Signal Processing Approach Using Hyperspectral Unmixing

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

IN

Artifical Intelligence and Machine Learning

Submitted by

K. Sai Keerthana - 222010322001

Ssrk Kasyap - 222010322005

K. Sharan Kumar - 222010322043

R. Ramkoteswara Rao -222010323002

Under the esteemed guidance of

Dr. Dibyajyoti Guha
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GITAM (Deemed to be University)

HYDERABAD

October - 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "**Signal Processing Approach Using Hyperspectral Unmixing**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering in Artificial Intelligence and Machine Learning. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 27-10-2023

Registration No(s).	Name(s)
222010322001	K. Sai Keerthana
222010322005	Ssrk Kasyap
222010322043	K. Sharan Kumar
222010323002	R. Ramkoteswara Rao

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM
(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled "**Signal Processing Approach Using Hyperspectral Unmixing**" is a bonafide record of work carried out by **K. Sai Keerthana (222010322001), Ssrk Kasyap (222010322005), K. Sharan Kumar (222010322043), R. Ramkoteswara Roa (222010323002)** students submitted in partial fulfilment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	Project Coordinator	Head of the Department
Dr. Dibyajyoti Guha **Associate Professor** **Dept. of CSE**	**Dr. Arshad Ahmad Khan** **Assistant Professor** **Dept. of CSE**	**Dr. K.S.Sudeep** **Associate Professor & HOD** **Dept. of CSE**

ACKNOWLEDGEMENT

Our project report would not have been successful without the help of several people. We would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honourable Pro-Vice-Chancellor, **Prof. D. Sambasiva Rao**,for providing the necessary infrastructure and resources for the accomplishment of our seminar.We are highly indebted to **Prof. N. Seetharamaiah**, Associate Director, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Dr. K.S.Sudeep** , Head of the Department of Computer Science & Engineering, for providing the opportunity to undertake this seminar and encouragement in the completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr Arshad Ahmad Khan**, Assistant Professor, Project Coordinator, Department of Computer Science and Engineering, School of Technology and to our guide, **Dr. Dibyajyoti Guha**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the project report.

We are also thankful to all the Computer Science and Engineering department staff members who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement, and moral support directly or indirectly in our seminar work.

Sincerely,
Our Team

Table of Contents

List Of Figures

1. Abstract

- Blind Hyperspectral Unmixing (HU) is a captivating research area within the signal processing domain, particularly in the context of hyperspectral remote sensing. This field is about unravelling the rich spectral information in hyperspectral images to identify materials within a scene and understand their compositions. Imagine it as solving a complex puzzle where each piece represents a different material, and you're trying to put them together to reveal the big picture.

- The roots of blind HU go back to the 1990s when hyperspectral sensing technology was making significant strides. This technological progress enabled researchers to explore and tackle the task of unmixing hyperspectral data. Over time, this research has transcended its origins in geoscience and remote sensing, piquing the interest of scholars from various disciplines, including machine learning and optimization. Consequently, it has become a vibrant and interdisciplinary research topic, bridging the gap between remote sensing and advanced signal processing.

- The fundamental problem in blind HU is essentially a Blind Source Separation (BSS) issue from a signal processing perspective. Researchers aim to tease apart the individual components in the hyperspectral data, like unmixing colours, to understand the pigments that make up a painting. This problem has paved the way for developing novel blind signal processing methods that differ from classical BSS approaches.

- One of the distinguishing features of blind HU is the application of convex geometry concepts, which were initially observed empirically by early remote sensing researchers and later refined by subsequent studies. These concepts are elegant and distinct from the statistical independence-based approaches traditionally used in signal processing. This divergence from established methods has opened up new avenues for exploration and innovation in the field.

- Furthermore, modern blind HU research embraces advanced techniques from sparse signal processing and optimization. This infusion of state-of-the-art methods drives the field towards convergence, where the distinctions between remote sensing-inspired ideas and advanced signal processing and optimization concepts are becoming increasingly blurred. Insights and techniques from both sides of this interdisciplinary divide are being harnessed to create more effective unmixing methods.

