $\mathrm{div} : \mathscr{H}(T\mathscr{X}) \to \mathscr{H}(\mathscr{X})$ can be defined, which are the operators defined in the direction of tangent spaces. Based on these, Laplacian $L : \mathscr{H}(\mathscr{X}) \to \mathscr{H}(\mathscr{X})$ is also defined as the intrinsic divergence of the intrinsic gradient. Thus, it's also completely based on the intrinsic properties of the manifold. In the article, a differential operator $df : T\mathscr{X} \to \mathbb{R}$ from both extrinsic and intrinsic point of view is defined as follows

$$df(x)F(x) = < \nabla f(x), F(x) >_{T_x\mathscr{X}} \tag{2.13}$$

Even though this equation 2.13 is coordinate-free, the choice of basis functions of each tangent space is required. For a general manifold, it's very problematic to define infinitely many bases for each different tangent spaces unless very specific type of manifolds are given.

2.2.3 Problem Definition

After the definition of manifolds, linear inverse problems for manifold signals can be studied. Let us denote this type of signal as $f : \mathscr{X} \to \mathbb{R}$. As explained in Section 2.1 and shown in Equation 2.3, the data model is $\mathbf{Y} = Hf + \mathbf{n}$. Then, discretization of this problem should be applied to propose an iterative solution. Sampling functions are defined with some basis for the continuous signal in Euclidean space, which results in Equation 2.5 as stated by Simeoni [1]. However, it's problematic to define the basis of the analysis operators for the manifolds. For example, for the pixelisation operator, there are no well-defined pixel grids for a generic

manifold. In addition, the difference operators are also problematic to be defined canonically as explained in the previous section. For example, the generalised regularisation cannot be applied for manifolds as it requires the difference operator. Therefore, this problem cannot be directly solved by manifolds systematically and it requires the discretisation of the continuous manifold signal.

To discretise the manifold signal for the linear inverse problems, two different solutions are proposed and explained in the next section. The first one is the traditional approach, while the second one constitutes the main proposal of this thesis.

2.2.4 Proposed Solutions

Solution By Tangent Space Projection

The manifold signal is projected into a tangent space of a chosen point on the manifold as presented in Boumal [24]. For example, 2D manifold for the surface of a 3D object can be projected into 2D Euclidean space. Tangent plane of a chosen point on the surface can be chosen as such Euclidean space. Given the projection operator as $P : \mathscr{X} \rightarrow T_x\mathscr{X}$, the resulted signal is denoted as $f_p = Pf$. This operation is illustrated in Figure 2.5 for Earth manifold signal as an example.

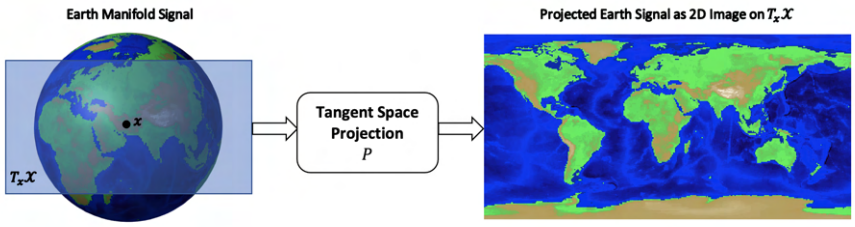


Figure 2.5: Example of tangent plane projection of earth manifold signal.

Here, the obtained continuous signal f_p in Euclidean space is suitable for the model depicted in Figure 2.2. Then, a sampling functional denoted as Φ^* needs to be chosen to discretise the continuous signal as explained in Section 2.1. The discrete signal to be estimated becomes $\alpha = \Phi^* f_p$. Then, the problem results in a discretised linear inverse problem on Euclidean space, which can be solved by the methods explained in Section 2.1.

Main problem of this method is that the continuous manifold is handled only by an intrinsic point of view. This perspective is resulted by looking at the signal on single tangent space. This approach neglects the extrinsic analysis of the model, which results in the deformations of the manifold's structure. Thus, it is expected to have undesired noisy fluctuations in the signal estimated by this method.

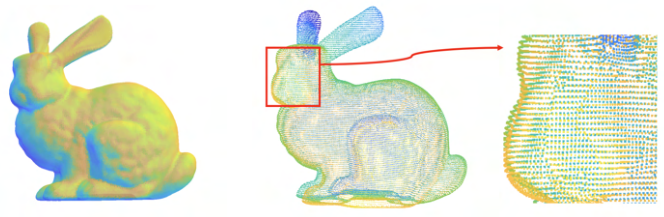
Solution By Graph Signal Processing

In order to handle the problem on the continuous manifolds by including the extrinsic perspective, some form of direct sampling is needed for such data types. According to Bronstein et al's article [19], the graphs are useful candidate for modeling the manifold in discrete domain. In a very basic definition, the graphs are set of pair of vertices and edges. Here, the vertices can be attributed to the point cloud of the manifold, which can be obtained by pixelisation of the manifold. Then, the edges can be created by either by *triangular meshing* or *k nearest neighbor (kNN)* algorithm.

In triangular meshing, the surface is divided into a collection of finite number of triangles that cover the entire surface, where vertices and edges of triangles correspond to those of the constructed graph as explained by Botsch et al [25]. On the other hand, for kNN algorithm, k closest points of each vertex are chosen according to the distance in coordinate space for creating edges as proposed by Kang [26]. When it comes to extrinsic analysis of a manifold, a graph created by triangular meshing is generally better suited due to its explicit surface representation, ability to analyze surface geometry and perform surface-based operations, and its compatibility with visualization techniques. On the other hand, a kNN graph primarily captures the local relationships and intrinsic properties of the manifold, without explicit reference to its embedding.

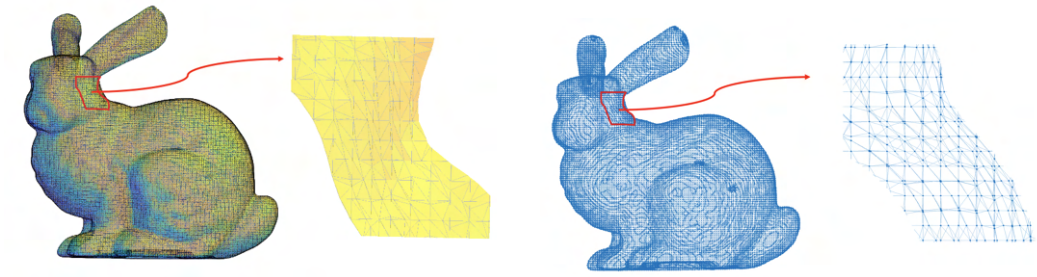
Then, the edge weights of the graphs can be decided to correctly model the complex structure of manifolds according to some measurements related to the distance metrics. The results of the whole operation graphs whose edges created by triangular meshing and kNN algorithm are illustrated in Figure 2.6. In Figure 2.6b, the point cloud of continuous manifold is seen which directly represents the vertices of graph and in Figure 2.6c and 2.6d, we can see the edges generated on top of this point cloud by triangular meshing and kNN algorithm.

The careful definition and analysis of graph signals are essential to adapt the optimisation problem formulated in Equation 2.7 for signals on graph data. This leads to the need to define Hilbert spaces, inner products, differential operators, smoothness measurements, spectral analysis, generalized operators etc. of the signals in the graph setting. For this purpose, the theory of graph signal processing is studied in detail in the next chapter.



(a) Continuous Stanford bunny manifold.

(b) Point cloud of Stanford bunny.



(c) Stanford bunny graph by triangular meshing.

(d) kNN Stanford bunny graph with $k = 8$.

Figure 2.6: Discretisation of continuous manifold: obtaining the point clouds and graphs whose edges created by triangular meshing and kNN method of continuous Stanford bunny manifold.

3 Theory of Graph Signal Processing

Graph signal processing is a field in which classical signal processing methods developed in the Euclidean domain are generalised to non-Euclidean irregular domains such as graphs. To adapt the signal processing concepts to the graph setting, a mathematical description of graph theory is essential. In addition, graph signals need to be defined for further analysis and processing. In this chapter, some real-world examples of graphs and graph signals and the proposed methods for construction are presented with discussions regarding the theoretical aspects. Then, the calculus methods for finite weighted undirected graphs are studied by proposing differential operators and smoothness functionals for graph signals. Based on this, the graph Fourier transform is described using the graph spectral theory. Then, generalizations of useful operators such as convolution, shift, modulation and dilation to graphs are given using the duality relationship between the spectrum and space domain in classical signal processing. The filtering operation is also defined based on graph convolution and basic properties of the graph filtering are discussed. Finally, the theoretical integration of graph signal processing methods, explained in this chapter, to inverse problems on manifolds are given.

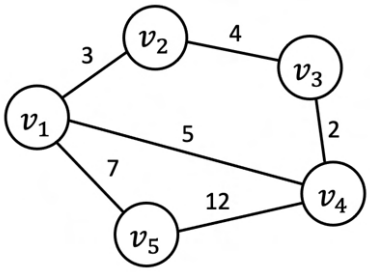
3.1 Graph Theory

A graph is defined as the set of ordered pair of vertices and edges. Let the graph be denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where

- $\mathcal{V}$ is the set of vertices. For the simplicity of the notation, the size of the vertex space is chosen as $|\mathcal{V}| = N$. Therefore, the set of vertices can be denoted as $\mathcal{V} = \{v_i\}_{i=1}^{N}$.
- $\mathcal{E}$ is the set of edges, which are pairs of connected vertices. The dimension of this set is chosen as $|\mathcal{E}| = N_e$. It should be noted that $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.

As explained by Diestel [27], any graph can be encoded by graph weight adjacency matrix, $\mathbf{W} = \{w_{ij}\} \in \mathbb{R}^{N \times N}$. Each element of this matrix, w_{ij} represents the weight of the edge $e_{ij} = (v_i, v_j)$.

Thus, the set of edges can be defined mathematically as $\mathscr{E} = \{(v_i, v_j) | (v_i, v_j) \in \mathscr{V} \times \mathscr{V}, w_{ij} \neq 0\}$. An example for graph weight adjacency matrix can be seen in Figure 3.1. Here, each column and row encodes the specific vertices in the order of $\{v_1, ..., v_5\}$.



(a) Topology of the graph.

$$W = \begin{bmatrix} 0 & 3 & 0 & 5 & 7 \\ 3 & 0 & 4 & 0 & 0 \\ 0 & 4 & 0 & 2 & 0 \\ 5 & 0 & 2 & 0 & 12 \\ 7 & 0 & 0 & 12 & 0 \end{bmatrix}$$

(b) Weight adjacency matrix of the graph.

Figure 3.1: Different representations of a simple undirected graph.

There are different types of graphs, which are listed as below:

- **Directed and Undirected Graphs**: If any edge has a direction, in other words $w_{ij} \neq w_{ji}$, the graph is said to be directed. Otherwise, it's called undirected with the symmetric weight adjacency matrix **W**.
- **Weighted and Unweighted Graphs**: If all of the elements of weight adjacency matrix **W** of the graph is either 0 or 1, the graph is called as unweighted. Otherwise, it is called weighted, where the edge weights can be any nonnegative real number without the symmetry in the matrix.
- **Null Graph**: If there is no edge in the graph, it's called null.
- **Connected and Disconnected Graphs**: If there is a path between any two vertices, the graph is connected. Otherwise, it's disconnected.
- **Complete Graph**: Each pair of the vertices is connected by an edge for the complete graph.
- **Bi-partite Graph**: There are two independent sets which contain the set of vertices, where the vertices of the graph is connected in such a way that each edge has a connection from the first set to the second set.

The graphs can be used for various purposes that require different types of graphs. One of the common applications of graphs are network analysis such as social networks, transportation networks, computer networks, etc. Analysing the connectivity of such networks, identifying the key nodes and the flow of information are a few case studies of this field as explained by Bondy et al [28]. Another application is the data analysis, where clustering, classification and

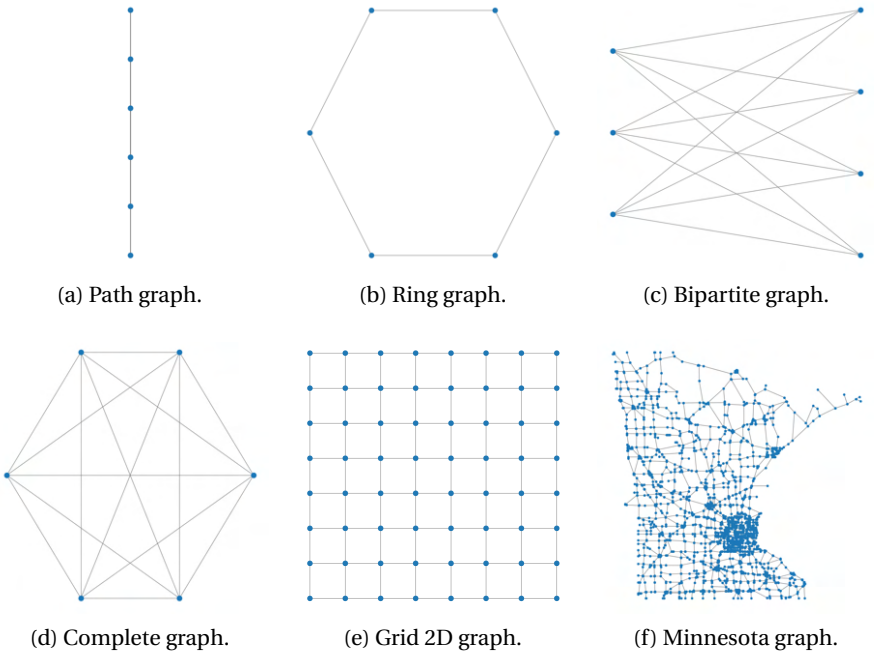
anomaly detection are a few examples. The graphs can also be very useful for recommendation systems, to represent structured knowledge and biological modeling as well. Particularly, the neurons in the brain can be modeled by the graphs very suitably according to Huang et al's article [29]. In this thesis, the graphs are used for extrinsic modeling of the manifolds in a discrete way for solving the inverse problem where the motivation behind it is explained in Section 2.2.4. Here, the finite-weighted undirected graphs are chosen as the most suitable graph type for this problem. For the rest of the thesis, only this type of graphs is studied and analyzed. A very basic example of this graph type can be seen in Figure 3.1. However, more complicated shape for real-world signal can be seen in Figure 3.3d, which is generated from the continuous manifold. In Section 2.2.4, Figure 2.6 also shows another visuals for graph generation of the same problem. It can be seen that there is no direction in the edges, which makes the weight adjacency matrix symmetric. In addition, since there is no self-loop on vertices, the diagonal elements are equal to 0.

Some examples of different types of graph data can be seen in Figure 3.2. In classical signal processing, data domain consists of a set of equidistant instants in time or a set of points in space on a regular grid. This data domain is regular and Euclidean and this structure is called as *path graph* as seen in Figure 3.2a. For the analysis of finite-length signals, *ring graph* is more suitable as the structure. For example, discrete Fourier transform (DFT) in classical signal processing assumes the data domain as a ring which is illustrated in Figure 3.2b. The examples of bipartite and complete graphs are also given in Figure 3.2c and 3.2d, respectively. More sophisticated graph types as grid 2D and Minnesota road graphs are given in Figure 3.2e and 3.2f. Here, grid 2D graph is similar to the 2D signal in 2D Euclidan domain but the edge weights may differ to model the distances between vertices better. On the other hand, Minnesota graph is more like a direct graph representation of the road network and it doesn't have any similarity with any kind of Euclidean spaces.

The graphs are useful mathematical objects to model the real-world data. In order to model the data in an appropriate way, there is a generic graph construction method that consists of three main steps as proposed by Ortega [30]:

1. **Generation of Nodes with Node Coordinates** A graph vertices can be constructed from a set of points. Some of the examples can be listed as the Twitter users being the vertices and the coordinates are the location of the users, the cities or the places as the vertices and the geographical locations of the cities as the coordinates or the neurons in the brain as the vertices and the location of them in the brain region as the coordinates. The second example is an example of pixelisation in 2D manifold that models 3D geometric structure.

2. **Finding the Structures (Edges)** Choice of the edges is essential to model the relation of the data points, vertices to each other. Edges shape the graph structure. Useful technique for that is to connect a vertex to its k-nearest neighbors, which is specifically called as *kNN graph* as proposed by Kang [26]. Otherwise, the edges of any graph can be



(a) Path graph. (b) Ring graph. (c) Bipartite graph.

(d) Complete graph. (e) Grid 2D graph. (f) Minnesota graph.

Figure 3.2: Different graph types.

constructed by connecting a point to points closer than some measure. This measure is mostly related to the distances between two vertices which is dependent on the type of the applications. Following the previous examples, the other knowledge of data can be used here to create the structure. For Twitter example, whether two users are following each other or not can be represented by edges, while for the brain neuron example, the edges can represent the axon between two neurons.

3. **Associate a Weight to Each Edge** Gaussian kernel weighting function can be used to choose the edge weights as stated by Shekkizhar et al's article [31]. Given kNN graph is used, the function can be defined as follows

$$w_{i,j} = \begin{cases} \exp(-\frac{[dist(i,j)]^2}{2\theta^2}), & \text{if } dist(i,j) \leq k \\ 0, & \text{o.w.} \end{cases} \tag{3.1}$$

where θ denotes the standard deviation and k is threshold value for edge creation. The first parameter relates to how the edge weight deviates with respect to distance metrics. Edge weight is inversely proportional to the distances between vertices as it makes sense to have a strong relation for two vertices that are close to each other. The threshold parameter enables the sparse graph weight adjacency matrix, which can be useful for efficient computational purposes. This aspect will be discussed later.

Some examples of the graph construction methods can be seen in Figure 3.3. In Figure 3.3a, the graph construction for the Konigsberg bridge problem is illustrated. It's the problem that

is solved by Leonhard Euler [32] in 1736. The problem was an old puzzle concerning the possibility of finding a path over every one of seven bridges that span a forked river flowing past an island but without crossing any bridge twice. Euler presented the solution that these types of walks are not possible by creating this unweighted graph as follows: the vertices are used to show the landmasses and the edges are used to show the bridges. Figure 3.3b shows the construction of the weighted graph from geographical locations for modeling the sensed measures like the temperature of the cities. Here, the vertices are the cities of Switzerland and the edges are created by Gaussian weight function in Equation 3.1 and distance can be defined as the geographical distance.

Images are also suitable to be modeled by graph data according to the article [33] of Cheung et al and this construction is visible in Figure 3.3c. Here, the vertices are the pixels of the image and the edges are valid for the horizontal, vertical and diagonal neighbors of each pixels. The weights are chosen with the Gaussian method as well and the distance can be chosen as the norm square of the difference of the pixel intensities. Images are type of manifolds that are globally Euclidean. Therefore, this construction can be useful for the problem presented in Section 2.1. More generic graph construction of the manifold that is only Euclidean locally is illustrated in Figure 3.3d. Here, the vertices are chosen as the points on the 2D manifold, that is the surface of 3D geometric object. The edges are chosen by kNN algorithm and edge weights are determined by the Gaussian method. This example is the one that is studied in the thesis. Also, in Section 2.2.4, Figure 2.6 shows another graphs generated by the same principle.

3.2 Graph Signals

To model the signals on the irregular graph data domain, the graph signals are required to be defined. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, the graph signals can be defined by proposing two necessary spaces for such signals, which are built on the vertex space $\mathcal{V}$ and the edge space $\mathcal{E}$.

Space of Real Signals on Vertex Space:

Generic real functions on vertex space can be defined as $f : \mathcal{V} \to \mathbb{R}$. Such signals are called as *graph signals*. As inspired by the article [19] of Bronstein et al, Hilbert space of such signals can be defined as $\mathcal{H}(\mathcal{V}) = \{f : \mathcal{V} \to \mathbb{R}\}$. Since $|\mathcal{V}| = N$, $\mathcal{H}(\mathcal{V}) \cong \mathbb{R}^N$. Based on this definition of Hilbert space, inner product and norm should be defined.