- There are several key developments, including:

  1. **Pure Pixel Search:** This involves the search for pixels in hyperspectral data that predominantly represent a single material. These "pure" pixels are essential for unmixing.

  2. **Convex Geometry:** This concept plays a crucial role in understanding the geometry of hyperspectral data and has been used to develop innovative unmixing techniques.

  3. **Dictionary-Based Sparse Regression:** This technique utilizes dictionaries of spectral signatures to model the materials in a scene, which aids in unmixing.

  4. **Nonnegative Matrix Factorization:** This mathematical method factors a hyperspectral data matrix into the constituent spectra and abundance fractions.

1. While these methods represent the core toolbox for unmixing hyperspectral data, it's important to note that Bayesian techniques also play a significant role in blind HU but are not covered in this article.
2. Our emphasis in this article is on providing insights into how each approach works and highlighting their significant results from the perspective of signal processing theory and methods. We will also discuss some of the latest advancements in this dynamic field. Remember that this article isn't intended as a comprehensive survey; for a more comprehensive overview, you can refer to a recent paper that covers many blind HU methods and other related aspects.

2. Introduction

- We critically explore an issue of great significance in computer vision and remote sensing: enhancing the spatial resolution of off-the-shelf hyperspectral sensors. This improvement can significantly benefit a range of typical computer vision tasks, including target tracking and image classification. The core idea underpinning our investigation revolves around harnessing the capabilities of two types of cameras: one equipped with a conventional RGB sensor and the other with a hyperspectral sensor.

- This study addresses the scenario in which these two cameras simultaneously capture the same scene. We endeavor to extract redundant and complementary information from their respective data streams. To facilitate this, we propose a novel framework strongly emphasizing two fundamental concepts: non-negativity and sparsity. These concepts form the foundation of our data-processing approach, which we will elucidate.

- Non-negativity is an essential assumption grounded in the observation that material properties inherently possess non-negative values. This assumption is critical for ensuring that our results are physically meaningful. We also incorporate the idea of sparsity, which arises from the practical understanding that only a few materials are typically present at each pixel location in a scene. This sparsity constraint is pivotal in enabling the unique reconstruction of hyperspectral images.

- In practical terms, we use a mathematical technique called "sparse non-negative matrix factorization" to implement our proposed framework. This method takes advantage of prior knowledge about spectral and spatial transform responses, an essential element in effectively integrating hyperspectral and RGB data. We advocate an alternating optimization strategy to tackle the complex problem at hand. This strategy breaks down the problem into subcomponents, each addressed through efficient convex optimization solvers, with the alternating direction method of multipliers (ADMM) serving as a prominent example of such a solver.

- To provide context, we recognize the limitations of hardware in directly obtaining high-resolution hyperspectral data. Instead of pursuing the costly path of acquiring such data now, our study explores the cost-effective alternative of estimating high-resolution hyperspectral images. We estimate this by drawing on lower-resolution hyperspectral observations (X) and high-resolution RGB data (Y). Additionally, we account for scenarios where the downsampling matrix (G) may not be perfectly known, a real-world challenge frequently encountered in practice.

- Our experimental findings, conducted across diverse scenes from a public database, are promising. They demonstrate that our proposed method consistently outperforms state-of-the-art techniques, irrespective of the degree of knowledge about the downsampling matrix (G). In essence, our approach has the potential to significantly enhance hyperspectral image processing and various computer vision tasks by capitalizing

on the detailed spectral information of hyperspectral cameras and the high spatial resolution of RGB sensors.

2.1 Problem Statement

- The problem at the heart of this research is the need to enhance the spatial resolution of hyperspectral images captured by sensors. Hyperspectral cameras offer rich spectral information but often at the cost of reduced spatial resolution. To overcome this limitation, the paper focuses on a scenario in which two types of cameras capture the same scene: one equipped with a conventional RGB sensor and the other with a hyperspectral sensor. The objective is to extract redundant and complementary information from these two data sources. The primary challenge is effectively integrating this data to produce a high-resolution hyperspectral image.