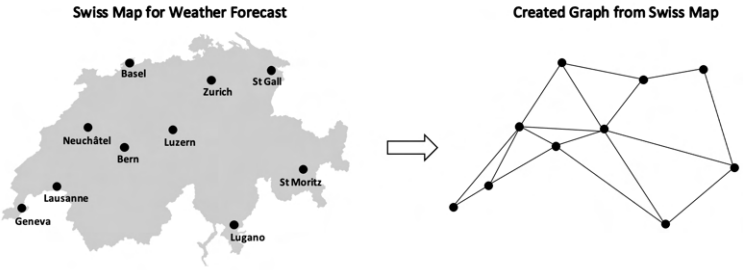
Inner product of $\mathcal{H}(\mathcal{V})$ can be defined as:

$$< f, g >_{\mathcal{H}(\mathcal{V})} = \sum_{i \in \mathcal{V}} f_i g_i, \quad \forall f, g \in \mathcal{H}(\mathcal{V}) \tag{3.2}$$

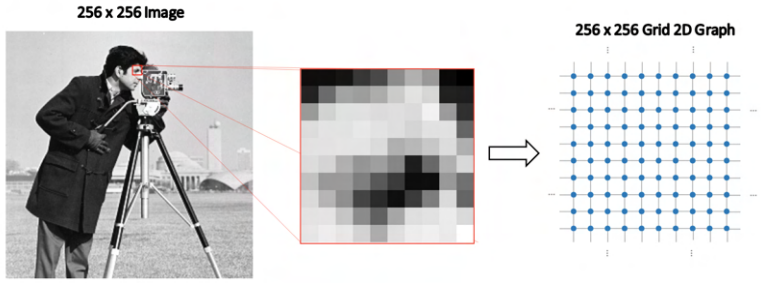
ℓ_p-norm of $\mathcal{H}(\mathcal{V})$ can be defined as

$$||f||_p = \begin{cases} \left(\sum_{i \in \mathcal{V}} |f_i|^p\right)^{1/p}, & 1 \leq p < \infty \\ \max_{i \in \mathcal{V}} |f_i|, & p \to \infty \end{cases} \tag{3.3}$$

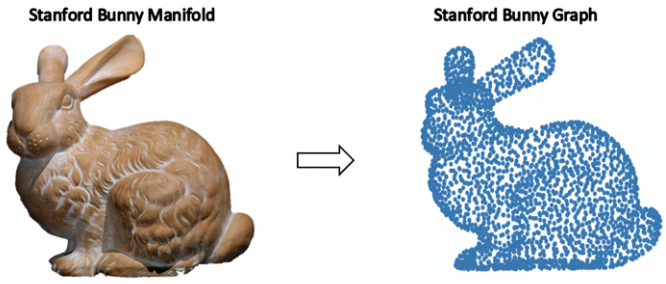
(a) Graph construction for Konigsberg bridge problem.



(b) Graph construction for Swiss map for weather forecast.



(c) Grid 2D graph construction from 2D image.



(d) Graph construction for Stanford bunny manifold.

Figure 3.3: Different graph construction examples.

It must be noted that ℓ_2-norm is induced by the inner product.

Space of Real Signals on Edge Space:

Generic real functions on edge space can be defined as $F : \mathscr{E} \to \mathbb{R}$. Such signals are called as *graph edge signals*. As inspired by the article [19] of Bronstein et al, Hilbert space of such signals can be defined as $\mathscr{H}(\mathscr{V}) = \{f : \mathscr{E} \to \mathbb{R}\}$. Since $|\mathscr{E}| = N_e$, $\mathscr{H}(\mathscr{E}) \cong \mathbb{R}^{N_e}$.

Inner product of $\mathscr{H}(\mathscr{E})$ can be defined as:

$$< F, G >_{\mathscr{H}(\mathscr{E})} = \sum_{(i,j)\in\mathscr{E}} F_{ij} G_{ij}, \quad \forall F, G \in \mathscr{H}(\mathscr{E}) \tag{3.4}$$

ℓ_p-norm of $\mathscr{H}(\mathscr{E})$ can be defined as:

$$||F||_p = \begin{cases} \left(\sum_{(i,j)\in\mathscr{E}} |F_{ij}|^p\right)^{1/p}, & 1 \le p < \infty \\ \max_{(i,j)\in\mathscr{E}} |F_{i,j}|, & p \to \infty \end{cases} \tag{3.5}$$

Similarly to the vertex space, ℓ_2-norm of $\mathscr{H}(\mathscr{E})$ is induced by the inner product. It should be noted that no edge weight information is used in Equations 3.2, 3.3, 3.4, 3.5. There are alternative definitions for inner products and norms of such Hilbert spaces as defined in the article [19] of Bronstein et al where edge weights are included. The main motivation could be to add the intrinsic structure of the model when measuring the distances from the inner products and norms. However, in this thesis, the main motivation of ignoring the edge weight effect here is to follow the Euclidean distances for graph signals, which makes the definition of the signal processing operators in the following sections simpler and more appropriate. In contrast to the article [19] of Bronstein et al, the effect of the intrinsic structure is added in the definition of graph difference operators, which will be seen in the following sections.

(a) Swiss map with temperatures measured for each cities.

(b) Projected temperature signal on 1D regular signal.

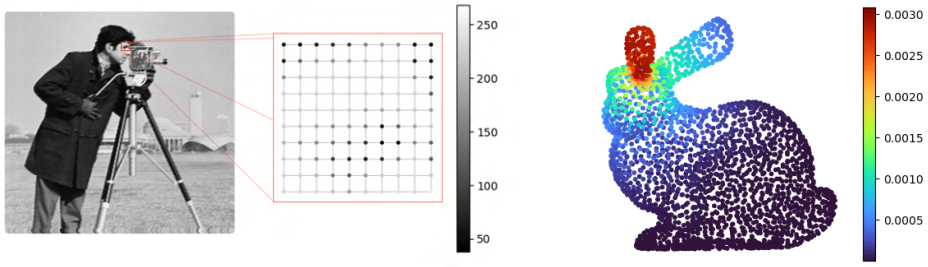
(c) Temperature Swiss map graph signal.

Figure 3.4: Visualization of temperature signals of Swiss cities for classical and graph signal processing.

Examples of Graph Signals

In Figure 3.4, temperature signal for Swiss cities is given as an example of graph signal. In

Figure 3.4b, the signal is finite-length discrete-time signal that is suitable for classical signal processing. However, the map information is lost by this projection, which may result in lack of information very similarly to the tangent plane projection method that ignores the manifold's structure in Section 2.2.4. To include this information, the graph is constructed as shown in Figure 3.3b and the resulted graph signal is seen in Figure 3.4c.



(a) Image grid 2d graph signal with close-up as well. (b) Bunny graph signal.

Figure 3.5: Image grid 2d graph signal and Bunny graph signal examples.

Two examples of graph signals on manifolds are given in Figure 3.5. Graph construction for image is given in Figure 3.3c and the graph signal version of it is seen in Figure 3.5a. For the Stanford bunny manifold, the graph construction is given Figure 3.3d and an example bunny graph signal is seen in Figure 3.5b.

3.3 Calculus for Graph Signals

3.3.1 Graph Differential Operators

Differential operators are generalization of the operation of differentiation in calculus. These operators can be extended to irregular domains such as graphs, which can be useful for some purposes such as defining the spectral domain of graph signals, applying spectral techniques or measuring some sort of smoothness value. The effect of the intrinsic structure of the graph is added by the edge weight inclusion and this is compatible with the definition of Hilbert spaces. In this section, the definition of four main differential operator are given and validated by showing the main characteristics of these operators in classical calculus.

Graph Gradient

As proposed by Shuman et al's article [34], graph gradient $\nabla : \mathcal{H}(V) \rightarrow \mathcal{H}(\mathcal{E})$ is a linear operator that can be defined as

$$(\nabla f)_{ij} = \sqrt{w_{ij}}(f_i - f_j) \tag{3.6}$$

In fact, this kind of gradient operator is called as Jacobian operator as well. It should be noted that for an undirected graph, the output of this operator is anti-symmetric, i.e. $(\nabla f)_{ij} = -(\nabla f)_{ji}$.

Graph Divergence

Graph divergence $\mathrm{div} : \mathscr{H}(\mathscr{E}) \rightarrow \mathscr{H}(\mathscr{V})$ is a linear operator that can be defined as

$$(\mathrm{div}F)_i = \sum_{j \in \mathscr{V}} \sqrt{w_{ij}}(F_{ij} - F_{ji}) \tag{3.7}$$

For anti-symmetric graph edge signals, i.e. $F_{ij} = -F_{ji}$, like the output of the graph gradient, the graph divergence in Equation 3.7 results in $(\mathrm{div}F)_i = \sum_{j \in \mathscr{V}} 2\sqrt{w_{ij}}F_{ij}$.

From the proof in Appendix A.1, the graph divergence is adjoint operator of the graph gradient for the validation of these definitions:

$$< \nabla f, G >_{\mathscr{H}(\mathscr{E})} = < f, \mathrm{div}G >_{\mathscr{H}(\mathscr{V})} \tag{3.8}$$

Graph Laplacian

Graph Laplacian operator is a significant operator that captures the geometry of the manifold and it serves as basis for the Fourier transform and graph convolution. As proposed by Shuman et al's article [34], graph Laplacian $L : \mathscr{H}(\mathscr{V}) \rightarrow \mathscr{H}(\mathscr{V})$ is a linear operator that can be defined as the graph divergence of the graph gradient, $L = \mathrm{div}\nabla$:

$$(Lf)_i = (\mathrm{div}(\nabla f))_i = \sum_{j \in \mathscr{V}} w_{ij}(f_i - f_j) \tag{3.9}$$

As graph Laplacian is a linear operator, the operation 3.9 can be represented by matrix vector multiplication, in which the Laplacian matrix **L** can be defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{3.10}$$

Here, **W** is weight adjacency matrix that is defined in Section 3.1 and **D** is the degree matrix, which is a diagonal matrix whose i^{th} diagonal element is $d_i = \sum_{j \in \mathscr{V}} w_{ij}$. The analysis of this Laplacian matrix **L** offers a potential definition of spectrum domain, which will be studied in Section 3.4.

From the adjointness relation of the graph gradient and graph divergence, it can easily be seen that graph Laplacian is self-adjoint operator.

The graph Laplacian in formula 3.9 is described as *combinatorial*, which is only one type of

generalised graph Laplacian operators. According to definition given by Biyikoğlu et al [35], a generalized Laplacian is any symmetric matrix whose i, j^{th} entry is negative if there is an edge between connected vertices v_i, v_j, equal to 0 if $v_i \neq v_j$ and v_i is not connected to v_j. Another type of generalised graph Laplacian is *normalized graph Laplacian* and its matrix $\tilde{\mathbf{L}}$, which can be defined as

$$\tilde{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2} \tag{3.11}$$

This is also equivalent to the following definition

$$(\tilde{L}f)_i = \frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{V}} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right) \tag{3.12}$$

Theoretical comparison of combinatorial and normalized graph Laplacian is studied in Section 3.4 as the main difference of these operators occur in the eigenvectors and eigenvalues of the matrices. From Burago's article [36], the combinatorial graph Laplacian is a graph discretization of Laplace-Beltrami operator, which allows us to perform spectral method of signal processing on manifolds via graphs. Therefore, in this thesis, the combinatorial graph Laplacian is chosen to be the standard graph Laplacian for the rest of the sections.

Graph Hessian

Graph Hessian $H : \mathscr{H}(\mathcal{V}) \rightarrow \mathbb{R}^{N \times N \times N}$ is a linear operator that can be defined as

$$(Hf)_{ijk} = \frac{w_{ij}}{2}(f_i - f_j) + \frac{w_{ik}}{2}(f_i - f_k) \tag{3.13}$$

This operation is the graph discretized version of second order differential Hessian operator described in classical calculus. From the proof in Appendix A.2, graph Hessian is the trace of graph Laplacian. This is a validation of the definition of graph Hessian.

All in all, the graph differential operators are defined and validated by the natural properties in the classical calculus. On top of that, graph smoothness functions are proposed for the rest of this section.

3.3.2 Graph Signal Smoothness

Measuring the graph signal smoothness is a critical objective for linear inverse problems on manifolds in particular for the regularization term. For this purpose, a few graph signal measurements are listed from the studies of Shuman et al [34].

Local Variation

Obtaining the local information could be useful for many applications as the smoothness of the graph signals may differ from some regions of the manifold to another. Graph gradient is a convenient approximation of difference operation, thus the local information can be obtained by taking the norm of the graph gradient outputs through the neighbors of the specific vertex. Therefore, as proposed in the Shuman et al's article [34], the local variation of a graph signal f at vertex v_i can be defined as

$$\begin{aligned} ||\nabla_i f||_2 = ||(\nabla f)_{i.}||_2 &= \left(\sum_{j:(i,j)\in\mathscr{E}} (\nabla f)_{ij}^2 \right)^{\frac{1}{2}} \\ &= \left(\sum_{j:(i,j)\in\mathscr{E}} (w_{ij}(f_i - f_j))^2 \right)^{\frac{1}{2}} \end{aligned} \tag{3.14}$$

Global Variation

Global smoothness value is needed as well, and it can generated from the local variation. For that purpose, discrete p-Dirichlet form of graph signal f can be useful for providing the generalised global smoothness measurement as stated by Shuman et al in their article [34].

$$S_p(f) = \frac{1}{p} \sum_{i\in\mathscr{V}} ||\nabla_i f||_2^p = \frac{1}{p} \sum_{i\in\mathscr{V}} \left(\sum_{j\in\mathscr{V}} w_{ij}(f_i - f_j)^2 \right)^{p/2} \tag{3.15}$$

- $p = 1$: Total variation of the graph signal

$$S_1(f) = \sum_{i\in\mathscr{V}} ||\nabla_i f||_2 \tag{3.16}$$

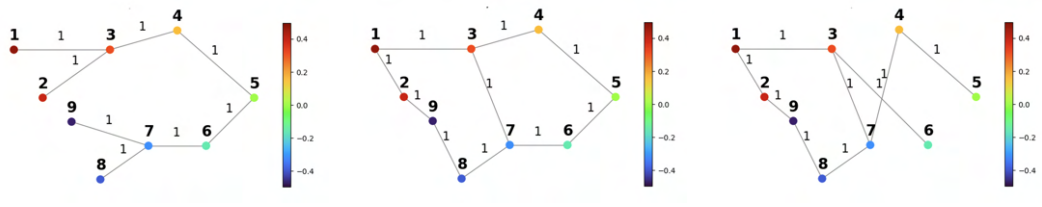
- $p = 2$: Graph Laplacian quadratic form

$$S_2(f) = \sum_{(i,j)\in\mathscr{E}} w_{ij}(f_i - f_j)^2 = \mathbf{f}^T\mathbf{L}\mathbf{f} \tag{3.17}$$

  We can also define the semi-norm: $||\mathbf{f}||_L = ||\mathbf{L}^{1/2}\mathbf{f}||_2 = \sqrt{\mathbf{f}^T\mathbf{L}\mathbf{f}} = \sqrt{S_2(f)}$. An important observation is that this value is nonnegative and inversely proportional to the smoothness of the graph signal. This value takes the minimum value, 0, when the graph signal is constant, i.e. it's maximally smooth.

Example - Smoothness Comparison:

In Figure 3.6, three different graph structures can be observed with the same signal values and vertices. The only differences are due to the edges. For the graph signals, the smoothness is affected by the intrinsic structure of the graph. From Table 3.1, the expected smoothness measurements are obtained that the least value is obtained for the smoothest graph.



(a) Smoother graph. (b) Moderate smooth graph. (c) Less smooth graph.

Figure 3.6: Graphs with the same signal values and vertices with different edges.

Table 3.1: Graph smoothness comparison table.

Graph Type	$S_1(f)$	$\mathbf{f}^T\mathbf{L}\mathbf{f}$
Fig. 3.6a	1.04	0.16
Fig. 3.6b	2.30	1.28
Fig. 3.6c	2.59	1.65

3.4 Graph Spectral Theory

In this section, the spectral domain for the graph signals are defined based on the graph differential operators defined in Section 3.3.1, similar to that in classical signal processing in classical calculus. The similarity of the definitions can be seen in the derivation in classical signal processing presented in Appendix B.2. The eigen-decomposition of the graph Laplacian operator is critical for defining the graph spectral basis of the graph signals.

3.4.1 Eigen-Decomposition of Graph Laplacian

From the definition of combinatorial graph Laplacian in 3.9, the matrix of graph Laplacian $\mathbf{L}$ in 3.10 can be investigated. First of all, this matrix is positive semi-definite from the fact that $\mathbf{f}^T\mathbf{L}\mathbf{f} \geq 0,\ \forall f \in \mathscr{H}(\mathscr{V}) \approx \mathbb{R}^N$ as stated in [37].

Being the positive semi-definite, the matrix $\mathbf{L}$ can be decomposed as follows

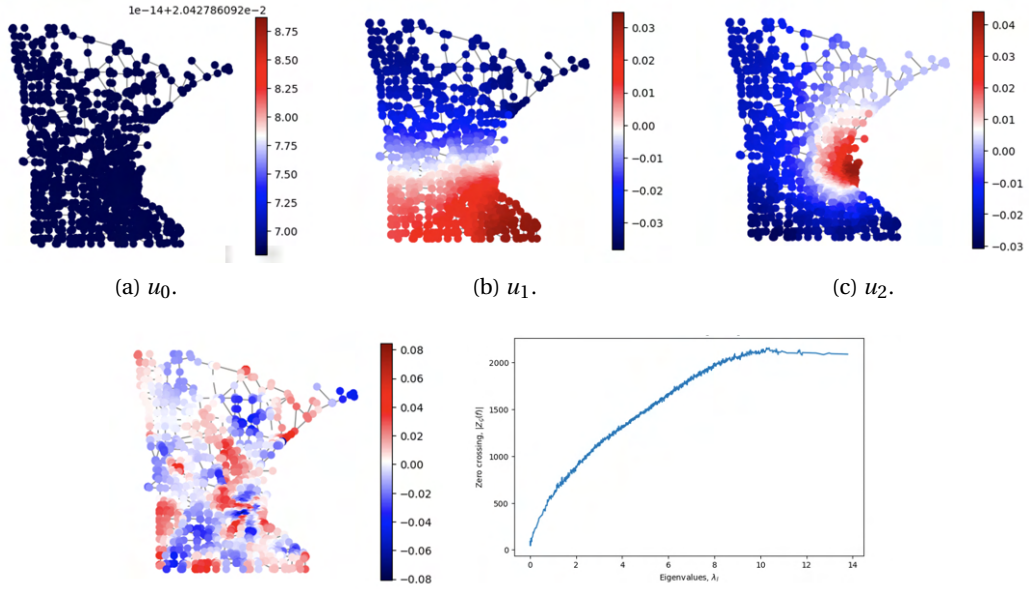
$$\mathbf{L} = \begin{bmatrix} | & \dots & | \\ u_0 & \dots & u_{N-1} \\ | & \dots & | \end{bmatrix} \begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix} \begin{bmatrix} - & u_0 & - \\ \vdots & \vdots & \vdots \\ - & u_{N-1} & - \end{bmatrix} = \mathbf{U}\Lambda\mathbf{U}^T \tag{3.18}$$

In 3.18, $\{u_l\}_{i=0}^{N-1}$ and $\{\lambda_l\}_{i=0}^{N-1}$ are eigenvectors and eigenvalues of the matrix $\mathbf{L}$, i.e. $Lu_l = \lambda_l u_l$. Without loss of generality, the eigenvalues can be sorted as $0 = \lambda_0 \leq \dots \leq \lambda_{N-1} = \lambda_{max}$ and eigenvectors are sorted according to the eigenvalues as well.

Very important observation is that eigenvalues have the notion of frequency. As the eigenvalues decrease, the associated eigenvectors vary more slowly across the graph, i.e. if two vertices are

connected by an edge with a large weight, the values of the eigenvector at those locations are likely to be similar. This phenomena can be seen in the Figure 3.7 where the eigenvectors for different eigenvalues are seen. Zero-crossing is also defined to measure another smoothness: $\mathcal{Z}_{\mathcal{G}}(f) = \{e = (i,j) \in \mathcal{E} : f_i f_j < 0\}$. In the results, it can be seen that the larger eigenvalue we have, the more rapidly the eigenvector changes. In fact, for eigenvalue of 0, the corresponded eigenvector is a constant vector, $u_0 = \frac{\mathbf{1}}{\sqrt{N}}$ as seen also in Figure 3.7a. Also, the zero-crossing versus eigenvalues in Figure 3.7e validates the expected behavior of frequency.

For the normalized graph Laplacian, the matrix is still positive semi-definite. Thus, the eigen-decomposition can be applied for this matrix as well. As a result, different eigenvectors and eigenvalues are obtained. The main difference is the range of the eigenvalues. The maximum eigenvalue is fixed and equal to 2, thus the range is $\lambda_i \in [0,2]$. One difference is that eigenvector corresponded to zero eigenvalue is not constant. However, besides these differences there is not a major difference with the use of combinatorial and normalized (CITE SMT) one. Therefore, since it's more simpler and takes less time to compute which will be mostly discussed in Section 4, the combinatorial graph Laplacian is used for rest of the sections.



(a) u_0. (b) u_1. (c) u_2.

(d) u_{50}. (e) Zero-crossing values of eigenvectors versus corresponded eigenvalues.

Figure 3.7: Eigenvectors of the combinatorial graph Laplacian corresponded to eigenvalues λ_0, λ_1, λ_2 and λ_{50} and zero crossing values for different number of eigenvalues.

For graph representing the discretized version of a continuous manifold, the eigenvectors and eigenvalues provide insights into the geometry of the underling manifold. Eigenvectors relate to the components of the graph at which it resonates. On the other hand, eigenvalues give the

information about the intrinsic properties of the manifold like smoothness or curvature.

3.4.2 Graph Fourier Transform

As stated by Chung [38], the eigenvectors of graph Laplacian form an orthogonal basis for graph spectral domain. Similarly to steps to define the continuous time Fourier transform (CTFT) shown in Appendix B.2.2, the Fourier transform for graph signals can be defined as follows

$$\hat{f}(\lambda_l) =< f, u_l >= \sum_{i=1}^{N} f_i u_l(i) \tag{3.19}$$

This can be interpreted as the projection of the graph signal into the orthogonal Fourier basis. This helps to inspect the signal in graph spectral domain. Set of eigenvalues of graph Laplacian $\{\lambda_l\}$ is called as spectrum, which is the range of the graph spectral domain.