2.2 Objective Function

- The central objective of this research is to find a solution that integrates hyperspectral data (X) and high-resolution RGB data (Y) to estimate a high-resolution hyperspectral image (Z). This estimation is accomplished by solving a sparse non-negative matrix factorization problem, where the goal is to find two matrices, A and S:

    1. **A (Hyperspectral Material Basis Matrix) :** A size Mh x N matrix, where Mh is the number of hyperspectral bands, and N is the number of scene materials. The columns of A represent the hyperspectral basis vectors corresponding to different scene materials.

    2. **S (Hyperspectral Material Coefficients Matrix) :** A size N x Lc matrix, where Lc is the number of pixels in the desired high-resolution image. The columns of S represent the material coefficients present at each pixel in the high-resolution hyperspectral data Z.

    3. **The equation can represent the problem:**

        - **$Z = AS$**

- Here, Z represents the desired high-resolution hyperspectral image we aim to estimate. The task is to find the matrices A and S that best fit the observed data from X and Y. The challenge is to ensure that these matrices are both non-negative and sparse, adhering to the physical constraints and characteristics of real-world materials and scenes.

- The research employs a mathematical framework incorporating prior knowledge about spectral and spatial transformations (F and G) to find the best solution. The optimization of this problem is achieved using an alternating optimization approach, using efficient convex optimization solvers such as the alternating direction method of multipliers (ADMM).

- In summary, the objective function can be defined as finding matrices A and S that, when multiplied, provide a hyperspectral image Z while adhering to non-negativity and sparsity constraints. This optimization seeks to balance capturing the richness of spectral information and achieving high spatial resolution in hyperspectral images.

3 Algorithm Used

3.1 Non-Negative

- In this context, "non-negative" refers to the constraint that the elements in matrices or vectors should not have negative values. In hyperspectral imaging, it's a crucial assumption based on the physical characteristics of materials. Materials' spectral signatures represent how they reflect or emit light at different wavelengths and are inherently non-negative. Since hyperspectral imaging aims to capture these spectral signatures, the estimated material coefficients and spectral basis matrices (S and A) must also be non-negative.

- In practical terms, this non-negativity constraint ensures that the results of hyperspectral unmixing (the process of identifying materials in a mixed hyperspectral image) make physical sense. It's highly unlikely that materials could have negative abundance values, and enforcing this constraint helps achieve meaningful unmixing results.

3.1.1 Sparse

- In the context of this framework, "sparse" refers to a condition where most of the elements in a matrix or vector are zero or close to zero. This concept is applied to the material coefficients matrix (S) in hyperspectral unmixing. The assumption is that only a few materials are present at each pixel in a hyperspectral image. Therefore, the coefficients representing the abundances of these materials should be sparse, with most entries being zero.

- The sparsity constraint is grounded in the notion that real-world scenes typically comprise limited materials. Enforcing sparsity in the material coefficients is advantageous for several reasons. It helps find a unique and physically meaningful solution when solving the underdetermined equations arising from hyperspectral unmixing. Furthermore, it aids in reducing noise and making the unmixing process more robust, especially when dealing with mixed or complex scenes.

3.1.2 Non-Negative Sparse Promoting Framework

- The "non-negative sparse promoting" framework presented in the paper combines non-negativity and sparsity concepts in the context of hyperspectral image processing. It leverages the physical principles that materials' spectral properties are non-negative while recognizing that only a few materials are usually present at any pixel location.

- In this framework, the goal is to estimate both the material coefficients (S) and the spectral basis (A) matrices from lower-resolution hyperspectral data (X) and high-resolution RGB data (Y). The method enforces non-negativity and sparsity in these matrices to ensure the results are physically meaningful and suitable for unmixing complex scenes. The key to achieving this is a mathematical approach called "sparse non-negative matrix factorization."

- The framework integrates prior knowledge about spectral and spatial transformations (F and G) into the problem, and it utilizes alternating optimization techniques, such as the alternating direction method of

multipliers (ADMM), to efficiently solve the problem. This approach aims to promote the meaningful recovery of materials and their abundance in a scene while maintaining computational efficiency.

- In summary, the "non-negative sparse promoting" framework is a robust methodology in hyperspectral imaging that leverages the inherent non-negativity of material properties and the sparsity of materials at pixel locations to estimate material coefficients and spectral basis matrices in a manner that is both physically meaningful and computationally efficient. This framework contributes to the field of hyperspectral image processing and improves the accuracy of tasks like material identification and scene unmixing.

- Douglas-Rachford Splitting (DRS) is an iterative optimization technique used to solve convex optimization problems, especially in the context of convex optimization over the sum of two sets or functions. It is named after the mathematicians James Douglas and Henry Rachford, who independently developed this method. DRS is particularly valuable for solving problems where the objective function can be divided into two simpler convex functions, each associated with a specific set of variables. The method aims to find a point that minimizes the sum of these two functions.

3.2 Douglas-Rachford Splitting Method

- The Douglas-Rachford Splitting method is widely used in various fields, including signal processing, image reconstruction, machine learning, and mathematical optimization. It is well-suited for problems with a structure where the objective function can be decomposed into two convex functions but is challenging to optimize directly.

- The method's basic principle is to iteratively update the variables associated with each of the two functions while keeping the other set of variables fixed. The update procedure involves a combination of the proximal operator, which is a fundamental concept in convex analysis, and some simple arithmetic operations.
- Here's a simplified description of the DRS algorithm:
    1. Initialize the variables associated with each of the two functions.
    2. In each iteration, update one set of variables while keeping the other fixed.
    3. Repeat this process until convergence.
- The method is known for its convergence properties and its ability to efficiently solve complex convex optimization problems. It has applications in a wide range of fields, including:
    1. Image Reconstruction: DRS is used in medical imaging, tomography, and remote sensing to reconstruct high-quality images from incomplete or noisy data.
    2. Compressed Sensing: It plays a crucial role in recovering sparse signals from under-sampled measurements, a common problem in signal processing and data acquisition.
    3. Machine Learning: DRS is used in various machine learning tasks, such as support vector machines (SVM), LASSO regression, and more, to solve optimization problems arising in these domains.
    4. Inverse Problems: It helps solve inverse problems in physics, where parameters need to be estimated from indirect measurements.

- **Notations Involved**

1. The notations in the Douglas-Rachford Splitting method are typically associated with the specific convex optimization problem being solved. However, some common notations and terms used in the method include:
2. f(x): One of the convex functions to be minimized. It represents the first term in the objective function.
3. g(x): The other convex function to be minimized. It represents the second term in the objective function.
4. x: The variable to be optimized, which is typically partitioned into two sets.
5. Proximal Operator (prox): A mathematical operator denoted as prox(·), used to compute the proximal mapping of a function. This operator is central to the DRS algorithm.
6. Iteration: The DRS method is an iterative algorithm, so it involves updating the variables in each iteration.
7. Convergence: DRS aims to find a point where the sum of the two functions f(x) and g(x) is minimized. The method should converge to this point.
8. In summary, the Douglas-Rachford Splitting method is a powerful tool for solving convex optimization problems where the objective function can be decomposed into two simpler convex functions. It is widely used in various fields, and its success is attributed to its ability to efficiently handle complex problems through iterative updates of variables and the application of proximal operators. The specific notations used in DRS depend on the problem being addressed, but the key components involve the functions f(x) and g(x) and the variable x.

3.3 ADMM (Alternating Direction Method of Multipliers)

- The Alternating Direction Method of Multipliers (ADMM) is a powerful optimization technique used to solve a wide range of convex and non-convex optimization problems. ADMM is particularly valuable for problems with a structured or separable objective function, where the variables can be divided into distinct groups, each associated with a specific part of the objective.

- The basic idea behind ADMM is to decompose a complex optimization problem into simpler subproblems that can be efficiently solved. It employs an iterative approach, where at each iteration, it optimizes the variables associated with one part of the objective function while keeping other variables fixed, and then alternates to optimize the other set of variables. This alternating process continues until convergence is achieved.