One observation about Fourier transform is that for zero eigenvalue the graph spectrum is the DC component of the graph signal: $\hat{f}(0) = \sum_{i=1}^{N} f_i u_0(i) = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} f_i$.

Another property that can be observed is that if the impulse function in vertex domain is defined, then its graph Fourier transform would be the eigenfunction:

$$\delta_k(i) = \begin{cases} 1, & \text{if } i = k \\ 0, & \text{o.w.} \end{cases} \quad \leftrightarrow \quad \hat{\delta}_k(\lambda_l) = u_l(k) \tag{3.20}$$

Similarly to classical signal processing shown in Appendix B.2.2, the inverse graph Fourier transform can be defined as

$$f_i = \sum_{l=0}^{N-1} \hat{f}(\lambda_l) u_l(i) \tag{3.21}$$

Matrix vector multiplication forms of the formulas 3.19 and 3.21 are $\hat{\mathbf{f}} = \mathbf{U}^T \mathbf{f}$ and $\mathbf{f} = \mathbf{U}\hat{\mathbf{f}}$.

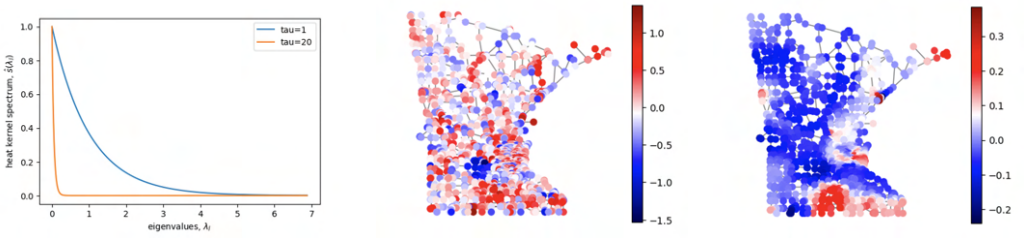
Any graph signal can be either represented in graph vertex domain or graph spectral domain. Such signals are called as *kernels* in general as stated in the article [34] of Shuman et al. If the one would like to design low-pass, high-pass or band-pass filter characteristic, the kernel can be designed in graph spectral domain and then the inverse graph Fourier transform formula in 3.21 can be applied to obtain the kernel in space domain as proposed by Shuman et al [34].

An Example - Heat Kernel: The heat kernel in graph spectral domain is the exponential filter with rate τ, which is as follows

$$\hat{f}(\lambda_l) = e^{-\tau \lambda_l} \tag{3.22}$$

This kernel is the solution of the diffusion problem which is discussed in Appendix C.1 and it

has low-pass filter characteristic. From Figure 3.8b, this characteristic can easily be seen in the graph spectral domain. By using the inverse graph Fourier transform, the one can obtain the kernel in vertex domain as well. Figure 3.8c and 3.8a shows that the one with larger diffusion rate results in smoother signal as the cutoff value in the low-pass is much lower.



(a) Heat kernels of $\tau = 1$ and $\tau = 20$ in graph spectral domain.

(b) Heat kernel with $\tau = 1$ in vertex domain.

(c) Heat kernel with $\tau = 20$ in vertex domain.

Figure 3.8: Graphs of heat kernels for rates $\tau = 1$ and $\tau = 20$ in graph spectral and vertex domain.

There are also another graph kernels for different purposes as well. The one that has low-pass filter characteristic which would solve Tikhonov problem is presented in Appendix C.2. Another graph kernel can be defined by having an idealized high pass filter spectrum. This kernel can be used for edge detection in graph signals, where the examples can be seen in C.3.

3.5 Generalized Operators for Graph Signals

Fundamental operations of signals like convolution, filtering, shifting, dilation and modulation in classical signal processing offer a variety of methods that can be used for the processing of the signals. Therefore, to use this method for the inverse problem for manifolds in the graph setting, these operations need to be generalized to the graph signals.

3.5.1 Convolution

As given by Oppenheim [39], the convolution in classical signal processing for discrete-time signals is defined as:

$$(f * h)_n = \sum_{k=0}^{N-1} f_k h_{n-k} \tag{3.23}$$

In Fourier domain, the convolution results in the multiplication operator:

$$\widehat{(f * h)}_k = \hat{f}_k . \hat{h}_k \tag{3.24}$$

The shifting of the graph signal, h_{n-k} is not easy to do so. This is because of the irregular

shape of the data domain. There is not one consistent direction for shifting operation as in discrete-time signals, where the data is defined on a regular path graph, which has only one direction for the shift. Therefore as proposed by Shuman et al [34], we can define the convolution of graph signals as the multiplication in graph spectral domain:

$$\widehat{(f * h)}(\lambda_l) = \hat{f}(\lambda_l).\hat{h}(\lambda_l) \tag{3.25}$$

Taking the inverse graph Fourier transform, the convolution operator becomes as follows:

$$(f * h)_i = \sum_{l=0}^{N-1} \hat{f}(\lambda_l)\hat{h}(\lambda_l)u_l(i) \tag{3.26}$$

3.5.2 Filtering

In classical signal processing, the filtering of the linear inverse system are equivalent to the convolution operator. In Fourier domain, it's equivalent to multiplication operator. For the graph setting, the filtering can be defined as the graph convolution operator. This type of filtering design can be called as *spectral filtering*. Let us denote the kernel as g and $\hat{g}$ in vertex and graph spectral domain, respectively. From Equation 3.26, the spectral filtering becomes as follows

$$\mathbf{y} = \mathbf{U} \underbrace{\begin{bmatrix} \hat{g}(\lambda_0) & & 0 \\ & \ddots & \\ 0 & & \hat{g}(\lambda_{N-1}) \end{bmatrix}}_{\hat{g}(\Lambda)} \mathbf{U}^T\mathbf{f} = \hat{g}(\mathbf{L})\mathbf{f} \tag{3.27}$$

It should be noted that this definition of filtering results in non-localized operation in vertex domain as stated by Defferrard et al's article [40]. In order to extract local information, the one can also define the filtering in vertex domain for filters that are assumed to be polynomial linear operator with the order of K as proposed by Defferrard et al [40]. In this type of filters, the kernel is represented as

$$\hat{g}(\lambda_l) = \sum_{k=0}^{K} \theta_k \lambda_l^k \implies \hat{g}(\mathbf{L}) = \sum_{k=0}^{K} \theta_k \mathbf{L}^k \tag{3.28}$$

Then, the filtering becomes as follows

$$\begin{aligned} y_i &= \sum_{l=0}^{N-1} \hat{f}(\lambda_l)\hat{h}(\lambda_l)u_l(i) \\ &= \sum_{j=1}^{N}\sum_{l=0}^{N-1} f(j)u_l^*(j)\sum_{k=0}^{K}\theta_k\lambda_l^k u_l(i) \\ &= \sum_{j=1}^{N} f(j)\sum_{k=0}^{K}\theta_k(\mathbf{L}^k)_{ij} \end{aligned} \tag{3.29}$$

Here, $(\mathbf{L}^k)_{ij}$ is nonzero if and only if there exists a path from vertex v_i to vertex v_j with length k. Thus, given $d_{\mathcal{G}}(i,j)$ is the geodesic distance, i.e. the shortest path length from vertex v_i to vertex v_j, we have the following lemma:

$$d_{\mathcal{G}}(i,j) > k \implies (\mathscr{L}^k)_{ij} = 0 \tag{3.30}$$

From the Lemma 3.30, the filtering formula results in the following formula:

$$y_i = \sum_{j\in\mathcal{N}(i,K)\cup\{i\}} b_{ij}f_{in}(j) \tag{3.31}$$

where $b_{ij} = \sum_{k=d_{\mathcal{G}}(i,j)}^{K}\theta_k(\mathbf{L}^k)_{ij}$ and $\mathcal{N}(i,K)$ is the neighborhood of vertex v_i with path length less than k. This equation 3.31 shows that the graph filtering is K-localized transform with the coefficients b_{ij}. This property can be very useful for graph machine learning applications, in particular for graph convolution network, where the filterbanks consists of different polynomial filters and these extract the local features from the input, which is the main objective to interfere useful information.

3.5.3 Shifting

From Oppenheim [39], shifting in classical signal processing for continuous-time signals is defined as

$$(\mathcal{T}_k f)(t) = f(t-k) \tag{3.32}$$

It can be also seen as the convolution with the filter $\delta(t-k)$. In Fourier domain, it's the multiplication with eigenfunction corresponding to eigenvalue, i.e. the frequency $2\pi k$:

$$\widehat{(\mathcal{T}_l f)}_k = \hat{f}(w).e^{jkw} \tag{3.33}$$

The shifting operation for graph signals is not easy to do so because of the irregular structure of the data domain as explained before. With the use of spectral equivalence, the one can generalize it as the graph convolution with impulse signal as proposed by Shuman et al [34].

$$(\mathcal{T}_k f)_i = \sqrt{N}(f * \delta_k)_i \tag{3.34}$$

where the impulse graph signal is defined as

$$\delta_k(i) \begin{cases} 1, & \text{if } i = k \\ 0, & \text{o.w.} \end{cases} \tag{3.35}$$

In graph spectral domain, the impulse graph signal is $\hat{\delta}_k(\lambda_l) = u_l(k)$. Then, the shifting operator can be written as

$$(\mathscr{T}_k f)_i = \sqrt{N} \sum_{l=0}^{N-1} \hat{f}(\lambda_l) u_l(k) u_l(i) \tag{3.36}$$

$\sqrt{N}$ term makes the conversation of the mean of the graph signals: $\sum_{i \in \mathscr{V}} (\mathscr{T}_k f)_i = \sum_{i \in \mathscr{V}} f_i$ and its proof is given in Appendix A.4.

3.5.4 Modulation

From Oppenheim [39], modulation in classical signal processing for continuous-time signals is defined as:

$$(\mathscr{M}_{w_0} f)(t) = e^{j w_0 t} f(t) \tag{3.37}$$

where in Fourier domain, it becomes

$$\widehat{(\mathscr{M}_{w_0} f)}(w) = \hat{f}(w - w_0) \tag{3.38}$$

It can be also seen as the convolution with $\delta(w - w_0)$, i.e. shifting operation in Fourier domain. As proposed by Shuman et al [34], the way to generalize the modulation to the graph setting is to define it as the multiplication with eigenfunction in vertex domain:

$$(\mathscr{M}_k f)_i = \sqrt{N} f_i u_k(i) \tag{3.39}$$

It should be noted that this generalized modulation is not a translation in the graph spectral domain due to the irregular nature of the spectrum.

3.5.5 Dilation

Dilation in classical signal processing for continuous-time signals is defined as

$$(\mathscr{D}_s f)(t) = \frac{1}{s} f(\frac{1}{s}) \tag{3.40}$$

where in Fourier domain, it becomes

$$\widehat{(\mathscr{D}_s f)}(w) = \hat{f}(sw) \tag{3.41}$$

Dilation operation cannot be directly generalized to the graph setting. Scaling in graph spectral domain can be used to generalize this operation:

$$\widehat{(\mathscr{D}_s f)}(\lambda_l) = \hat{f}(s\lambda_l) \tag{3.42}$$

It should be noted that the generalized dilation requires the kernel '$\hat{f}(.)$ to be defined on the entire line, not just on $[0, \lambda_{max}]$.

3.6 Integration of GSP Methods to Inverse Problems on Manifolds

After discussing the theoretical aspects of graph signal processing (GSP), the integration of the described methods to the problems presented in Section 2.2.4 need to be done. The main proposal of this thesis is to use graph signal modeling to manifolds and solve the linear inverse problems by graph signal processing techniques. In this section, this is done only in a theoretical way.

Defining the graph difference and convolution operators, we can update the problem given in Figure 2.2. Let us first define the graph obtained by triangular meshing of continuous manifold as $\mathscr{G}$ and the graph signal as $\mathbf{f} \in \mathscr{H}(V) \cong \mathbb{R}^N$. Then, the model presented in Equation 2.5 can be reformulated as follows

$$\mathbf{Y} = \mathbf{G}_{\mathscr{G}} f + \mathbf{n} \tag{3.43}$$

where $\mathbf{G}_{\mathscr{G}} = \hat{g}(\mathbf{L})$ is the graph convolution operator as defined in Equation 3.27. In addition, the graph gradient denoted as ∇ is chosen as a difference operator for graph signals as given in Equation 3.6. Then, the problem given in Equation 2.2 can be reformulated for graph signals, where are two examples are listed.

Generalised LASSO Problem for Graph Signals:

The problem presented in Equation 2.8 can be directly updated to graph signals with the use of graph convolution operator $\mathbf{G}_{\mathscr{G}}$, graph gradient operator ∇. As a result, the problem becomes as follows

$$\arg \min_{\mathbf{f} \in \mathbb{R}^N \cong \mathscr{H}(V)} ||\mathbf{y} - \mathbf{G}\mathbf{f}||_2^2 + \lambda ||\nabla \mathbf{f}||_1 \tag{3.44}$$

Here, ℓ_1-norm in regularisation term is the one defined for graph edge signals in Equation 3.5.

Tiknonov Problem for Graph Signals:

The problem presented in Equation 2.9 can be similarly updated to graph signals as well. Here, the smoothness functional to be used is Graph Laplacian quadratic form $S_2(f)$ given in

Equation 3.17. As a result, the problem becomes as follows

$$\arg \min_{\mathbf{f} \in \mathbb{R}^N \cong \mathscr{H}(\mathscr{V})} ||\mathbf{y} - \mathbf{G}\mathbf{f}||_2^2 + \lambda S_2(\mathbf{f}) \tag{3.45}$$

These problems can be solved by Primal Dual Splitting algorithm presented in Section 2.1 Algorithm 2. The computational aspects of the operators defined in this chapter are needed to be analysed in detail so that the implementation of such solution could be possible. In the next chapter, this issue is handled by proposing the development of a Python package Pycsou-GSP with computational analysis of graph signal processing methods.

4 Computational Aspects of Graph Signal Processing

Graph signal processing methods provide powerful solutions for various applications including the linear inverse imaging problem on manifolds as proposed in Section 2.2.4. In Section 3.6, the integration of such methods are formulated and the requirements of the implementation of graph differential and convolution operators are emphasized. To address real-world problems, there are many numerical challenges in the implementation of the graph signal processing methods. This chapter firstly discusses these challenges. Then, the computational analysis and the most efficient implementations of graph differential and convolution operators are selected. The choices are verified by numerical experiments. Finally, the development of a new Python package called **Pycsou-GSP**, designed as an extension of Pycsou, which is a software framework for advanced computational imaging [41], for graph signal processing is described. Here, the selected implementations and advanced features are presented with some examples of using the package.

4.1 Challenges of Implementing GSP Methods

For the implementation of the graph signal processing methods in the computer environment, the speed, memory/storage and correctness of the algorithms should be considered carefully. The framework is designed in Python environment which mostly uses Numpy package for the arrays [42]. However, Cupy for GPU usage and Dask.array can also be used for arrays [43], [44]. To implement the linear inverse problems on manifolds with graph signal processing methods, the numerical challenges of GSP methods in such environment should be discussed and solved for providing efficient and correct solutions. These challenges are listed below.

- Graphs can be very large in real-world problems. In particular, the number of vertices denoted as N could be large for many applications. This also results in very large matrices of $\mathbb{R}^{N\times N}$ like weight adjacency matrix, Laplacian and differential matrices. Therefore, the dense representation of such matrices could be very inefficient for usage of memory.

- In practice, the number of edges of the graph denoted as N_e is much less than the square of number of vertices: $N_e << N^2$. This results in sparse weight adjacency matrix. Therefore, the graphs can be encoded by any sparse matrix format as proposed by Scott et al [45]. However, the conversion and the optimal choice of the sparse formats also differ for different usages of this matrix. At least, we can say that the coordinate list format of sparse matrix can be used for storage as it's the most natural way to create the matrix. It can be defined either in SciPy.sparse as *coo_matrix* [46] or sparse package as *COO* and in case of the usage of GPU, and Cupy, this matrix can be created by Cupyx.sparse package as *coo_matrix*.

- Python is an interpreted language, which is slower than any other compiled languages. This is because the source code of Python program is converted into bytecode, which is then translated to machine code for the execution as explained in [47]. This operation is done for each run of Python program. Therefore, even the sparse matrix multiplication could be slow. Numba package offers the translation of Python functions to the optimized machine codes at runtime using the LLVM compiled library as stated in [48]. This operation is called as Jit (just-in time) compilation and the jit-compiled numerical algorithms can approach the speeds of compiled languages such as C as stated in [48]. However, the vectorized or matrix form solutions are not optimal for Numba jit compilation. Instead, simple matrix free solutions are favored [48]. This should also be applied to graph differential operators.

- Graph edge signals could be problematic to be stored as they could be very large in case they are stored as dense 2D arrays. For example, the output of graph gradient would be very large for huge N.

- To solve the optimization problem in the linear inverse problems for manifolds formulated in Section 3.6, the packages having proximal algorithms efficiently implemented could be used. For this purpose, a Python package called **Pycsou** with version 2 [41] is chosen to be used as it offers various functions for APGD and PDS algorithms with different types as well. In order to implement novel operators or algorithms in Pycsou, they should be created as a subclass of one of the abstract classes in Pycsou's class hierarchy as stated in [41]. For example, *Pycsou.abc.LinOp* could be used for graph differential operators.

- Using Pycsou would also create a few set of rules as presented in [41]. First of all, the operators should handle the case where the input array is a 1-D or N-D array. Another rule to be followed is to enforce the precision agnosticity. The numerical precision of the inputs and outputs of the operators should be kept constant with the use of decorator in Pycsou named as *Pycsou.runtime.enforce_precision*.

- The module agnosticity is another rule of Pycsou, where the inputs and outputs of any operator should be of the same array module as stated in [41]. The array modules can be either Numpy [42], Dask.array or Cupy. Here, Cupy is supported in case of GPU usage,

which might accelarate the computation time [43], while Dask.array handles large arrays by cutting up into many small arrays, which allows the computation of arrays larger than memory using all cores [44]. This can outperform the other packages in terms of speed. Therefore, this aspect should also be considered.

4.2 Proposed Implementations in GSP Methods

In this section, the computational aspects of graph differential and convolution operators are investigated and the optimal solutions for each operator in computer environment are chosen.

4.2.1 General Strategy

As explained in the previous section, the matrix-free solutions are desired to be proposed for operators such that Numba jit-compilation can be used for faster implementations. The verification of the solutions for graph differential operators are done by numerical experiments, where this solution is compared with alternative sparse matrix solutions. There is a Python package called Pygsp, that implements the operators without considering Pycsou environment and implementing the functions by sparse matrix vector multiplication mostly as it can be seen in [49]. Thus, the matrix-free jit-compiled solutions are compared with this technique.

First of all, the optimal choice of Numba [48] parameters are done by experimentation. Just one of the graph differential operators are used for this and the fastest combination of parameters are chosen for Numba jit-compilation of rest of the methods. Here, the first run of Numba function is not included because of just analyzing the compiled version. Then, numerical experiments for comparing all of the others are done for each operators. These experiements are executed for four different types of graph, which are Ring, Grid 2D, Comet and SwissRoll graphs, which have different sparsities. Grid 2D here could be more reliable as they have similar structure to the graphs obtained by triangular meshing of manifolds. The performance time is measured for different number of nodes for different methods. Here, the creation of matrices and first call of the functions and Numba jit-compiled functions are not included for the time measurements. This is because the iterative solution is the most part of the time consumption and the preprocessing part is not trying to be optimized. In addition, the first call of Numba funcitons is performing the Jit-compilation, which takes longer, and that is why it is not used in benchmarking. At the end, the fastest method in this experimentation is chosen to be used for the implementation.

Especially, for matrix-free solution, even sparse matrix format should be simplified to directly obtain the necessary arrays as inputs to function for Numba jit-compilation. A new format called row, column, data (RCD) format is to be proposed. This is applied to the graph weight adjacency matrix W and as an output, $(wrow, wcol, wdata)$ is obtained. Here, $wrow$ represents the row index array of nonzero elements, while $wcol$ is the counterpart for the columns.

The corresponding data is stored in $wdata$. As a result, all of these three arrays have N_e the number of elements in case the graph is undirected. This operation is depicted in Figure 4.1.

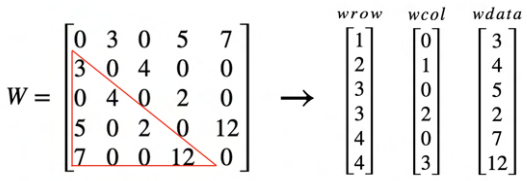


Figure 4.1: An example of RCD format.

Graph edge signals are in space congruent to $\mathbb{R}^{N_e}$ as stated by Bronstein et al [19], thus they can be encoded by 1D array with N_e elements. To identify the elements with edges, the order of this array should be the same as the order of RCD format of W. For this reason, this RCD format should be stored inside the graph data.

In addition, the matrix-free solution strategy is used for implementation of graph convolution operators as Laplacian matrix could be very large. Eigen-decomposition of such matrix could be very expensive and matrix vector multiplication could cost a lot as indicated by Defferrard et al [40]. Therefore, some form of approximation method is trying to be proposed for this operator. The approximation method is also tested by comparing the function in real number with different number of orders.

4.2.2 Graph Differential Operators

In this part, graph gradient, divergence and Laplacian with combinatorial and normalized operators are trying to be implemented efficiently.