- Here's a simplified description of the ADMM algorithm:

    1. Initialize the variables and Lagrange multipliers (dual variables).

    2. In each iteration, update one set of variables while keeping the other set fixed.

    3. Update the dual variables.

    4. Repeat this process until convergence.

- The central feature of ADMM is that it solves the original problem by iteratively solving simpler subproblems, each of which involves only a part of the objective function. The dual variables are updated to ensure that the iterates converge to the optimal solution.

3.3.1 Uses of ADMM

- ADMM is a versatile optimization method with a wide range of applications in various fields, including:

    1. **Machine Learning :** ADMM is used in various machine learning tasks, such as support vector machines (SVM), LASSO regression, and robust principal component analysis.

    2. **Signal Processing :** It is used in signal processing applications, such as image denoising, image reconstruction, and sparse signal recovery.

    3. **Convex Optimization :** ADMM can efficiently solve convex optimization problems with separable structures, including quadratic programming, linear programming, and more.

    4. **Statistics :** It is employed in statistical modeling and estimation problems, including constrained maximum likelihood estimation and Bayesian inference.

    5. **Control Systems :** ADMM has applications in control theory, distributed control, and optimal control problems.

3.3.2 Notations Involved in ADMM

- The notations in ADMM are closely tied to the specific optimization problem being solved. However, some common notations and terms used in ADMM include:

    1. **f(x) :** One of the convex functions to be minimized. It represents one part of the objective function.

    2. **g(z) :** The other convex function to be minimized. It represents another part of the objective function.

    3. **X and Z :** Variables to be optimized. The variables can be partitioned into distinct sets, with x associated with one part of the objective and z associated with the other.

    4. **Dual Variables (λ) :** Lagrange multipliers introduced to ensure that the iterates converge to the optimal solution.

    5. **Iteration :** ADMM is an iterative algorithm, so it involves updating the variables and dual variables in each iteration.

    6. **Rho (ρ) :** A positive parameter that controls the trade-off between the primal and dual updates in ADMM. It plays a critical role in algorithm convergence.

- ADMM is a versatile optimization method that excels in solving problems with structured or separable objectives. Its iterative approach to updating variables, along with the use of dual variables, makes it a powerful tool for addressing complex optimization problems in fields such as machine learning, signal processing, and control systems. The notations in ADMM depend on the specific problem at hand, but they typically include the functions f(x) and g(z), the variables x and z, the dual variables (λ), and the parameter ρ.

3.4 DRS vs ADMM

- **DRS:** Douglas-Rachford Splitting (DRS) and Alternating Direction Method of Multipliers (ADMM) are both optimization techniques used for solving complex problems. They share similarities but also have distinct characteristics. Let's compare them in detail:

  1. Basic Idea : DRS is an iterative optimization method primarily used for solving problems with the structure of the form: \(\min f(x) + g(x) \), where \(f \) and \(g \) are convex but not necessarily smooth functions.

  2. Algorithm : The DRS algorithm updates variables in a cyclic manner. In each iteration, it first projects onto the proximal operator of one function, then projects onto the proximal operator of the other function, effectively "splitting" the optimization problem into two subproblems.

  3. Convergence : DRS is known for its strong convergence properties, particularly for finding solutions to problems with convex but non-smooth functions. It can handle non-smooth regularizers effectively.

  4. Applications : DRS is widely used in signal processing, image reconstruction, machine learning, and various convex optimization problems.

- **ADMM**

  1. Basic Idea : ADMM is an optimization method designed for solving problems with multiple variables and multiple constraints. It is particularly useful for problems involving separable objectives or structured convex optimization.

  2. Algorithm : ADMM decomposes the problem into multiple smaller subproblems. It iteratively updates one variable while holding the others fixed. This alternation between variables and the use of a penalty parameter for constraints are key features of ADMM.

  3. Convergence : ADMM has strong convergence guarantees for a broad class of problems, including convex and non-convex ones. It often converges to the global optimum or a stationary point.

  4. Applications : ADMM is used in a wide range of fields, including machine learning, signal processing, image reconstruction, and optimization problems with constraints. It is particularly useful when the problem can be expressed in a way that allows for parallel or distributed computation.

- **Comparison**

  1. Decomposition : Both DRS and ADMM involve a decomposition of the problem into smaller subproblems. However, DRS is more suited for problems with two convex but not necessarily smooth functions, while ADMM is designed for multi-variable and multi-constraint problems.

  2. Convergence : Both methods have strong convergence properties, but ADMM is often preferred for problems with constraints and separable objectives. ADMM's convergence is well-understood and widely applicable.

  3. Applications : DRS is often used in image processing and non-smooth optimization problems, while ADMM is more versatile and widely used in machine learning, statistics, and various scientific and engineering disciplines.

  4. Complexity : ADMM can be more complex to implement due to its need to handle multiple variables and constraints. DRS is often simpler to implement for problems with two functions.

- In summary, both Douglas-Rachford Splitting and ADMM are powerful optimization methods, but they are best suited for different types of problems. DRS is ideal for problems with two convex but not necessarily smooth functions, while ADMM is more versatile and used in problems with multiple variables and constraints. The choice between them depends on the specific problem structure and requirements.

4. MATLAB

- MATLAB and Python are both powerful programming languages and environments used in various fields, especially in scientific and engineering applications. Each has its own strengths and is chosen based on specific needs and preferences. Let's discuss MATLAB and why it is chosen over Python in certain contexts:
  - MATLAB stands for "Matrix Laboratory." It is a proprietary high-level programming language and interactive environment developed by MathWorks. MATLAB is known for its extensive numerical and matrix computation capabilities. It provides a wide range of tools and functions for mathematical modeling, simulation, data analysis, and visualization. MATLAB is widely used in academia, research, and industry for various applications, including:

  1. **Mathematics and Numerical Analysis:** MATLAB excels in solving mathematical problems, differential equations, linear algebra, and numerical integration.

  2. **Signal Processing:** It is extensively used in signal processing tasks such as filtering, spectral analysis, and waveform generation.

  3. **Control Systems:** MATLAB is a primary choice for modeling, simulating, and designing control systems.

  4. **Image Processing :** It offers powerful image processing and computer vision toolboxes.

5. **Simulink:** A companion product to MATLAB, Simulink is used for modeling, simulating, and analyzing dynamic systems. It is widely used in control system design and simulations.

6. **Machine Learning:** MATLAB provides toolboxes for machine learning and deep learning, making it suitable for developing and experimenting with machine learning algorithms.

7. **Data Visualization:** MATLAB's plotting and visualization capabilities are well-regarded, allowing for the creation of complex graphs and figures.

5. UnLocBoX

- UnLocBoX, short for "Unsupervised Learning for Deconvolution and Demixing Toolbox," is an open-source MATLAB toolbox designed for solving various signal processing and inverse problem challenges. It focuses on unmixing and deconvolution problems, making it particularly useful for applications in imaging, spectroscopy, and signal processing. UnLocBoX provides a wide range of tools and algorithms for solving these problems. Below, I will provide a detailed description of UnLocBoX, its uses, notations involved, and what the toolbox contains:

- UnLocBoX is a versatile toolbox that primarily addresses problems related to signal deconvolution and unmixing. These problems typically involve the separation of mixed or convolved signals, making it essential in various domains, including:

    1. **Imaging:** It is used in image deblurring and restoration, where the goal is to recover a sharp image from a blurred or noisy version.

    2. **Spectroscopy:** It is applied in spectroscopic data analysis, where different spectral components need to be separated and identified.

    3. **Signal Processing:** UnLocBoX assists in blind source separation, image enhancement, and denoising applications.

5.1 Uses of UnLocBoX

- UnLocBoX serves a variety of applications, including:

    1. **Deconvolution:** It can be used to recover the original signal from a convolved or blurred version.

    2. **Spectral Unmixing:** It is helpful in separating and identifying different spectral components in a mixture.

    3. **Blind Source Separation:** UnLocBoX aids in separating mixed signals, such as separating sources in an audio recording.