Graph Gradient

Graph gradient is defined in Section 3.3.1 by Equation 3.6. Here, the output is indexed by two indices which denote the edges but we can look at these indices where each of them uniquely denoted by another index k. Then, three methods are proposed for implementation of this operator.

- **Sparse Dot Solution:** This method is the one that is given in the package of Pygsp [49]. Here, the differential matrix $\mathbf{D}$ is computed for once and stored as "csc sparse" matrix. This type is more efficient for sparse dot operation [46]. Then, an input signal is given to the transpose of differential matrix $\mathbf{D}^T$. For the experimentation, the creation of differential matrix is not included. The implementation of this method can be seen as follows

```
1 import pygsp
```

```
# G: Graph class, f: input array
G.compute_differential_operator()
# G.D is computed, with csc_matrix
y = G.D.T.dot(f) # === G.grad(f)
```

Listing 4.1: Codes for implementation of sparse dot solution of graph gradient.

- **Vectorized Solution in RCD Format:** From the basic relation, this operation can be solved by vectorized operation as follows

```
# f: input array
# wrow, wcol, wdata: RCD format of G.W
y = (wdata**0.5) * (f[wrow] - f[wcol])
```

Listing 4.2: Codes for implementation of vectorized solution in RCD format of graph gradient.

- **Jit Compiled Matrix-Free Solution:** Using RCD format, the matrix-free solution without vectorization can be implemented as follows

```
import numba as nb
@nb.njit(fastmath=True, parallel=False, nogil=True, cache=False)
def compute_grad(f, wrow, wcol, wdata, res):
    for i in range(len(wrow)):
        res[i] = (wdata[i]**0.5) * (f[wrow[i]] - f[wcol[i]])

# f: input array, res: zero array with same shape as output
# wrow, wcol, wdata: RCD format of G.W
y = compute_grad(f, wrow, wcol, wdata, res)
```

Listing 4.3: Codes for implementation of Numba jit-compiled matrix-free solution of graph gradient.

The experimentation for choices of Numba arguments are done firstly. Table 4.1 shows different choices of arguments for each type. Figure 4.2 demonstrates that the optimal combination in terms of speed is Type 7, where its description can be seen in Table 4.1. Here, the details of the arguments in optimal solution can be seen in [48].

Table 4.1: Different choices of Numba arguments, which are nopython, parallel, fastmath, nogil and cache.

Type No.	nopython	parallel	fastmath	nogil	cache
1	False	False	False	False	False
2	True	False	False	False	False
3	False	True	False	False	False
4	False	False	True	False	False
5	False	False	False	True	False
6	False	False	False	False	True
7	**True**	**False**	**True**	**True**	**False**

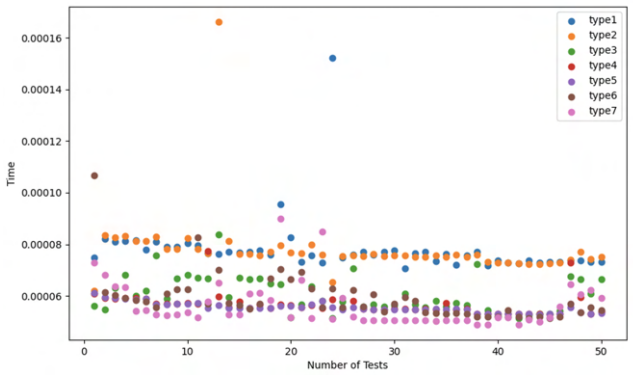
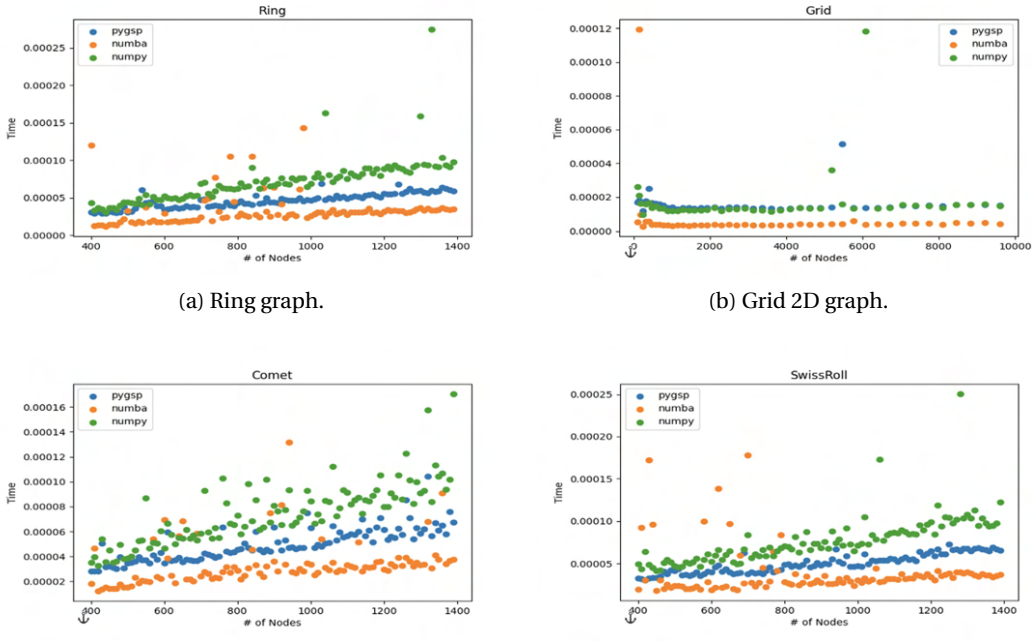


Figure 4.2: Performance time comparison of different Numba arguments for matrix-free jit-compiled solution of graph gradient.

In Figure 4.3, the performance time comparison of three methods are given as described before. From the results, it can be seen that the fastest solution is the matrix-free jit-compiled solution. Therefore, this method is used for the implementation.



(a) Ring graph.

(b) Grid 2D graph.

(c) Comet graph.

(d) SwissRoll graph.

Figure 4.3: Performance time comparison of Numba jit-compiled gradient with sparse dot implementation for four different graph types.

Graph Divergence

Graph divergence is defined in Section 3.3.1 by Equation 3.7. Here, the input is indexed by two indices and this is represented by 1D array similarly to the output of graph gradient. Two methods are proposed for implementation of this operator.

- **Sparse Dot Solution:** This method is the one that is given in the package of Pygsp [49]. Similarly to graph gradient, the differential matrix **D** is computed and stored. Then, an input signal is given to the differential matrix **D**. The implementation of this method can be seen as follows

```
import pygsp
# G: Graph class, f: input array
G.compute_differential_operator()
# G.D is computed, with csc_matrix
y = G.D.dot(f) # === G.div(f)
```

Listing 4.4: Codes for implementation of sparse dot solution of graph gradient.

- **Jit Compiled Matrix-Free Solution:** Using RCD format, the matrix-free solution can be implemented as follows

```
import numba as nb
@nb.njit(fastmath=True, parallel=False, nogil=True, cache=False)
def compute_div(f, wrow, wcol, wdata, res):
    for i in range(len(wrow)):
        res[wrow[i]] += (wdata[i]**0.5) * f[i]
        res[wcol[i]] -= (wdata[i]**0.5) * f[i]

# f: input array, res: zero array with same shape as output
# wrow, wcol, wdata: RCD format of G.W
y = compute_div(f, wrow, wcol, wdata, res)
```

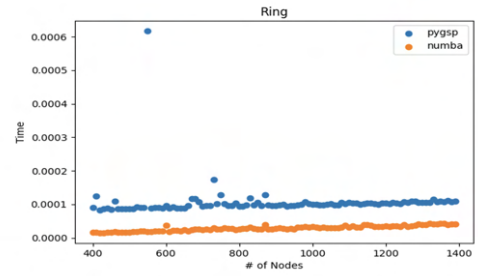
Listing 4.5: Codes for implementation of Numba jit-compiled matrix-free solution of graph divergence.

In Figure 4.4, the performance time comparison of two methods are given as described before. From the results, the fastest solution is the matrix-free jit-compiled solution again, which is therefore used for the implementation.

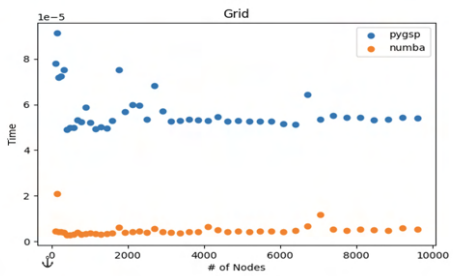
Graph Laplacian

Graph Laplacian operators of combinatorial and normalized are defined in Section 3.3.1 by Equation 3.9 and 3.12. Two methods are proposed for implementation of this operator.

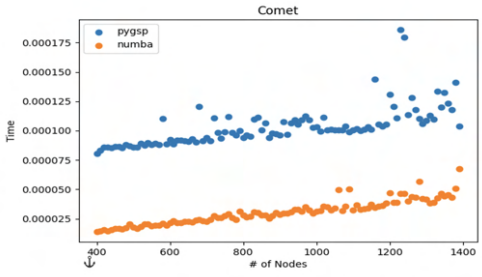
- **Sparse Dot Solution:** This method is the one that is given in the package of Pygsp [49]. Here, the Laplacian matrix **L** is computed for once by using Equation 3.10 and 3.11 and stored as “csc sparse“ matrix. Then, an input signal is given to this matrix. For the



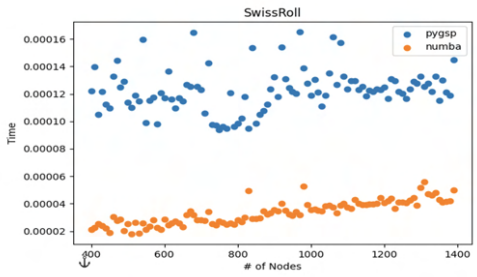
(a) Ring graph.



(b) Grid 2D graph.



(c) Comet graph.



(d) SwissRoll graph.

Figure 4.4: Performance time comparison of Numba jit-compiled divergence with sparse dot implementation for four different graph types.

experimentation, the creation of Laplacian matrix is not included. The implementation of this method can be seen as follows

```
import pygsp
# G: Graph class, lap_type: Laplacian type, f: input array
G.compute_laplacian(lap_type=lap_type)
# G.L is computed, with csc_matrix
y = G.L.dot(f)
```

Listing 4.6: Codes for implementation of sparse dot solution of graph Laplacian for types combinatorial or normalized according to input data “lap_type“.

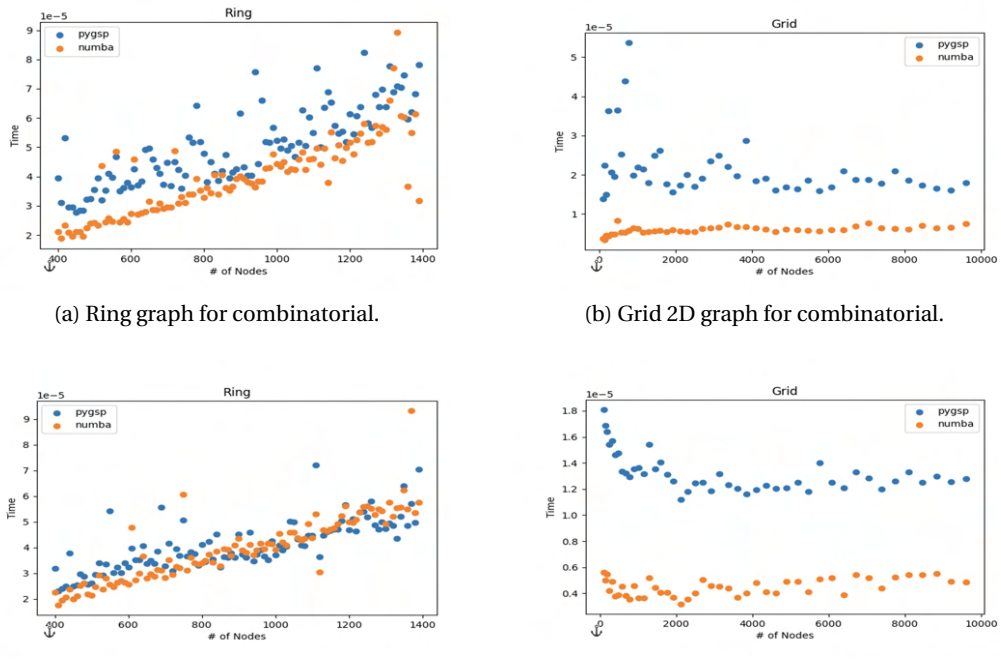
- **Jit Compiled Matrix-Free Solution:** Using RCD format, the matrix-free solution can be implemented as follows

```
import numba as nb
@nb.njit(fastmath=True, parallel=False, nogil=True, cache=False)
def compute_lap(f, wrow, wcol, wdata, asqrt, res):
    for i in range(len(wrow)):
        res[wrow[i]] += wdata[i] * (f[wrow[i]]/asqrt[wrow[i]] - f[wcol[i]]/
    asqrt[wcol[i]]/asqrt[wrow[i]])

# lap_type: either "combinatorial" or "normalized"
# xp: python module interpreted from input array f
if lap_type == "combinatorial":
    asqrt = xp.ones(G.N, dtype=wdata.dtype)
elif lap_type == "normalized":
```

```
    asqrt = xp.sqrt(xp.squeeze(xp.asarray(G.W.sum(0))))

# f: input array, res: zero array with same shape as output
# wrow, wcol, wdata: RCD format of G.W
y = compute_lap(f, wrow, wcol, wdata, res)
```

Listing 4.7: Codes for implementation of Numba jit-compiled matrix-free solution of graph Laplacian for types combinatorial or normalized according to input data “lap_type“.

In Figure 4.5, the performance time comparison of two methods are given for combinatorial and normalized Laplacian types but for two types of graph, which are Ring and Grid 2D. From the results, the fastest solution is the matrix-free jit-compiled solution again, which is therefore used for the implementation.



(a) Ring graph for combinatorial.

(b) Grid 2D graph for combinatorial.

(c) Ring graph for normalized.

(d) Grid 2D graph for normalized.

Figure 4.5: Performance time comparison of Numba jit-compiled Laplacian of combinatorial and normalized with sparse dot implementation for two different graph types.

4.2.3 Graph Filtering Operators

In Section 3.5.2, graph filtering operator defined as spectral filtering is formulated by Equation 3.27. Given that the number of vertices of the graph is N, the computational complexity of the operation is $O(N^2)$ as stated by Defferrard et al’s article [40], and it requires the storage of $N \times N$ dense matrix for $\hat{g}(\mathbf{L})$. In real-world problems, this might create memory overload and computationally expensive implementation. To decrease this computational cost, the filtering

operator $\hat{g}(\mathbf{L})$ is designed to be implemented without creating any matrix. For this purpose, the polynomial filter approximation is applied to the kernel as proposed by Shuman et al's article [50]. This idea is also motivated by the localized transform property of polynomial filter operators as explained in Section 3.5.2.

$$\hat{g}(\lambda_l) \approx \sum_{k=0}^{K} \theta_k \lambda_l^k \tag{4.1}$$

Here, the spectrum of the filter is approximated by K-order polynomial operator. As proposed by Shuman et al's article [50], Chebyshev polynomials $T_k : [-1,1] \rightarrow [-1,1]$ can be used for efficient implementation because of the recursive relation by its definition

$$T_k(x) = \begin{cases} 1, & k = 0 \\ x, & k = 1 \\ 2xT_{k-1}(x) - T_{k-2}(x), & k \geq 2 \end{cases} \tag{4.2}$$

These polynomials form an orthogonal basis for $\mathscr{L}^2([-1,1], dx/\sqrt{1-x^2})$ as stated by Shuman et al's article [50]. Thus, the representation of functions in this space is given as follows:

$$\hat{g}(x) = \frac{1}{2}c_0 + \sum_{k=1}^{\infty} c_k T_k(x) \tag{4.3}$$

where $\{c_k\}_{k=0,1,2,\ldots}$ is a sequence of Chebyshev coefficients.

In graph filtering, the input range is $[0, \lambda_{max}]$. Shifting and scaling can be applied to Chebyshev polynomials to approximate the kernel $\hat{g}(\lambda_l)$ via the transformation: $\frac{\lambda_{max}}{2}(x+1)$ as presented by Defferrard et al's article [40]. Then, the kernel can be represented as follows:

$$\hat{g}(x) = \frac{1}{2}c_0 + \sum_{k=1}^{\infty} c_k \overline{T}_k(x), \ \ \forall x \in [0, \lambda_{max}] \tag{4.4}$$

where $\overline{T}_k(x) = T_k(x/\alpha - 1)$, $\alpha = \frac{\lambda_{max}}{2}$ and

$$c_k = \frac{2}{\pi}\int_0^{\pi} \cos(k\theta)\hat{g}(\alpha(\cos\theta + 1))d\theta \tag{4.5}$$

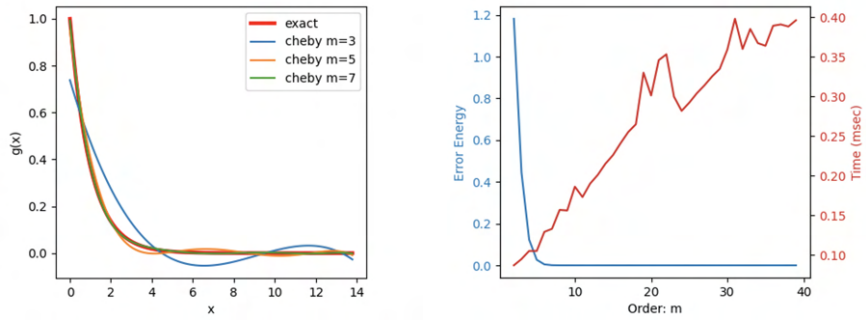
Then, as indicated in Shuman et al's article [50], the recurrence relation of the scaled Chebyshev polynomials becomes

$$\overline{T}_k(x) = \begin{cases} 1 & k = 0 \\ x/\alpha - 1 & k = 1 \\ \frac{2}{\alpha}(x-\alpha)\overline{T}_{k-1}(x) - \overline{T}_{k-2}(x) & k \geq 2 \end{cases} \tag{4.6}$$

Here, this is not the approximation but the representation of the kernel in another basis. However, since there is an infinite sum, the finite order can be chosen as K, which makes

it an approximation. In computer environment, this is done by sampling of the operators. The Chebyshev coefficients can be computed from the input kernel spectrum defined for the continuous variable as described in Equation 4.5. Regularly spaced sampling is applied here for the computation.

This is done for the kernel spectrum defined for the continuous input variable. For example, for the implementation of heat kernel, the input can be a Python function, which means that it can take any value as an input. Figure 4.6 shows that when the input is defined for continuous domain, error energy converges to zero as the number of order increases. However, the performed time also increases; therefore, the order should be chosen neither very large nor very low. $K = 20$ is chosen as a standard value.



(a) Chebyshev approximation of heat kernel spectrum $\hat{g}(\lambda_l) = e^{-\lambda_l}$ for order $m = 3, 5, 7$.

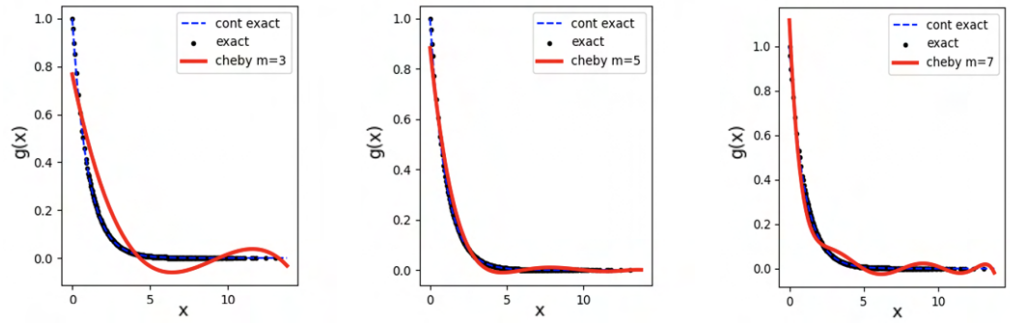
(b) Error energy and computational time versus order of chebyshev approximation.

Figure 4.6: Chebyshev approximation of heat kernel spectrum $\hat{g}(\lambda_l) = e^{-\lambda_l}$ with performance analysis for different orders.

However, the input can also be valid for discrete input value, only by the sampled points of the kernel spectrum on the eigenvalues $\lambda_{l=0,1,\ldots,N-1}$. These eigenvalues are not regularly spaced as stated in [38]. In this framework, the input cannot be a Python function for this approximation but 1D array with N elements. The same techniques are applied but the discretization is projected to regularly spaced values for continuous value solution. In Figure 4.7, the approximation can be seen. As the order increases, the approximation gets better until some value. For very large number of order, the deformation occurs as seen in 4.7c. This is because of the lack of continuous domain. Therefore, overfitting problem occurs for such cases. For this type of inputs, the standard choice of order is chosen to be $K = 6$.

As presented in Shuman et al's article [50], the approximation can be extended for graph filtering operation by taking input Laplacian matrix $\mathbf{L} \in \mathbb{R}^{N \times N}$, which results in the following formula

$$g(\mathbf{L}) \approx \frac{1}{2} c_0 \mathbf{I} + \sum_{k=1}^{K} c_k \overline{T}_k(\mathbf{L}) \tag{4.7}$$



(a) Chebyshev approximation for order $m = 3$.