    4. **Image Restoration:** It can restore images by removing noise, blur, or other degradations.

5.2 Notations Involved

- The notations used in UnLocBoX depend on the specific problem or algorithm being applied. Common notations may include:

    1. **f(x) :** Represents the objective function that needs to be minimized, often involving terms related to the data fidelity and regularization.

    2. **X :** The variable to be optimized or estimated. This could represent an image, spectrum, or source.

    3. **A :** Represents the system or mixing matrix, which describes the relationship between the observed data and the unknown variables.

    4. **D :** Regularization or penalty operator, which enforces certain constraints or promotes sparsity in the solution.

    5. **Λ :** Regularization parameter or Lagrange multiplier, controlling the trade-off between data fidelity and regularization.

5.3 What UnLocBoX Contains

- UnLocBoX contains a comprehensive set of tools and algorithms for solving signal deconvolution and unmixing problems. It includes a wide range of mathematical and optimization techniques, such as:

    1. **Regularization Methods :** UnLocBoX includes various regularization techniques like L1, L2, and total variation regularization.

    2. **Deconvolution Algorithms :** It offers algorithms for deconvolving signals, which are useful in image deblurring and spectral unmixing.

    3. **Optimization Solvers :** The toolbox includes various optimization solvers that are crucial for solving complex inverse problems efficiently.

    4. **Example Scripts :** UnLocBoX provides example scripts and demos to help users understand how to apply the toolbox to specific problems.

    5. **User-Friendly Interface:** It often features a user-friendly MATLAB interface with a clear structure for setting up and running algorithms.

- UnLocBoX is valuable because it simplifies the process of solving complex inverse problems and provides a toolbox of established algorithms that can be applied to a wide range of applications. It helps researchers and practitioners in fields like remote sensing, medical imaging, and signal processing to address real-world problems efficiently and accurately. Additionally, it encourages the development and exploration of new algorithms and techniques in signal processing and inverse problems.

- LTFAT (The Large Time-Frequency Analysis Toolbox) is a MATLAB toolbox designed for performing various time-frequency analysis and signal processing tasks. It provides a wide range of tools and algorithms for analyzing and processing signals in the time-frequency domain, making it particularly valuable for applications in audio and speech processing, music analysis, and more. Below, I will provide a detailed description of LTFAT, its uses, notations involved, and what the toolbox contains:

6. LTFAT (Large Time-Frequency Analysis Toolbox)

- The Large Time-Frequency Analysis Toolbox is an extensive collection of MATLAB functions and algorithms that focus on time-frequency analysis. It is especially useful for solving problems related to the analysis of signals in the time and frequency domains. LTFAT includes a wide variety of methods and tools for tasks such as:

    1. **Time-Frequency Transformations:** It provides various time-frequency analysis methods, including the Short-Time Fourier Transform (STFT), wavelet transforms, and the Gabor transform.

    2. **Audio and Speech Processing:** LTFAT is widely used in audio and speech signal processing for tasks like time-scale modification, source separation, and denoising.

    3. **Music Analysis:** It aids in music signal analysis, including tasks such as pitch estimation, musical note detection, and instrument recognition.

    4. **Signal Reconstruction:** LTFAT provides tools for signal reconstruction from time-frequency representations, such as inverse STFT and reconstruction using Gabor atoms.

6.1 Uses of LTFAT

- LTFAT serves various applications, including:

    1. **Audio Signal Processing:** It is used for audio coding, time-stretching, pitch-shifting, and audio restoration.

    2. **Music Analysis:** LTFAT is applied in music transcription, musical instrument recognition, and tempo estimation.

    3. **Speech Processing:** It helps in speech analysis, speech recognition, and speech enhancement.

    4. **Time-Frequency Analysis:** LTFAT is valuable for characterizing signals in both the time and frequency domains, allowing researchers to extract meaningful information from complex signals.