(b) Chebyshev approximation for order $m = 5$.

(c) Chebyshev approximation for order $m = 7$.

Figure 4.7: Chebyshev approximation of discrete sampled heat kernel spectrum $\hat{g}(\lambda_l) = e^{-\lambda_l}$ for order $m = 3, 5, 7$.

where

$$\overline{T}_k(\mathbf{L}) = \begin{cases} \mathbf{I} & k = 0 \\ \mathbf{L}/\alpha - \mathbf{I} & k = 1 \\ \frac{2}{\alpha}(\mathbf{L} - \alpha\mathbf{I})\overline{T}_{k-1}(\mathbf{L}) - \overline{T}_{k-2}(\mathbf{L}) \\ k \geq 2 \end{cases} \tag{4.8}$$

Then, this formula is applied for the arbitrary graph signals. As a result, from the recurrence relation, the computational cost of the implementation becomes $O(KN)$, which is much less than $O(N^2)$ for very large number of N. The output of this technique is compared by implementing the exact formula and it is verified that it gives correct results.

4.3 Development of Pycsou-GSP

As discussed in Section 4.1, Pycsou framework has many useful features for solving the optimization problems [41]. In this thesis, an extension of Pycsou version 2 for graph signal processing is decided to be developed. This new package is called as **Pycsou-GSP**, where the source code can be found here [51] and the main advantage of this package is the efficient usage of graph signal processing methods with built-in proximal algorithms in Pycsou [41]. It also has the module and precision agnosticity to the input signal which makes it flexible and robust. The developed structure, few examples of codes and useful properties of the developed Pycsou-GSP are explained in this section.

4.3.1 Overall Structure

Overall structure of Pycsou-GSP can be seen in Figure 4.8. There are three main modules of Pycsou-GSP: Core, Operator and Util. In Core, there is a Graph module where the base

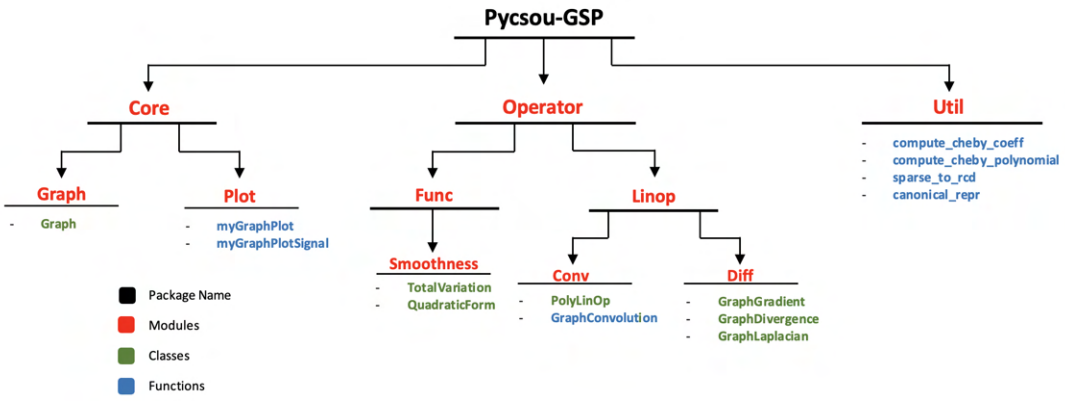


Figure 4.8: Structure of Pycsou-GSP.

class of Graph is implemented. Here, the features of the graph are deeply inspired by the Pygsp.graphs.Graph class [49]. Main differences of Graph base class of Pycsou-GSP is that it doesn't store the huge matrices like Laplacian and differential matrices even in case they are desired to be computed. They also don't have so many attributes like grad, div, compute_differential_operator, compute_fourier_basis and compute_laplacian that exist in Pygsp [49]. Similarly, the graph constructor requires the weight adjacency matrix as an input. This matrix is stored in "coo" sparse matrix format, but the RCD conversion is also applied. The motivation to store the RCD format is to be able to directly decode the graph edge signal like the 1D array output of gradient operator. The sample example of creation of graphs can be seen in the following.

```
from pycgsp.graph import Graph
import numpy as np

N = 30
W = np.random.uniform(size=(N,N))
W[W < 0.95] = 0 # Sparse graph
W = W + W.T     # Undirected graph = symmetric matrix
np.fill_diagonal(W, 0)  # no self-loop
G = Graph(W)        # <- Pycgsp Graph creation from dense matrix W
```

Listing 4.8: Sample codes for the creation of Pycsou-GSP graph object.

Plot module provides two functions: myGraphPlot and myGraphPlotSignal. The former is for the plot of graph object while the latter showes the signal values on vertices to visualize the graph signals. This plot function is deeply inspired by the plot functions of Pygsp [49], while there are differences where Pycsou-GSP plots have features like adding vertex names, showing the edge values, setting the axis off, updating the title etc.

Util module provides utility functions that can be used for miscellaneous functionalities. One is for the RCD format conversion: canonical_repr. This particular function is applied inside the graph differential operators and constructor of graph class. The simple usage of this function can be seen in the following code.

```
from pycgsp.util import pycgspu

# G: created graph object, G.W: weight matrix
wrow, wcol, wdata = pycgspu.canonical_repr(G.W)
```

Listing 4.9: Sample codes for conversion to RCD format.

Another utility functions are related to the chebyshev approximation, which will be explained in the next subsection. Here, compute_cheby_coeff is to obtain the chebyshev coefficients while compute_cheby_polynomial is to convert the chebyshev coefficients to monomial polynomial coefficient. The details will be explained in the next section. Both of these functions are used inside the GraphConvolution function.

One of the core module of Pycsou-GSP is Operator as it provides the implementation of graph differential, graph convolution operators by the modules of Linop.Diff and Linop.Conv respectively. Operator also has the module of Func.Smoothness, where the simple Graph smoothness measurements defined in Section 3.3.2 are implemented. However, matrix-free Numba jit-compiled solutions are chosen for graph differential operators in Section 4.2.2 and efficient Chebyshev approximation solution for graph convolution is formulated in Section 4.2.3

4.3.2 Integration of Differential and Convolution Methods

One of the most important aspects of the integration is that the base class of the operator must be Pycsou.abc.LinOp such that they can be used as inputs to proximal algorithms in Pycsou. However, creating a new subclass of these operators, module and precision agnosticity and the implementation of apply and adjoint operators are compulsory [41].

For graph differential operators, matrix-free Numba jit-compiled functions are to be called inside the apply operator of the generated class. The example of the integration of graph gradient to Pycsou environment can be seen in the following code.

```
import pycsou.abc as pyca
import pycsou.util as pycu

# compute_grad is defined as in Listing 4.3
# compute_div is defined as in Listing 4.5

class GraphGradient(pyca.LinOp)
    def __init__(self, W: typ.Union[pyct.NDArray, pyct.SparseArray]):
        self._wrow, self._wcol, self._wdata = pycgspu.canonical_repr(W)

        self._N = W.shape[0]
        self._Ne = len(self._wdata)
        super().__init__(shape=(self._Ne, self._N))

    @pycrt.enforce_precision(i="arr")
    def apply(self, arr: pyct.NDArray) -> pyct.NDArray:
        return compute_grad(arr, self._wdata, self._wrow, self._wcol)

    @pycrt.enforce_precision(i="arr")
```

```
    def adjoint(self, arr: pyct.NDArray) -> pyct.NDArray:
        # Adjoint of Graph Gradient is Graph Divergence
        xp = pycu.get_array_module(arr)
        res = xp.zeros(self._N, dtype=self._wdata.dtype)
        return compute_div(arr, self._wdata, self._wrow, self._wcol, res)
```

Listing 4.10: Example code for integration of matrix-free jit-compiled solutions to GraphGradient as Pycsou.abc.LinOp.

Here the implementations of GraphDivergence and GraphLaplacian is also similar, where the base class of GraphLaplacian can be chosen as Pycsou.abc.SelfAdjointOp from its self-adjoint property. Then, only implementation of the apply operator is adequate [41].

For graph convolution method, the chosen approach is very different. Because of the locality property of the polynomial linear operator, the class for this type of operator is defined, called as PolyLinOp. The implementation of this function is as follows.

```
import pycsou.abc as pyca
import pycsou.util.ptype as pyct

# LinOp is typically Graph Laplacian operator whose subclass is Pycsou.abc.LinOp

class _PolyLinOp(pyca.LinOp):
    def __init__(self, LinOp: pyca.LinOp, coeffs: pyct.NDArray):

        self._LinOp = LinOp
        self._coeffs = coeffs
        self._N = len(coeffs)
        super(_PolyLinOp, self).__init__(shape=LinOp.shape)

    @pycrt.enforce_precision(i="arr")
    def apply(self, arr: pyct.NDArray) -> pyct.NDArray:
        y = self._coeffs[0] * arr
        z = arr
        for i in range(1, self._N):
            z = self._LinOp(z)      # Recursive update to decrease the
    computational cost
            y += self._coeffs[i] * z
        return y


    @pycrt.enforce_precision(i="arr")
    def adjoint(self, arr: pyct.NDArray) -> pyct.NDArray:
        z = arr
        y = self._coeffs[0] * arr
        for i in range(1, self._N):
            z = self._LinOp.adjoint(z)  # Recursive update to decrease the
    computational cost
            y += self._coeffs[i] * z
        return y
```

Listing 4.11: Example code for conversion to RCD format.

The implementation of this function is very simple and efficient because of the recursive update for the power of graph Laplacian as Pycsou.abc.LinOp. However, this operator requires the inputs as monomial polynomial coefficients which are $\{\theta_k\}$ as given in Equation 4.1.

GraphConvolution is a function to be implemented with two types of methods that it accepts. The first one is the “polylinop“, which directly expects the monomial polynomial coefficient and graph Laplacian as inputs. The other method is called as “chebyshev“, which

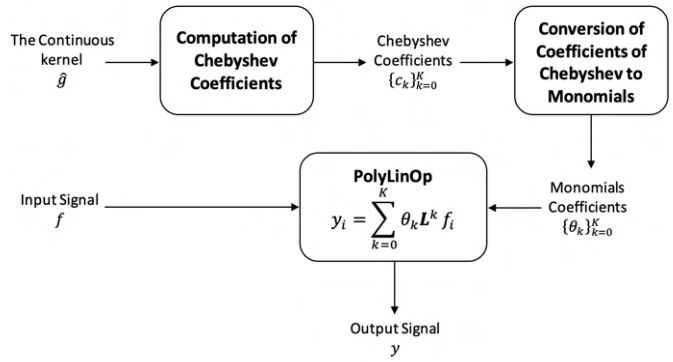


Figure 4.9: Block diagram of the algorithm presented in GraphConvolution operator with method of chebyshev.

expects a Python function that describes the kernel spectrum $\hat{g}(\lambda_l)$ as given in Equation 3.27. In Section 4.2.3, an algorithm to solve the graph filtering output is proposed by firstly computing the Chebyshev coefficients and then applying the chebyshev polynomial operators by using the recursive relation. Instead, the simpler approach is proposed. The algorithm is depicted in Figure 4.9. Firstly, the Chebyshev coefficients are computed by the function Pycgsp.util.compute_cheby_coeff. Here, Also, the maximum eigenvalue should also be computed for the scaling of the chebyshev polynomials. This can be computed by the “lipschitz“ method of Pycsou.abc.LinOp with argument of tight as “bound“ [41]. Then, these coefficients are converted to monomial basis coefficients with the use of function Pycgsp.util.compute_cheby_polynomial. Then, PolyLinOp object is created inside the function and this object is returned to be used as a Pycsou.abc.LinOp operator.

4.3.3 Features, Usages, Limitations and Future Works

Features

The prominent features of Pycsou-GSP can be listed as below.

- The operators are module and precision agnostic, which means the the arraymodules sparse arraymodules and precisions of inputs, outputs and backend implementation are being constant.
- Pycsou-GSP provides fast computational speed and low memory overload.
- Differential and convolution operators are defined as Pycsou.abc.LinOp, which is suitable for proximal algorithms in Pycsou.

Usages and Verifications

Example usages of graph differential and convolution operators developed in Pycsou-GSP are given in the following code. Also, the testing of the solution is done by comparing with Pygsp solution.

```
import pygsp
from pycgsp.core.graph import Graph
import pycgsp.operator.linop.diff as pycgspd
import pycgsp.operator.linop.conv as pycgspc

G = pygsp.graphs.Minnesota()
pycgsp_G = Graph(W=G.W)

f = np.random.randn(pycgsp_G.N)      # input array
g = lambda x: np.exp(-x)             # heat kernel function

G.compute_differential_operator()
y1 = G.grad(f)
GGrad = pycgspd.GraphGradient(W = pycgsp_G.W)
y2 = GGrad(f)
print("Passed GraphGradient Test? ->", np.allclose(y1, y2))

y1 = G.div(f)
GDiv = pycgspd.GraphDivergence(W = pycgsp_G.W)
y2 = GDiv(f)
print("Passed GraphDivergence Test? ->", np.allclose(y1, y2))

G.compute_laplacian()
y1 = G.L.dot(f)
GLap = pycgspd.GraphLaplacian(W = pycgsp_G.W)
y2 = GLap(f)
print("Passed GraphLaplacian Test? ->", np.allclose(y1, y2))

filt = pygsp.filter.Filter(G=G)
y1 = filt.filter(f)
GConv = pycgspc.GraphConvolution(L = GLap)
y2 = GConv(f)
print("Passed GraphConvolution Test? ->", np.allclose(y1, y2))
```

Listing 4.12: Sample code for usages of Pysou-GSP differential and convolution operators with testing by comparison with Pygsp solution as well.

Output of this code is as follows.

```
Passed GraphGradient Test -> True
Passed GraphDivergence Test -> True
Passed GraphLaplacian Test -> True
Passed GraphConvolution Test -> True
```

Listing 4.13: Output of the code in Listing 4.12.

As a result, all the outputs are correct. They can be used for real-world applications as well, as it is done in the next section.

Limitations and Future Works

- Graph base class developed in Pycsou-GSP is very basic graph class. This could be more sophisticated including some features of graph signal processing.
- Computation of Fourier basis matrices, which can be obtained by eigen-decomposition of the Laplacian matrix. This operation is computationally expensive and there are functions for both dense and sparse matrices particularly in Scipy and Cupyx.scipy packages. A careful implementation of this eigen-decomposition could be applied by considering the module agnosticity of the Pycsou framework.
- Graph Hessian is not implemented even though it is described in Section 3.3.1 by Equation 3.13. The output of Graph Hessian is in the space of three dimension, which results in very large matrices with $\mathbb{R}^{N\times N\times N}$. The efficient coding of this matrix should be studied as a future work.
- All of the methods described and implemented in this thesis and package Pycsou-GSP are compatible for undirected graphs. However, there are also directed graphs, which requires broader perspective on the techniques. This is left as a future work.

5 Application in Astronomy

In this chapter, an application of linear inverse problem on manifolds are studied in the field of astronomy. The selected application seeks to recreate the input celestial signals from the output measured data. First of all, the problem definition of this application is presented by fitting the formulation presented in Section 2.2.4 and 3.6. Also, 2D manifold for spherical surface is described for the celestial signals. Then, suitable graph construction method for such manifolds are described both theoretically and computationally. The implementation of the two proposed methods, which are tangent plane projection method and graph-based method, are tested with synthetic celestial signal created for the application. The results are compared and discussed. Finally, graph-based solution is chosen to be tested for real-world signal as it provides more robust solution. The discussion and the conclusion of this result are presented in the end of this chapter.

5.1 Problem Definition

The selected application in astronomy aims to reconstruct celestial signals from sensor data gathered by optical telescopes on the surface of the Earth. The recorded data is a convolution of the celestial objects in the sky with the point spread function (PSF) of the telescope. The focus is on resolving the inverse problem of deconvolving the observed data in order to restore the original astronomical signal because this procedure creates blurring and distortion. Here, we can use the same symbols as in Section 2.1 Equation 2.3 to designate the recorded sensor data as $\mathbf{y}$, the original celestial signal as f, and PSF function as H operator.

The output data is obtained from an abstract sphere that has a large radius and has the same center concentric to Earth's center. It is called as celestial sphere as given by Kaler in his book [52], and this is a useful construct for describing locations of objects in the sky. The celestial objects can be conceived as being projected upon the inner surface of this sphere, which is observed from the measurements on Earth's surface as presented by Smith [53]. The structure of celestial sphere can be seen in Figure 5.1.

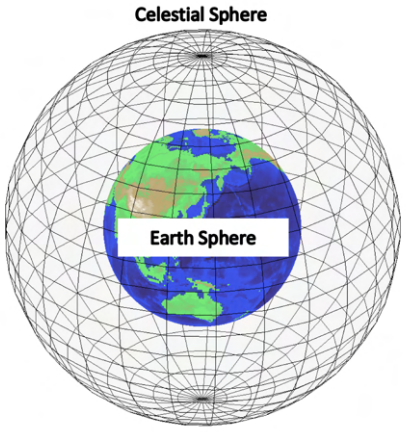


Figure 5.1: Celestial and Earth spheres.

In this application, the manifold is the 2D surface of celestial sphere. The output and input signals are defined on this manifold. Discretization of the problem is compulsory and the main proposed solution of this thesis can be applied, which is the extrinsic modeling of this manifold by graphs as explained in Section 2.2.4. A graph construction method should be proposed for this type of spherical surface manifold. In the next section, this method is explained in detail.

After creating the graph signal, the solution of inverse problems for graph signals should be obtained by following the formulation presented in Section 3.6. By using Pycsou [41] and the developed package Pycsou-GSP [51] explained in Section 4.3, graph-based optimization for generalized LASSO given in 3.44 can be solved computationally. The details will be explained in the next sections.

The implementation of the proposed graph-based solution is desired to be compared with the traditional method of tangent plane projection. For this purpose, a synthetic data is proposed to have a clear output for the comparison. The proposed synthetic data is obtained by adding of the random circles to the sky inside the field of the view of 20 degrees from the sphere center. The orthographic and gnomonic view of such synthetic signal can be seen in Figure 5.2. Here, gnomonic view looks visually better as the data is inside a small part of the sphere. The projection to 2D Euclidean space is achieved with the use of direction cosine grids. The details of the implementation of this projection will be presented in the following sections. Pycsou is directy to be used for the solution of discrete signal in Euclidean space. Generalised LASSO problem formulated in Section 2.2.4 Equation 2.8 can be applied with the proximal algorithm of PDS as proposed in Section 2.2.4 Algorithm 2. At the end, the estimated signal is interpolated to spherical surface again, which results in additional noise as well.

Before experimentation, the output signal is prepared in computer environment for tests. A point spread function (PSF) is chosen and applied by graph convolution for the graph solution and classical convolution. After estimating the input signals for both of the proposed

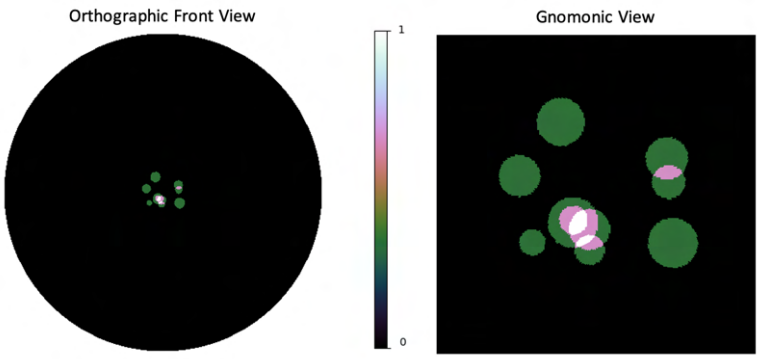


Figure 5.2: Orthographic front view and gnomonic view with $(0,90)$ degree rotation of synthetic signal on celestial manifold.

solutions, the comparison is done by analyzing the estimated signals at gnomonic view. Extra measurements like minimum square error are also provided for the detailed comparison. As a result, better solution is expected for the proposed graph-based solution.

Because of having more robust solution for inverse problem on manifold, The implementation of the solution of inverse problems for graph signals should be tested with real-world input signals as well. For that purpose, the data is chosen to be Hydrogen-alpha ions intensity over the warm ionized sky measured by the *Wisconsin H-Alpha Mapper (WHAM)* telescope, where the details of the data can be found in [54]. The orthographic view of this signal on the described manifold can be seen in Figure 5.3.

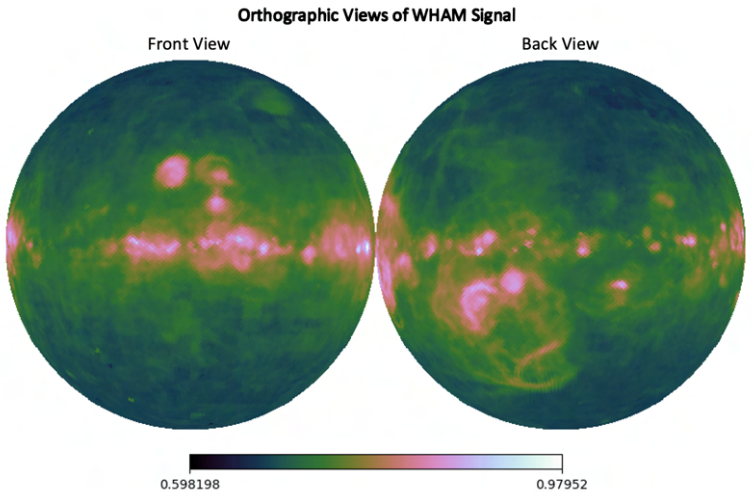


Figure 5.3: Orthographic view from front and back of Hydrogen-alpha ions intensity signal on celestial manifold measured by Wisconsin H-Alpha Mapper (WHAM) telescope.

All in all, two proposed methods are updated for inverse problems of celestial signals, which are the signals defined on celestial sphere. The complete proposed methods can be seen in

Figure 5.4. All the experimentation is applied by following this framework.

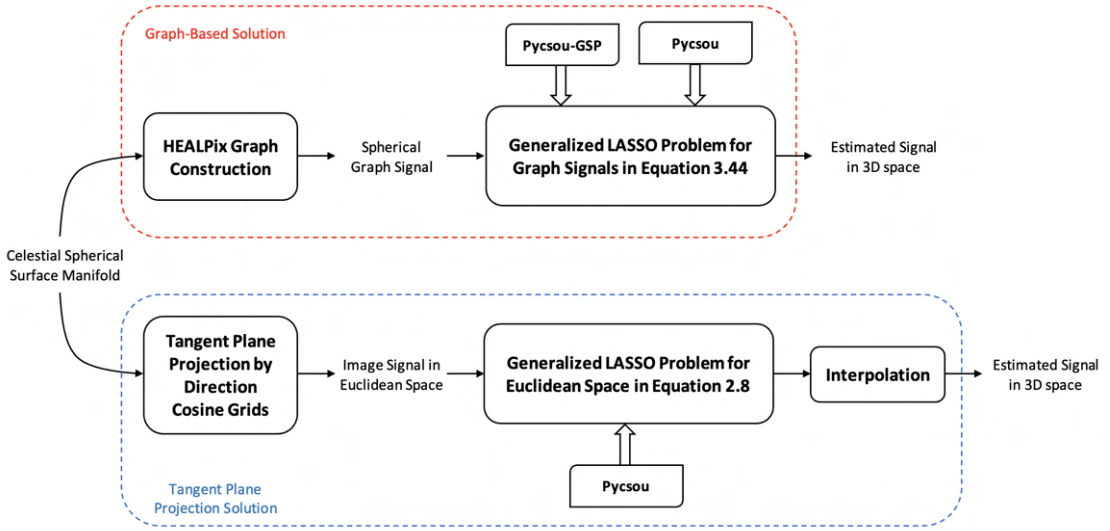


Figure 5.4: Block diagram of graph-based and tangent plane projection solutions for the application of inverse problems on celestial manifold signals.

5.2 HEALPix Graph Construction

Manifold for celestial signal represents the 2D surface of the celestial sphere. To discretize this manifold, HEALPix graph construction method is proposed which can be divided into two subsystem as illustrated in Figure 5.5. Firstly, as stated by Gorski et al [55], HEALPix pixelisation is applied for obtaining the vertices of the graph and then kNN algorithm is utilized for the creation of graph edges. Here, k is chosen as 8, which is not too small to model the manifold's geometry and not too large to have computationally expensive and memory inefficient adjacency matrix $\mathbf{W}$ as stated by Kang [26].

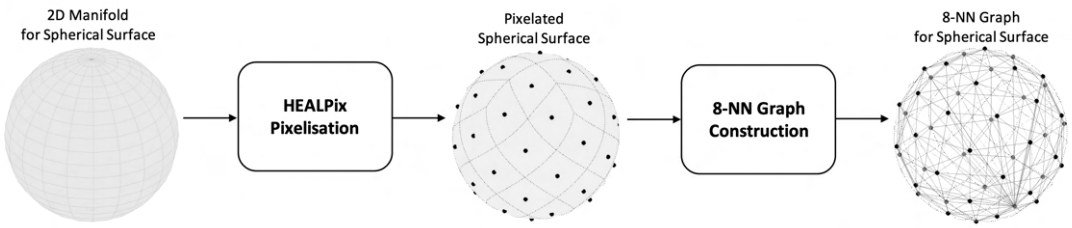


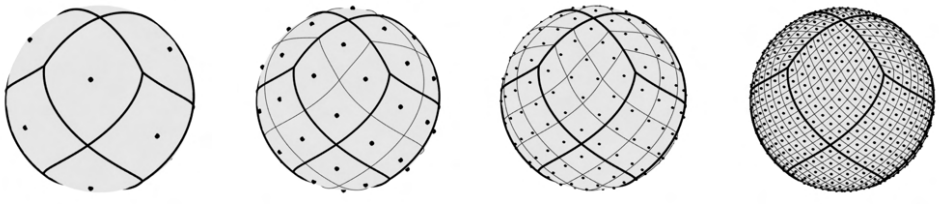
Figure 5.5: Block diagram of HEALPix graph construction.

kNN algorithm is described in Section 2.2.4 and 3.1 with detailed analysis. As explained before, it is a generic method for graph edge construction. However, HEALPix is proposed here for selecting the nodes or point cloud properly. HEALPix stands for Hierarchical Equal Area

isoLatitude Pixelation of a sphere. As implied by its name, this pixelation technique divides a spherical surface into smaller regions where each pixel has an equal surface area as any other pixel as indicated by Gorski et al [55]. The primary objectives in creating HEALPix were to establish a mathematical framework that enables the effective discretization of functions on a sphere at a high resolution and to simplify the rapid and precise statistical and astrophysical analysis of vast full-sky data sets. Therefore, this technique can be useful for celestial signals.

HEALPix fulfills the above requirements by possessing the following fundamental properties:

- The sphere is tessellated into curvilinear quadrilaterals in a hierarchical manner [55]. Twelve base pixels make up the division with the lowest resolution, that is related to the number of side $N_{\text{side}} = 1$. Resolution of the tessellation increases by division of each pixel into four new ones and the number of side is the number of pixel for one side of twelve base regions is denoted as N_{side}. Figure 5.6 illustrate the increase in the resolution from left to right with $N_{\text{side}} = 1, 2, 4, 8$. Here, total number of pixels can be calculated using $N_{\text{pix}} = 12N_{\text{side}}^2$.



(a) $N_{\text{side}} = 1$. (b) $N_{\text{side}} = 2$. (c) $N_{\text{side}} = 4$. (d) $N_{\text{side}} = 8$.

Figure 5.6: Orthographic view of the HEALPix pixelisation of the sphere for $N_{\text{side}} = 1, 2, 4, 8$.

- At a certain resolution, all pixel areas are the same.
- The pixel centers occur on rings of constant latitude Thus, it provides iso-latitude map projection [55].

All in all, this sampling technique's main goal is the effective storage and processing of large data sets for astronomical research. Also the method is beneficial in general when we want to handle mathematical functions defined on spherical surface. Therefore, it's conveniently used in the solution of the problem as presented in Figure 5.4. As a result, the graph signal is obtained for the graph-based solution.

5.3 Comparison of Methods on Synthetic Data

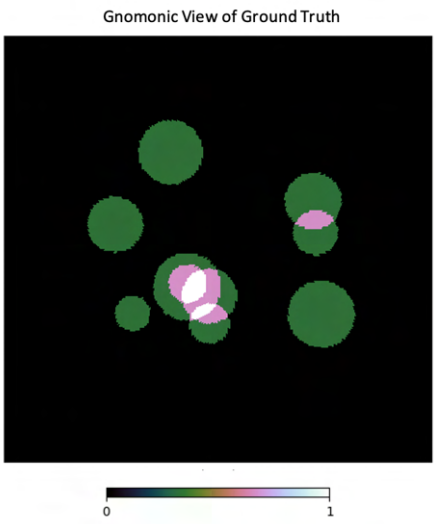
As illustrated in Figure 5.4, the constructed graph signal is given as an input to generalized LASSO problem for graph signals formulated in Section 3.6 Equation 3.44. The solution of this

problem requires graph convolution and graph gradient operators, which are implemented in the developed package Pycsou-GSP, as explained in Section 4.3 in detail. Since these operators are implemented as Pycsou operators, the whole system can be solved with the use of proximal algorithms in Pycsou. On the other hand, tangent plane projection is applied with the use of direction cosine grids as stated in [56]. However, this technique cannot be applied for entire part of the sphere. Only a small part of the sphere is chosen for projection and used for the comparison of two proposed techniques. For this purpose, a synthetic signal for celestial sphere is produced as explained in Section 5.1. There are random circles for a very small part in ground truth data, which can be seen in Figure 5.7a. Here, the zoomed out version is illustrated with gnomonic projection for better visualization.

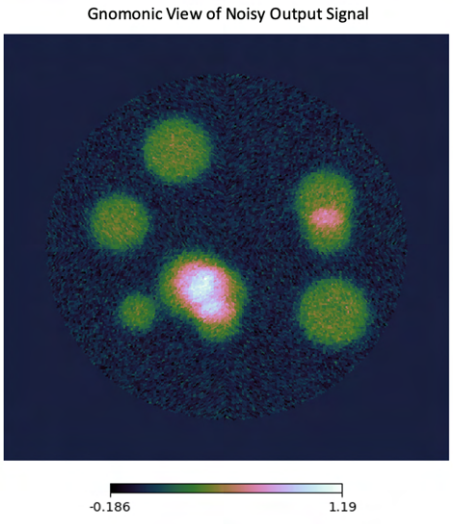
The described generated signals are chosen as a ground truth of the problem. In order to generate noisy output signal, this ground truth signal is firstly filtered and a Gaussian noise is added. For the filtering operation, heat kernel with diffusion rate $\tau = 5$ is chosen where the spectrum becomes $\hat{g}(\lambda_l) = e^{-5\lambda_l}$. This is a low-pass filter which models the blurring effect of the acquisition system. Pycsou-GSP's GraphConvolution is used and the operator is stored as a Pycsou.abc.operator.LinOp class, which is suitable for the proximal solution. To construct GraphConvolution, GraphLaplacian is also obtained as it's used as an input to GraphConvolution. Then, noise with zero mean and standard deviation of 0.0005 is added to the graph convoluted signal. The obtained signal can be seen in Figure 5.7b.

After creating the noisy output signal, the graph-based solution is applied to estimate the input ground-truth signal. Generalized LASSO problem is used as explained before. For the implementation, Pycsou.opt.solver.PD30 is used which implements primal dual splitting (PDS) algorithm [41]. This is suitable algorithm for generalized LASSO as explained in Section 2.1 and the details can be seen in Algorithm 2. For calling the proximal operator, both fitting term and regularization term should be given as an input. Graph convolution operator defined above is used for fitting terms. On the other hand, Pycsou-GSP's GraphGradient is used for regularization term. Also, the regularization constant λ in Equation 3.44 is tuned to be 0.1 by experimentation. Lastly, for the reconstruction, the initial estimated signal is given as well and it is chosen to be a randomized signal.

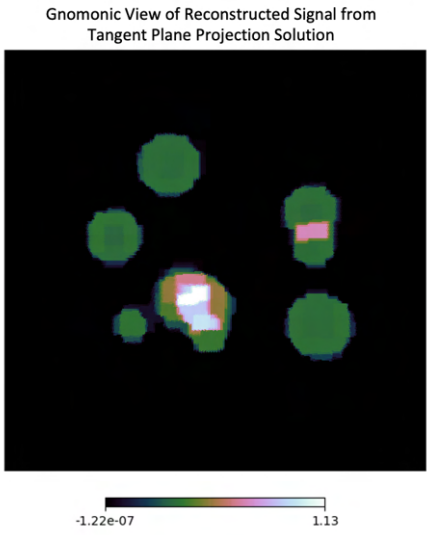
For the tangent plane method, the tangent plane projection with direction cosine grids [56] are applied after the generation of noisy output signal as explained previously. As a result, the projected signal is a 2D image in Euclidean space. For the solution of inverse problem for tangent plane projection method, generalised LASSO problem in Euclidean space formulated in Section 2.1 Equation 2.8 is solved with the use of Pycsou operators. Again, Pycsou.opt.solver.PD30 is used for the algorithm of PDS. Convolution and gradient operator is direcly implemented in Pycsou environment for fitting and regularization term, respectively. The reconstructed signal is also a 2D image signal. Then, the interpolation to sphere is applied to this signal. As a result, the estimated signal of this method can be seen in Figure 5.7c. On the other hand, the estimated signal of the graph-based solution can be seen in Figure 5.7d. The complete implementation of this test can be found in the link [57].



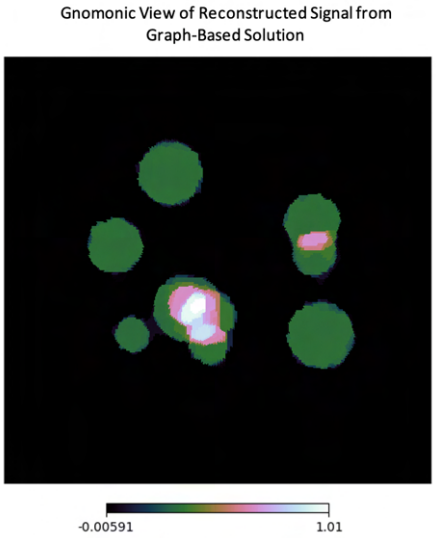
(a) Ground truth synthetic signal.



(b) Noisy output synthetic signal.



(c) Reconstructed synthetic signal from tangent plane projection solution.



(d) Reconstructed synthetic signal from graph-based solution.

Figure 5.7: Gnomonic views of ground truth, noisy output and reconstructed synthetic signals for linear inverse problem on manifold solved by graph signal processing methods and tangent plane projection, where output signal is resulted from graph convolution of ground truth with heat kernel of rate $\tau = 5$ with addition of Gaussian noise.

When the mean squared errors of estimated signals are computed for zoomed part of the estimated signals, we observe 28.90% and 16.49 for tangent plane projection and graph-based solutions, respectively. This result shows that the graph-based solution is more powerful in terms of accuracy for solving inverse problems on manifolds. Additionally, there are obvious difference between the visuals of estimated signals of two proposed methods. The one from

the graph-based method is very close to the ground truth signal as observed from 5.7. However, for the tangent plane projection solution, the deformations on the structure is obvious as expected. This is because of the lack of modeling of the sphere structure. The tangent plane projection method just ignores the manifold's structure but represents the data in an intrinsic point of view. However, HEALPix graph discretizes the sphere in an extrinsic perspective, which models the structure more suitably. As a result, there is less deformation observed in the estimated signal.

5.4 Implementation of Proposed Graph-Based Method on Real-World Signal

As motivated by the robustness of solutions by graph signal processing methods compared to tangent plane projection, the graph-based solution is tested on the real-world celestial signal. As a ground truth signal, a real-world WHAM signal is chosen as the details are given in Section 5.1 and its orthographic view can be seen in Figure 5.3. This view is only the projection of 3D coordinate space for 2D visualisation, but it requires two maps for front and back part of sphere. However, to show the data in one map, Mollweide view is chosen for the visualization where the ground truth WHAM signal can be seen in Figure 5.8a.

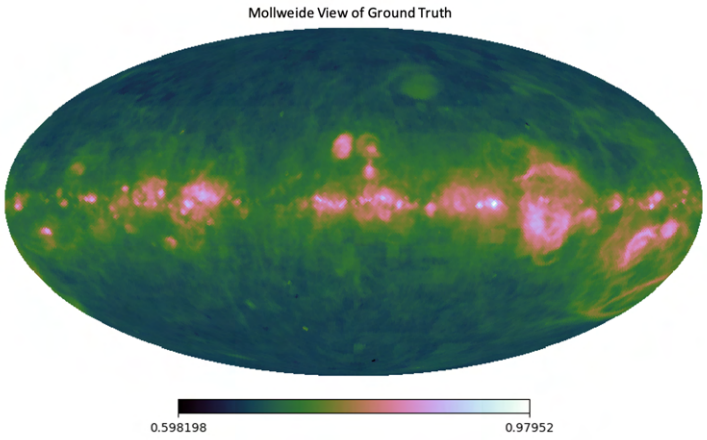
For the testing, signals of ground truth noisy output should be prepared again. As a ground truth signal, a real-world WHAM signal is chosen as the details are given in Section 5.1 and its orthographic view can be seen in Figure 5.3. This view is only the projection of 3D coordinate space for 2D visualisation, but it requires two maps for front and back part of sphere. However, to show the data in one map, Mollweide view is chosen for the visualization where the ground truth WHAM signal can be seen in Figure 5.8a.

To obtain noisy output signal, the similar technique with the convolution of heat kernel is applied in the same way as the previous section. For one more testing on real-world data, one more graph convolution operator is proposed with spherical pooling, which downgrades the resolution of HEALPix pixelisation and this can be useful to model the limited resolution in the acquisiton system. Pycsou-GSP's GraphConvolution is not used for this, the operator is directly implemented as a Pycsou.abc.operator.LinOp class where the functions of *healpy*, which is a Python package to handle pixelated data on the sphere [58], are used inside apply and adjoint methods. Noise addition is applied similarly and the resulted output can be seen in Figure 5.9b.

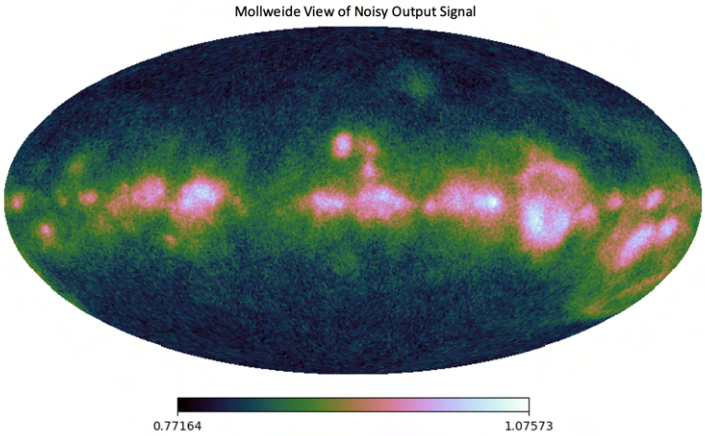
The solution by proximal algorithms are exactly the same as explained in the previous section. Reconstructed signals for tests with convolution of heat kernel and spherical pooling are obtained. The one with heat kernel can be seen in Figure 5.8c, while the other one can be seen in Figure 5.9c. The complete implementation of this test can be found in the link [59]. It is obviuous to see the deconvolution effects particularly on the fields that we're more interested in with larger values on equatorial line. These parts are much closer to the one in ground truth

signal when we compare it to the output signal. However, some artifacts on the parts of the sphere with small values that we're less interested in can also be seen. We cannot observe very smooth solution for these parts. This is because of the effect of Gaussian noise in the output. The reconstruction still adds the Gaussian noise effect particularly for these small values.

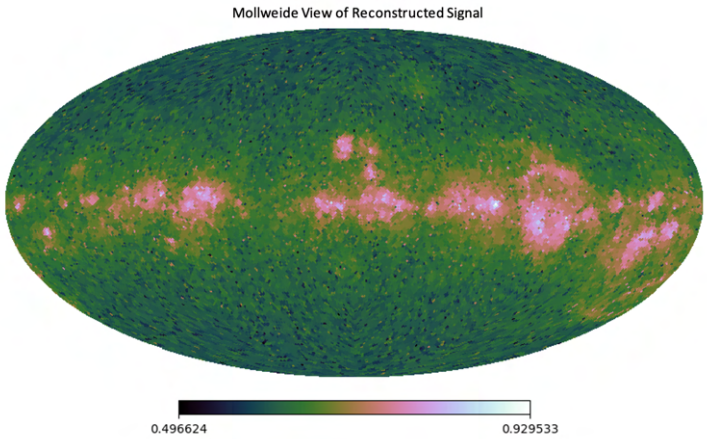
To conclude, this application in astronomy can be solved for a real-world WHAM signal by the proposed graph-based method where a proximal algorithm is used with graph signal processing methods. The power of this method compared to the traditional solution of tangent plane projection is also shown in the previous section. Therefore, graph-based solution proposed in this thesis offers a valid method for inverse problems on manifolds.



(a) Ground truth WHAM signal.

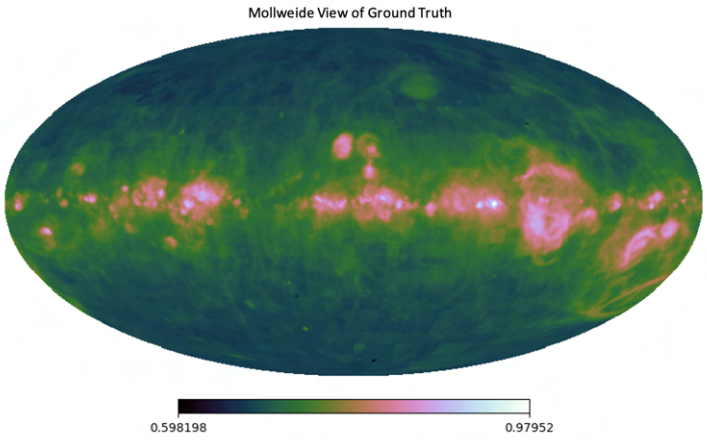


(b) Noisy output WHAM signal

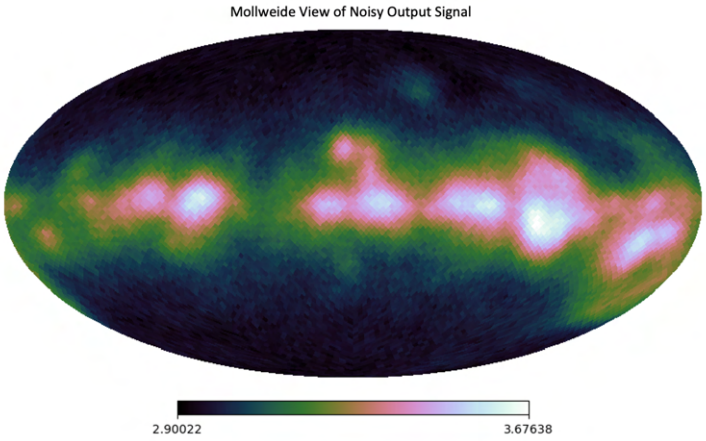


(c) Reconstructed WHAM signal

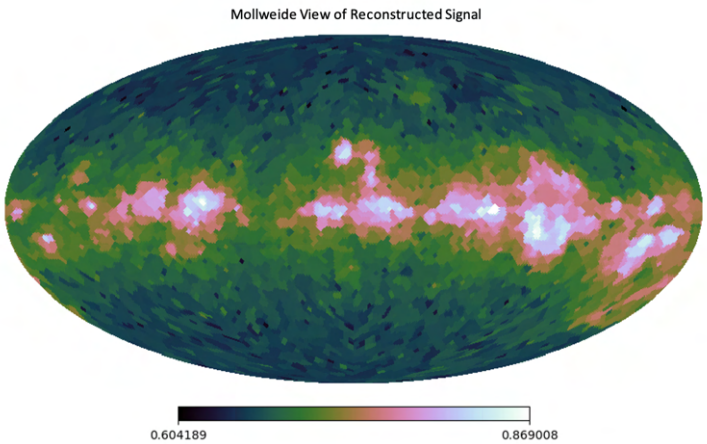
Figure 5.8: Mollweide views of ground truth, noisy output and reconstructed WHAM signals for linear inverse problem on manifold solved by graph signal processing, where output signal is resulted from heat kernel filtered input with addition of Gaussian noise.



(a) Ground truth WHAM signal.



(b) Noisy output WHAM signal



(c) Reconstructed WHAM signal

Figure 5.9: Mollweide views of ground truth, noisy output and reconstructed WHAM signals for linear inverse problem on manifold solved by graph signal processing, where output signal is resulted from spherical downsampled input with addition of Gaussian noise.

6 Conclusion

In this thesis, inverse imaging problems on manifolds were investigated. In order to carry out this research, the existing models and solutions in the literature of inverse problems for signals in the Euclidean domain were presented. The mathematical aspects of the solutions of such problems in the Euclidean domain were discussed in detail. Then, in order to solve this problem on manifolds defined in an irregular non-Euclidean domain, a mathematical definition of manifolds was given. Then, the discretization of the manifold was shown to be compulsory for proposing the solution of inverse problems. In order to do so, The intrinsic and extrinsic properties of manifolds were pointed out and it is concluded that attention should be paid to both for representing the manifold signals in a compact way. Intrinsic point of view focuses on local structure of manifold because of looking at each point with its own tangent space. On the other hand, the extrinsic point of view takes into account the global structure of manifold, which is very important for modeling geometric objects. Two different methods of discretization of manifolds were proposed putting different emphasis on these perspectives. The first one is the traditional method, tangent space projection and it was expected to have noisy output at the end because of the intrinsic perspective it has. The solution proposes the projection into Euclidean space and the solving the rest of problem with Euclidean methods. However, deformations in the global structure was unavoidable. The second proposed solution was the one solved by graph signals. They offer extrinsic discretization of manifolds, but the generalization of the inverse problems for graph signals required the elaborate analysis and definitions in theory of graph signal processing.

Theory of graph signal processing has led to the development of many methods in graph setting. First of all, a useful graph construction method was proposed for a generic manifold with the choice of nodes by pixelisation and generation of edges by kNN algorithm. Then, the valid mathematical definition of graph signals were presented with Hilbert spaces including the inner product and norm definitions, where the edge weights were not included to provide simple mathematical development similar to Eucldiean space. However, inclusion of edge weights were done in graph differential operators like gradient, divergence and Laplacian operators. In this thesis, graph Hessian is also proposed and all of the defined operators

are validared by the natural relation between the operators in classical calculus. Then, the definition of graph Laplacian offered the definition of spectral domain for graphs with the use of eigen-decomposition of graph Laplacian matrix. This created another basis of the graph signals, which gave graph Fourier transform operator from the eigenvector matrix of graph Laplacian. This definition mad the generalization of many operators in classical signal processing possible. These are convolution, shift, modulation, dilation and filtering. It was shown that shift did not have any meaning in irregular graph domain, therefore, the spectrum equivalent operations are applied. For convolution, multiplication in spectral domain is used and spectral filtering definition was obtained. Similarly, other operators were generalized and graph filtering was defined as graph convolution as well. Some examples of low-pass filtering was given and validated by experiments. At the end of mathematical analysis on graph signals, the integration of such signals to inverse problems were done by looking at generalised LASSO problem. Graph differential and convolution operators were seen to be useful for this problem. Therefore, the implementation aspects were decided to be done.

Computational analysis on graph signal processing was done for graph differential and convolution operators. It was seen that matrix-free Numba just-in time (Jit) compilation methods gave faster solution to sparse matrix multiplication implementation of operators. In order to provide matrix-free solution, preprocessing to weight adjacency matrix was proposed by the proposed RCD format and it was seen to work well. For graph filtering, the direct implementation of spectral filtering was shown to be very computationally expensive. Therefore, Chebyshev approximation is applied for the kernel defined on the spectrum of continuous domain. Discretized version of this approximation was also provided and shown to work with overfitting problems. However, considering to have Python function as an input, the direct Chebyshev approximation can be applied and shown to have computationally cheap and have localized transform, which can be useful for extracting the local information. Based on these solutions, a Python package called Pycsou-GSP was developed. It was designed as an extension to Pycsou, which provides solutions for inverse problems with proximal algorithms. However, this required the integration of the proposed methods into Pycsou classes and comply with Pycsou framework like modulation, precision agnosticity. For graph filtering, the Chebyshev coefficients were converted to monomial coefficients such that polynomial linear operator were used for faster solutions.

The development of Pycsou-GSP enabled the implementation of the proposed methods for solving inverse problems on manifolds. For this purpose, an application in the field of astronomy where the signal in the model is defined on celestial sphere. Therefore, the manifold represents the surface of the spherical structure. HEALPix pixelisation was proposed and discussed to provide efficient and correct modeling of the structure of manifold. Then, kNN algorithm was applied to obtain graph signals. For tangent plane projection, cosine grids are applied for the projection into Euclidean space. Firstly, with the use of Pycsou and Pycsou-GSP packages, the results of these two methods were obtained for synthetic data where random circles are generated for limited part of the sphere and it's shown that the graph-based solution give better estimation both in terms of accuracy of the structure and the minimum square

error. It is concluded that the graphs are suitable objects to model the continuous manifold in an extrinsic point of view. The usage of graph signal processing methods made the solution of inverse problems on manifolds more robust. Therefore, the implementation on real-world signal was also tested. The Wisconsin H-Alpha Mapper (WHAM) signal was chosen for that and two different modeling of acquisition system with heat kernel and spherical sampling were tested. The results of both modeling showed the estimation of signals to be closer to the ground truth signal. There were some artifacts observed for the low values of the signal, however for the parts of signal that are more interesting, the estimation is obvious.

For the future works, the graph-based methods could be extended for more. First of all, in this thesis the solution of inverse problems on manifolds were handled by generalized LASSO problem. Other types of inverse problems like generalised Tikhonov can be given by choosing the defining Bayesian approach on graph signals more elaborately. In this respect, the probability theory for graph signal can be formulated in more detail. Additionally, The comparison of the methods can also be done for more complicated manifolds. In this thesis, the application is done on manifold that represents 2D spherical surface. Here, the graph construction method is critical for having a good performance because of the need to represent the structure adequately. Therefore, more elaborate solutions could be proposed for graph construction methods as well. In addition, graph smoothness functions are defined for limited types of signals, it can also be generalized for making it corresponded to different apriori information. For graph filtering, there is a method called graph wavelet transform, which provide better localized filtering, which can be used for better feature extraction in inverse problems as well. Furthermore, in the development of Pycsou-GSP, not every operators and methods in theory of GSP could be implemented. For example, the efficient implementation of graph Fourier transform cannot be provided here and it is left as a future work. Also, different smoothness functions can be defined with Pycsou classes as well to be directly used in Pycsou's proximal functions. Hessian operator is not implemented either and the efficient solution could be proposed as well. In this thesis, the graphs that we deal with were only finite-weighted undirected graphs. Even though for manifolds the undirected edges might not give meaningful structure in most of the cases, we can also extend the methods for directed graphs as well. This requires more elaborate analysis on graph theory but it would give very significant solutions to different applications.

A Appendix: Validation of Calculus for Graph Signals

A.1 Proof of Adjointness of Graph Gradient and Graph Divergence

Graph divergence is adjoint operator of graph gradient.

Proof.

$$\begin{aligned}
<\nabla f, G>_{\mathscr{H}(\mathscr{E})} &= \sum_{(i,j)\in\mathscr{E}} (\nabla f)_{ij} G_{ij} \\
&= \sum_{(i,j)\in\mathscr{E}} \sqrt{w_{ij}}(f_i - f_j) G_{ij} \\
&= \sum_{(i,j)\in\mathscr{E}} \sqrt{w_{ij}} f_i (G_{ij} - G_{ji}) \\
&= \sum_{i\in\mathscr{V}} f_i \sum_{j\in\mathscr{V}} \sqrt{w_{ij}}(G_{ij} - G_{ji}) \\
&= \sum_{i\in\mathscr{V}} f_i (\mathrm{div} G)_i =< f, \mathrm{div} G >_{\mathscr{H}(\mathscr{V})}
\end{aligned}$$

A.2 Proof of Graph Hessian Being the Trace of Graph Laplacian

Graph Hessian is the trace of graph Laplacian.

Proof.

$$\begin{aligned}
\mathrm{tr}(Hf)_i &= \sum_{j,k\in\mathscr{V}:\, j=k} (Hf)_{ijk} \\
&= \sum_{j,k\in\mathscr{V}:\, j=k} \frac{w_{ij}}{2}(f_i - f_j) + \frac{w_{ik}}{2}(f_i - f_k) \\
&= \sum_{j\in\mathscr{V}} w_{ij}(f_i - f_j) = (Lf)_i
\end{aligned}$$

A.3 Proof of Graph Laplacian Being Positive Semi-Definite

Graph Laplacian **L** is positive semi-definite.

Proof.

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \sum_{(i,j)\in\mathscr{E}} w_{ij}(f_i - f_j)^2 \geq 0$$

A.4 Proof of Mean Conservation in Shift Operator

Mean of graph signal can be defined as follows

$$\sum_{i=0}^{N-1} g(i) = \sqrt{N}\sum_{i=0}^{N-1} g(i)\frac{1}{\sqrt{N}} = \sqrt{N}\sum_{i=0}^{N-1} g(i)u_0^*(i)$$

where the eigenvector corresponding to eigenvalue $\lambda_0 = 0$, $u_0(i) = \frac{1}{\sqrt{N}}$. This property can be shown from the fact the solution of the equation $\mathbf{L}u_0 = 0$.

This equation can also be written as:

$$\sum_{i=0}^{N-1} g(i) = \sqrt{N}\hat{g}(0)$$

Therefore, the conversion of the mean in shift operator for graph signal can be shown in the following proof.

Proof.

$$\begin{aligned}\sum_{i=0}^{N-1}(T_n g)(i) &= \sqrt{N}\widehat{(T_n g)}(0)\\ &= \sqrt{N}\sqrt{N}\hat{g}(0)u_0^*(n)\\ &= N\frac{1}{\sqrt{N}}\hat{g}(0)\\ &= \sqrt{N}\hat{g}(0) = \sum_{i=0}^{N-1} g(i)\end{aligned}$$

We can conclude that $\sqrt{N}$ in shift operator constant enables the mean conservation.

B Appendix: Theory of Classical Signal Processing

B.1 Differential Operators in Classical Calculus

Given that input signal for classical calculus x is a continuous time signal that is dependent on $\{t_i\}_{i=1}^N$, while the one for the graph calculus f is a graph signal.

B.1.1 Gradient Operator

Gradient $\nabla : \mathbb{R} \rightarrow \mathbb{R}^N$ is a linear operator that is defined as

$$(\nabla x) = \begin{bmatrix} \frac{dx}{dt_1} & \dots & \frac{dx}{dt_N} \end{bmatrix}^T \tag{B.1}$$

B.1.2 Divergence Operator

Divergence $\text{div} : \mathbb{R}^N \rightarrow \mathbb{R}$ is a linear operator that is defined as

$$\text{div} X = \sum_{i=1}^{N} \frac{dX_i}{dt_i} \tag{B.2}$$

B.1.3 Laplacian Operator

Laplacian $L : \mathbb{R} \rightarrow \mathbb{R}$ is a linear operator and it can be defined as the divergence of the gradient operator using B.1 and B.2.

$$Lx = \text{div}\nabla x = \sum_{i=1}^{N} \frac{d^2 x}{dt_i^2} \tag{B.3}$$

B.2 Definition of the Spectrum

B.2.1 Eigen-Decomposition of Laplacian

Given that the signal is dependent only on one parameter, then the Laplacian in B.3 becomes $Lx = \frac{d^2x}{dt^2}$. Then, the complex exponential, e^{jwt} is eigenvector of the Laplacian operator.

$$Le^{jwt} = \frac{d^2e^{jwt}}{dt^2} = -w^2e^{jwt} \tag{B.4}$$

Here, w is proportional to the eigenvalue and it is called as the angular frequency. The corresponded eigenvectors vary more rapidly as this angular frequency w increases. Then, e^{jwt} provides a basis for continuous-time signals.

B.2.2 Fourier Transform

Continuous time Fourier Transform (CTFT) can be defined as the projection of the continuous-time signal into complex exponential basis, in other words inner product of the signal with eigenvector of Laplacian operator.

$$X(w) =< x(t), e^{jwt} >= \int x(t)e^{-jwt}dt \tag{B.5}$$

This is the representation of the signal in different domain, which is called the frequency domain. It's also called as the spectrum of the signal. Similarly, the inverse CTFT is defined as the synthesis operation with the complex exponential basis.

$$x(t) = \int X(w)e^{jwt}dw \tag{B.6}$$

C Graph Filtering Design Examples

C.1 Diffusion Problem

An example of a discrete diffusion operator is the heat diffusion operator denoted as **R**, which can be defined as follows

$$R = e^{-\tau \mathbf{L}} \tag{C.1}$$

where L is graph Laplacian operator and τ is the rate of diffusion. As stated by Shuman et al [34], ehen the rates of flow are proportional to the edge weights encoded in, applying various powers τ of the heat diffusion operator to a signal f depicts the flow of heat over the graph intuitively. Given $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^T$, the operator results in as follows

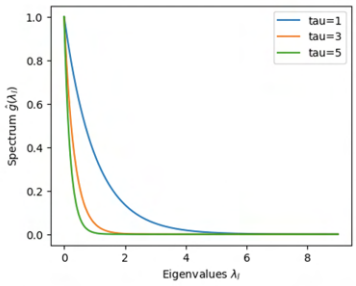
$$\begin{aligned}\mathbf{R} &= e^{-\tau \mathbf{U}\Lambda\mathbf{U}^T} \\ &= \mathbf{U}e^{-\tau\Lambda}\mathbf{U}^T\end{aligned}$$

In graph spectral domain, $\mathbf{U}^T\mathbf{R} = e^{-\tau\Lambda}\mathbf{U}^T$, thus the kernel spectrum is $\hat{g}(\lambda_l) = e^{-\tau\lambda_l}$, which is called as heat kernel. This is a low-pass filter that it's more selective for larger values of λ_l as seen from Figure C.1a. The diffusion of an impulsive signal on manifold and graph can be seen in Figure C.1 and C.2. Also, the spectrums of the output in Figure C.2 show the low-pass characteristic of the heat kernel.

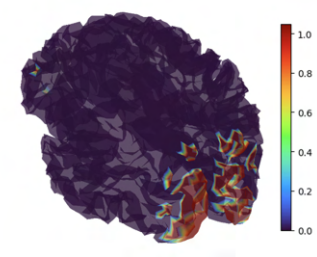
C.2 Tikhonov Regularization

Tikhonov problem can be formulated for graph signals, while treating the graph signal as N dimensional real vector and using quadratic smoothness function for regularization. As a result, the formulation is as follows

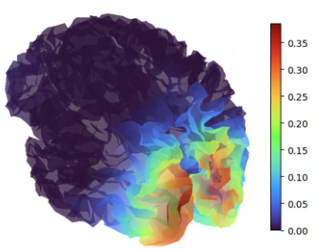
$$\arg\min_{\mathbf{f}\in\mathbb{R}^N} ||\mathbf{f}-\mathbf{y}||_2^2 + \lambda \mathbf{f}^T\mathbf{L}\mathbf{f} \tag{C.2}$$



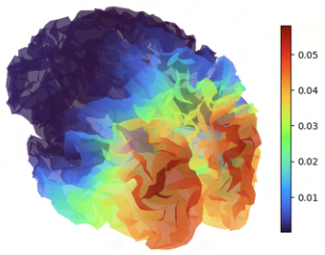
(a) Spectrum of heat kernel for rates $\tau = 1,3,5$.



(b) Input manifold brain signal.

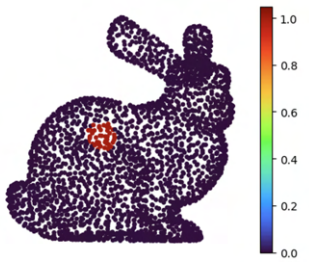


(c) Output manifold brain signal of heat kernel with rate $\tau = 1$.

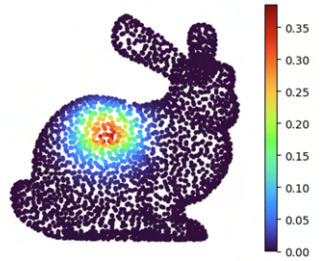


(d) Output manifold brain signal of heat kernel with rate $\tau = 10$.

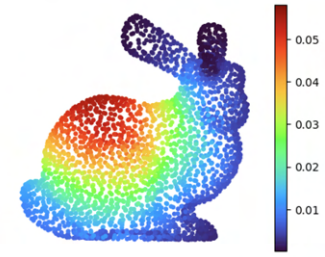
Figure C.1: Heat kernel on manifolds with rate $\tau = 1, 10$.



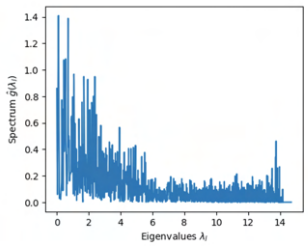
(a) Input bunny graph signal.



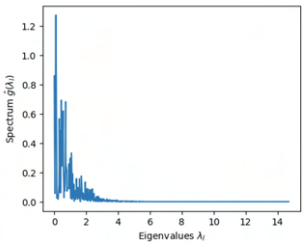
(b) Output bunny graph signal of heat kernel with rate $\tau = 1$.



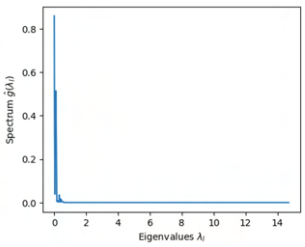
(c) Output bunny graph signal of heat kernel with rate $\tau = 10$.



(d) Spectrum of input bunny graph signal.



(e) Spectrum of output bunny graph signal of heat kernel with rate $\tau = 1$.



(f) Spectrum of output bunny graph signal of heat kernel with rate $\tau = 10$.

Figure C.2: Heat kernel on graph signals of rates $\tau = 1, 10$ with their spectrum.

The first-order optimality conditions of the convex objective function show that the optimal reconstruction is given by

$$f^{\star}(i) = \sum_{l=0}^{N-1} \left[\frac{1}{1+\gamma\lambda_l} \right] \hat{y}(\lambda_l) u_l(i) \tag{C.3}$$

or, equivalently, $\mathbf{f} = \hat{h}(Ł)\mathbf{y}$, where $\hat{h}(\lambda) = \frac{1}{1+\gamma\lambda}$ can be viewed as a low-pass filter.

Proof.

The objective function in the minimization problem is convex. Therefore, the problem can be solved by the first-order optimality condition. Firstly, the derivative of the objective function is taken and making equal to zero. Then the following equation is as follows

$$\mathbf{f} - \mathbf{y} + \gamma \mathbf{L}\mathbf{f} = 0$$

Use the fact that $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^T$, then we have the following

$$\gamma \mathbf{U}\Lambda\mathbf{U}^T\mathbf{f} = \mathbf{y} - \mathbf{f}$$

Then, the graph Fourier transforms of both sides are taken, i.e. multiplied by $\mathbf{U}^T$

$$\gamma \Lambda\mathbf{U}^T\mathbf{f} = \hat{\mathbf{y}} - \hat{\mathbf{f}}$$

Then, this equation can be written in graph spectral domain

$$\gamma \Lambda\hat{\mathbf{f}} = \hat{\mathbf{y}} - \hat{\mathbf{f}}$$
$$\hat{\mathbf{f}} + \gamma \Lambda\hat{\mathbf{f}} = \hat{\mathbf{y}}$$

This can also be written as

$$\begin{aligned} \hat{f}(\lambda_l) + \gamma\lambda_l \hat{f}(\lambda_l) &= \hat{y}(\lambda_l) \\ \hat{f}(\lambda_l) &= \frac{1}{1+\gamma\lambda_l} \hat{y}(\lambda_l) \end{aligned} \tag{C.4}$$

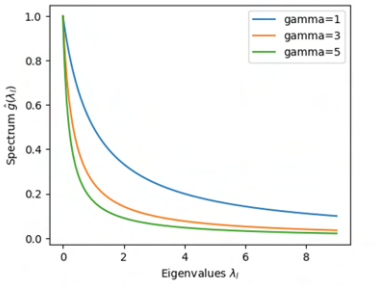
The graph filter as a solution to Tikhonov problem is characterized by the spectrum with the following function

$$\hat{g}(\lambda_l) = \frac{1}{1+\gamma\lambda_l} \tag{C.5}$$

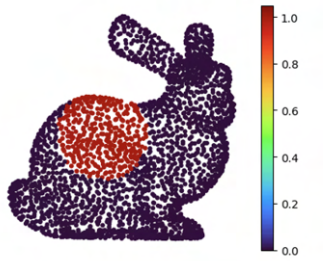
where γ is the regularization constant, whose increase results in smoother solution. This low-pass filtering characteristic can be seen in Figure C.3a.

Tikhonov problem solution by kernel can be tested for a graph signal given in Figure C.3b. Gaussian noise is added to that signal, which results in Figure C.3c. The solution by the kernel

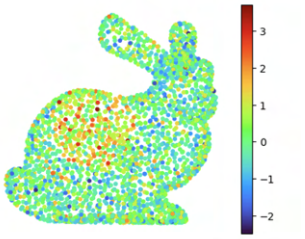
with rate $\gamma = 10$ results in estimation of ground truth signal where detection of circle shape on bunny signal is visible in C.3d.



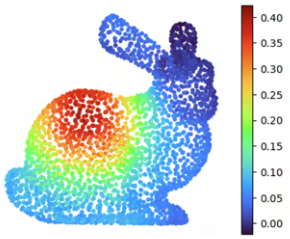
(a) Spectrum of kernel for Tikhonov problem for rates $\gamma = 1, 3, 5$.



(b) Ground truth bunny signal.



(c) Noisy bunny signal.

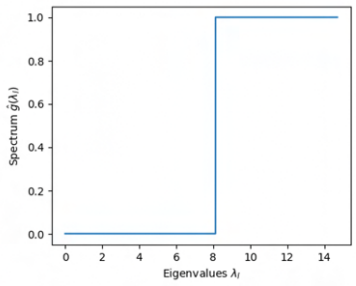


(d) Solution bunny signal by kernel for Tikhonov problem with rate $\gamma = 10$.

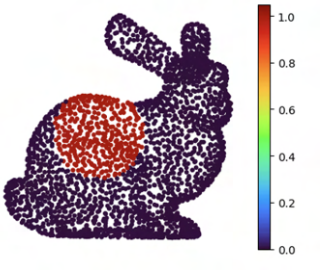
Figure C.3: Tiknonov problem solution by kernel on bunny graph signal with rate $\gamma = 10$.

C.3 Edge Detection by Ideal High Pass Filter

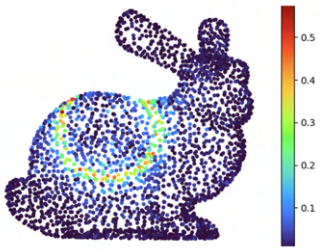
Edge detection for graph signals can be possible by simply applying high pass filter on the graph signal. The ideal high pass filter spectrum can be seen in Figure C.4a. Input signal has a visible circle as seen in Figure C.4b and the edges can be detected from the presented high pass filter as the output can be seen in Figure C.4c.



(a) Spectrum of ideal high-pass filter.



(b) Input bunny graph signal.



(c) High pass filtered bunny graph signal.

Figure C.4: High pass filtering on bunny graph signal.

Bibliography

[1] M. Simeoni, "Lecture slides: mathematical foundations of signal processing", Annotated slides, 2021. [Online]. Available: http://infoscience.epfl.ch/record/283918.

[2] N. Parikh and S. Boyd, "Proximal algorithms", *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014, ISSN: 2167-3888. DOI: 10.1561/2400000003. [Online]. Available: http://dx.doi.org/10.1561/2400000003.

[3] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems", *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009. DOI: 10.1137/080716542. eprint: https://doi.org/10.1137/080716542. [Online]. Available: https://doi.org/10.1137/080716542.

[4] S. Arridge, P. Maass, O. Öktem, and C.-B. Schönlieb, "Solving inverse problems using data-driven models", *Acta Numerica*, vol. 28, pp. 1–174, 2019. DOI: 10.1017/S0962492919000059.

[5] M. Dashti and A. M. Stuart, *The bayesian approach to inverse problems*, 2015. arXiv: 1302.6989 [math.PR].

[6] A. M. Stuart, "Inverse problems: a bayesian perspective", *Acta Numerica*, vol. 19, pp. 451–559, 2010. DOI: 10.1017/S0962492910000061.

[7] M. Vetterli, J. Kovačević, and V. K. Goyal, *Foundations of Signal Processing*. Cambridge University Press, 2014. DOI: 10.1017/CBO9781139839099.

[8] H. Gupta, J. Fageot, and M. Unser, "Continuous-domain solutions of linear inverse problems with tikhonov versus generalized tv regularization", *IEEE Transactions on Signal Processing*, vol. 66, no. 17, pp. 4670–4684, 2018. DOI: 10.1109/TSP.2018.2860549.

[9] P. Hansen, *Discrete Inverse Problems: Insight and Algorithms* (Fundamentals of Algorithms). Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2010, ISBN: 9780898718836. [Online]. Available: https://books.google.ch/books?id=r-uK2bzAUrUC.

[10] H. Pishro-Nik, *Introduction to Probability, Statistics, and Random Processes*. Kappa Research, LLC, 2014, ISBN: 9780990637202. [Online]. Available: https://books.google.ch/books?id=3yq%5C_oQEACAAJ.

[11] A. F. Vidal, V. D. Bortoli, M. Pereyra, and A. Durmus, *Maximum likelihood estimation of regularisation parameters in high-dimensional inverse problems: an empirical bayesian approach. part i: methodology and experiments*, 2020. arXiv: 1911.11709 [stat.ME].

[12] A. P. Valentine and M. Sambridge, "Optimal regularization for a class of linear inverse problem", *Geophysical Journal International*, vol. 215, no. 2, pp. 1003–1021, Jul. 2018, ISSN: 0956-540X. DOI: 10.1093/gji/ggy303. eprint: https://academic.oup.com/gji/article-pdf/215/2/1003/39586071/gji_215_2_1003.pdf. [Online]. Available: https://doi.org/10.1093/gji/ggy303.

[13] A. Ali and R. J. Tibshirani, *The generalized lasso problem and uniqueness*, 2019. arXiv: 1805.07682 [math.ST].

[14] "Gradient methods", in *An Introduction to Optimization*. John Wiley & Sons, Ltd, 2008, ch. 8, pp. 125–153, ISBN: 9781118033340. DOI: https://doi.org/10.1002/9781118033340.ch8. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118033340.ch8. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118033340.ch8.

[15] H. Li and Z. Lin, "Accelerated proximal gradient methods for nonconvex programming", in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/f7664060cc52bc6f3d620bcedc94a4b6-Paper.pdf.

[16] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems", *Acta Numerica*, vol. 14, pp. 1–137, 2005. DOI: 10.1017/S0962492904000212.

[17] R. Bergmann, R. Herzog, M. S. Louzeiro, D. Tenbrinck, and J. Vidal-Núñez, *Fenchel duality theory and a primal-dual algorithm on riemannian manifolds*, 2020. arXiv: 1908.02022 [math.NA].

[18] L. Condat, "A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms", *Journal of Optimization Theory and Applications*, vol. 158, pp. 460–479, 2013.

[19] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data", *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, Jul. 2017. DOI: 10.1109/msp.2017.2693418. [Online]. Available: https://doi.org/10.1109%5C%2Fmsp.2017.2693418.

[20] I. Chavel, *Riemannian Geometry: A Modern Introduction* (Cambridge Studies in Advanced Mathematics), 2nd ed. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511616822.

[21] J. Nash, "The imbedding problem for riemannian manifolds", *Annals of Mathematics*, vol. 63, no. 1, pp. 20–63, 1956, ISSN: 0003486X. [Online]. Available: http://www.jstor.org/stable/1969989 (visited on 06/20/2023).

[22] M. P. do Carmo, *Differential geometry of curves and surfaces*. Prentice Hall, 1976, pp. I–VIII, 1–503, ISBN: 978-0-13-212589-5.

[23] R. Bhattacharya and V. Patrangenaru, "Large sample theory of intrinsic and extrinsic sample means on manifolds. i", *The Annals of Statistics*, vol. 31, no. 1, pp. 1–29, 2003, ISSN: 00905364. [Online]. Available: http://www.jstor.org/stable/3448366 (visited on 06/22/2023).

[24] N. Boumal, *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023. DOI: 10.1017/9781009166164. [Online]. Available: https://www.nicolasboumal.net/book.

[25] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Levy, *Polygon Mesh Processing*. CRC Press, 2010, ISBN: 9781439865316. [Online]. Available: https://books.google.ch/books?id=AuXqBgAAQBAJ.

[26] S. Kang, "K-nearest neighbor learning with graph neural networks", *Mathematics*, vol. 9, no. 8, 2021, ISSN: 2227-7390. DOI: 10.3390/math9080830. [Online]. Available: https://www.mdpi.com/2227-7390/9/8/830.

[27] R. Diestel, *Graph Theory (Graduate Texts in Mathematics)*. Springer, Aug. 2005, ISBN: 3540261826. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20%7B%5C&%7Dpath=ASIN/3540261826.

[28] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. New York: Elsevier, 1976.

[29] W. Huang, T. A. W. Bolton, J. D. Medaglia, D. S. Bassett, A. Ribeiro, and D. Van De Ville, "A graph signal processing perspective on functional brain imaging", *Proceedings of the IEEE*, vol. 106, no. 5, pp. 868–885, 2018. DOI: 10.1109/JPROC.2018.2798928.

[30] A. Ortega, *Introduction to Graph Signal Processing*. Cambridge University Press, 2022, ISBN: 9781108428132. [Online]. Available: https://books.google.ch/books?id=iS0fzgEACAAJ.

[31] S. Shekkizhar and A. Ortega, *Neighborhood and graph constructions using non-negative kernel regression*, 2023. arXiv: 1910.09383 [cs.LG].

[32] B. Hopkins and R. J. Wilson, "The truth about königsberg", *The College Mathematics Journal*, vol. 35, no. 3, pp. 198–207, 2004. DOI: 10.1080/07468342.2004.11922073. eprint: https://doi.org/10.1080/07468342.2004.11922073. [Online]. Available: https://doi.org/10.1080/07468342.2004.11922073.

[33] G. Cheung, E. Magli, Y. Tanaka, and M. K. Ng, "Graph spectral image processing", *Proceedings of the IEEE*, vol. 106, no. 5, pp. 907–930, 2018. DOI: 10.1109/JPROC.2018.2799702.

[34] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains", *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, May 2013. DOI: 10.1109/msp.2012.2235192. [Online]. Available: https://doi.org/10.1109%5C%2Fmsp.2012.2235192.

[35] T. Biyikoğlu, J. Leydold, and P. F. Stadler, "Laplacian eigenvectors of graphs", 2007.

[36] D. Burago, S. Ivanov, and Y. Kurylev, *A graph discretization of the laplace-beltrami operator*, 2014. arXiv: 1301.2222 [math.AP].

[37] "Chapter eight. positive-semidefinite matrices", in *Theory, Facts, and Formulas (Second Edition)*. Princeton: Princeton University Press, 2009, pp. 459–596, ISBN: 9781400833344. DOI: doi:10.1515/9781400833344.459. [Online]. Available: https://doi.org/10.1515/9781400833344.459.

[38] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997.

[39] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd. USA: Prentice Hall Press, 2009, ISBN: 0131988425.

[40] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, 2017. arXiv: 1606.09375 [cs.LG].

[41] M. SIMEONI and P. del Aguila Pla, *Matthieumeo/pycsou: pycsou 1.0.6*, version v1.0.6, Apr. 2021. DOI: 10.5281/zenodo.4715243. [Online]. Available: https://doi.org/10.5281/zenodo.4715243.

[42] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[43] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "Cupy: a numpy-compatible library for nvidia gpu calculations", in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. [Online]. Available: http://learningsys.org/nips17/assets/papers/paper_16.pdf.

[44] M. Rocklin, "Dask: parallel computation with blocked algorithms and task scheduling", in *Proceedings of the 14th python in science conference*, Citeseer, 2015.

[45] J. Scott and M. Tůma, "Sparse matrices and their graphs", in *Algorithms for Sparse Linear Systems*. Cham: Springer International Publishing, 2023, pp. 19–30, ISBN: 978-3-031-25820-6. DOI: 10.1007/978-3-031-25820-6_2. [Online]. Available: https://doi.org/10.1007/978-3-031-25820-6_2.

[46] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[47] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[48] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: a llvm-based python jit compiler", in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.

[49] M. Defferrard, L. Martin, R. Pena, and N. Perraudin, *Pygsp: graph signal processing in python*, version v0.5.0, Oct. 2017. DOI: 10.5281/zenodo.1003158. [Online]. Available: https://doi.org/10.5281/zenodo.1003158.

[50] D. I. Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, *Distributed signal processing via chebyshev polynomial approximation*, 2017. arXiv: 1111.5239 [cs.DC].

[51] K. Okumus, *Pycsou-gsp*, https://github.com/okumuskaan/pycsou-gsp/tree/v2-dev, 2023.

[52] J. B. Kaler, *The Ever-changing Sky: A Guide to the Celestial Sphere*. 2002.

[53] R. Smith, "Astrophysical techniques, sixth edition, by c.r.kitchin", *Contemporary Physics*, vol. 55, Oct. 2014. DOI: 10.1080/00107514.2014.948932.

[54] G. J. Madsen, L. M. Haffner, and R. J. Reynolds, "The WHAM survey of ionized gas in the galaxy.",
memsai, vol. 77, p. 1163, Jan. 2006.

[55] K. M. Gorski, E. Hivon, A. J. Banday, *et al.*, "HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere", *The Astrophysical Journal*, vol. 622, no. 2, pp. 759–771, Apr. 2005. DOI: 10.1086/427976. [Online]. Available: https://doi.org/10.1086%2F427976.

[56] B. Xavier, M. Egydio-Silva, G. Sadowski, B. Silva, and V. Takara, "Construction of structural geological model using monte carlo simulation", Feb. 2021. DOI: 10.21203/rs.3.rs-212576/v1.

[57] K. Okumus, *Pycsou-gsp: notebook for comparison of gsp and tangent plane solution*, https://github.com/okumuskaan/pycsou-gsp/blob/v2-dev/application/HEALPix_vs_DCOS.ipynb, 2023.

[58] A. Zonca, L. Singer, D. Lenz, *et al.*, "Healpy: equal area pixelization and spherical harmonics transforms for data on the sphere in python", *Journal of Open Source Software*, vol. 4, no. 35, p. 1298, Mar. 2019. DOI: 10.21105/joss.01298. [Online]. Available: https://doi.org/10.21105/joss.01298.

[59] K. Okumus, *Pycsou-gsp: notebook for application in astronomy on real-world signals*, https://github.com/okumuskaan/pycsou-gsp/blob/v2-dev/application/App_RealData.ipynb, 2023.

KAAN OKUMUŞ

1007, Avenue Des Bains 9, Lausanne Switzerland
+41 78 230 85 79
kaan.okumus@epfl.ch
Personal Website: https://okumuskaan.github.io/

EDUCATION

Master of Science, École Polytechnique Fédérale de Lausanne (EPFL) — Sep. 2021 - Present
Communication Systems

Bachelor of Science, Middle East Technical University (METU) — Sep. 2016 - Jul. 2021
Electrical and Electronics Engineering

EXPERIENCE

Student Assistant — Apr. 2023 - Jun. 2023
École Polytechnique Fédérale de Lausanne (EPFL)

I worked as a student assistant of a master course Mobile Networks COM-405. My main duty was helping the students about the homeworks in the exercise session, grading the homeworks and helping the professor and teaching assistant about anything related to the course.

Student Assistant — Apr. 2022 - Jul. 2022
EPFL Hub for Advanced Image Reconstruction

I contributed to the development of PycSou v2, which is a Python 3 package for solving linear inverse problems with state-of-art proximal algorithms. I built universal functions, LSMR/LSQR algorithms. Source: https://github.com/matthieumeo/pycsou/tree/v2-dev

Intern — Jul. 2022 - Sep. 2022
Kandou Bus

Differential evolution and genetic algorithms are analyzed and implemented for the optimization of PLL performance parameters. A GUI is designed to provide a complete app that wraps everything implemented with a visually appealing interface.

Intern — Aug. 2020 - Sep. 2020
HAVELSAN A.Ş., Information and Communication Technologies

I designed and developed a cat breed detection app using object detection and image classification tools with deep learning techniques. I also designed a GUI in Python for my application.

Intern — Jun. 2020 - Aug. 2020
ASELSAN A.Ş., SST - Digital and Embedded Systems

Using MicroZed 7010, I designed various embedded hardware systems using Vivado with software in C. I also designed Digital IIR and FIR filters using Python and run it on MicroZed OS compiled by Petalinux.

PUBLICATIONS

PoGaIN: Poisson-Gaussian Image Noise Modeling from Paired Samples — Nov. 2022
N Bähler, ME Helou, E Objois, K Okumuş, S Süsstrunk — *IEEE Signal Processing*

Poisson-Gaussian image noise with paired noisy and noise-free samples is analyzed. A cumulant-based approach was derived. We achieved improved performance over different baselines and additionally derive the log-likelihood function for further insight. Source: https://arxiv.org/abs/2210.04866

DiffuserCam Project Report Dec. 2021
MO Kaya, K Okumuş, E Mielonen, A Jarret *EPFL Infoscience*

Lensless imaging is performed and a reconstruction of clean images is achieved by state-of-the-art optimisation algorithms with Pycsou. Source: `https://infoscience.epfl.ch/record/291501?ln=en`

COMPUTER SKILLS

C, C++, MATLAB, Simulink, Java, Scala, Python with Numpy, Scipy, TensorFlow and PyTorch
HTML, CSS, Javascript, MySQL, Linux, Bash, Swift

LANGUAGE PROFICIENCY

Turkish (*Native Speaker*), English (*Advanced*), French (*Intermediate*)

HONORS AND AWARDS

Award for best user interface as a member of the team, HAI-CO. Jul. 2021
Middle East Technical University, Electrical and Electronics Engineering - Senior Engineering Design Committee

Doç. Dr. Bülent Kerim Altay (BKA) Award Jun. 2021
Middle East Technical University, Electrical and Electronics Engineering

Dean's High Honor Roll - 8 times (every semester) 2017-2021
Middle East Technical University

PROJECTS

Optimisation of Statistical Arbitrage with Signal Processing Techniques Jan. 2023
École Polytechnique Fédérale de Lausanne, Audiovisual Communications Laboratory

Cointegration of the assets, pairs trading and the design of it by the-state-of-the-art signal processing methods are deeply studied to increase the profit of the investments. For further improvement, the optimisation of the statistical arbitrage is studied and designed.

Ground-Truth Aware Po-Ga Noise Parameters Estimation of Images Jun. 2022
École Polytechnique Fédérale de Lausanne, Computational Photography Project

The purpose is to propose a Poisson-Gaussian modeling for the raw-image of the sensors and propose an algorithm to solve the parameters of this model with the use of ground-truth images. Algorithms based on maximum likelihood solution are proposed. Source: `https://arxiv.org/abs/2210.12142`

Deep Learning Based Discomfort Glare Detection Dec. 2021
École Polytechnique Fédérale de Lausanne, Machine Learning Project

The goal is to use deep learning algorithms to detect discomfort glare from the human facial analysis. This is achieved by CNN architecture with statistical methods. This project is conducted under the supervision of LIPID, EPFL. Source: `https://github.com/CS-433/ml-project-2-dikaro_2`

Self-Monitoring for Symptoms Jun. 2021
Middle East Technical University, Design Project

A wearable device as a chest strap and mobile app are designed to monitor COVID-19 symptoms of the user and warn about the probability of having COVID-19. Cough, fever and heart rate of the user are automatically detected by the wearable device, while other symptoms are to be logged in via mobile app. The data transmission between the device and the mobile app is handled by BLE. Reports: `https://drive.google.com/drive/folders/1MaDxG5oqLDm_k3GUcDb-XVjIq039IVAj?usp=sharing`