6.2 Notations Involved in LTFAT

- The notations used in LTFAT depend on the specific time-frequency analysis or signal processing task being performed. Common notations may include:

1. **f(t, f):** Represents the time-frequency representation of a signal, where "t" is time and "f" is frequency. This could be a spectrogram or another time-frequency representation.

2. **x(t):** The original signal in the time domain.

3. **X(f):** The signal's representation in the frequency domain (e.g., its Fourier transform).

4. **T:** Time domain representation or operator, which can be a transformation or a filter.

5. **F :** Frequency domain representation or operator, such as a Fourier transform or a filter.

6.3 What LTFAT Contains

- LTFAT contains a wide array of functions and algorithms for performing various time-frequency analysis and signal processing tasks. Some of the common components include:

    1. **Time-Frequency Transformations:** A collection of functions for performing time-frequency analysis using methods like the STFT, wavelets, and Gabor transform.

    2. **Signal Processing Tools:** LTFAT offers tools for audio and speech signal processing, including time-scale modification, source separation, and denoising.

    3. **Music Analysis Algorithms:** It includes algorithms for music analysis, such as pitch estimation, musical note detection, and instrument recognition.

    4. **Reconstruction Methods:** Functions for signal reconstruction from time-frequency representations, such as inverse STFT and reconstruction using Gabor atoms.

    5. **Demo Scripts:** LTFAT often provides demo scripts and example code to help users understand how to apply the toolbox to specific problems.

    6. **User-Friendly Interface:** It typically features a user-friendly MATLAB interface with well-documented functions and examples.

- LTFAT is valuable because it simplifies the process of performing time-frequency analysis and signal processing tasks, providing a toolbox of established algorithms and tools that can be applied to a wide range of applications in audio, music, speech, and signal processing. It helps researchers and practitioners address real-world problems efficiently and accurately. Additionally, it encourages the development and exploration of new algorithms and techniques in time-frequency analysis and signal processing.

7. Initialization of Unlocbox

```
function init_unlocbox()
%INIT_UNLOCBOX Initialize the toolbox
%   Usage: init_unlocbox()
%
%   Initialisation script for the unlocbox
%   This script add the different path needed to run the toolbox
%
%   Url: https://epfl-lts2.github.io/unlocbox-html/doc/init_unlocbox.html

% Copyright (C) 2012-2016 Nathanael Perraudin.
% This file is part of UNLOCBOX version 1.7.5
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <http://www.gnu.org/licenses/>.


% Author: Nathanael Perraudin
% Date: nov 2012


%% adding dependency
global GLOBAL_useGPU;
global GLOBAL_path;
GLOBAL_path = fileparts(mfilename('fullpath'));
GLOBAL_useGPU = 0;

addpath(genpath(GLOBAL_path));

% Load the version number
bp=[GLOBAL_path,filesep];
[FID, MSG] = fopen ([bp,'unlocbox_version'],'r');
if FID == -1
error(MSG);
else
unlocbox_version = fgetl (FID);
fclose(FID);
end

banner = sprintf(strcat(...
'UnLocBoX version %s. Copyright 2012-2015 LTS2-EPFL, by Nathanael Perraudin'), ...
unlocbox_version);
% display banner
disp(banner);

if GLOBAL_useGPU && gpuDeviceCount
```

```
dev=gpuDevice(1);
if dev.DeviceSupported
reset(gpuDevice);
disp('GPU loaded');
else
disp(['GPU not loaded.  To remove the previous warning, '...
'set GLOBAL_useGPU to 0']);
GLOBAL_useGPU=0;
end
end

kbstop('stop');
```

This script initializes the "UnLocBoX" toolbox by setting global variables, adding necessary paths to the MATLAB environment, loading the version number, displaying a banner, checking GPU usage, and making a function call. The script is designed to prepare the environment for using the "UnLocBoX" toolbox for optimization and signal processing tasks.

- The above code to be an initialization script for the "UnLocBoX" toolbox, which seems to be a software library for certain computational tasks. Here's a step-by-step explanation of the code:

- **Global Variable Initialization:**

1. global GLOBAL_useGPU; and global GLOBAL_path; declare two global variables, GLOBAL_useGPU and GLOBAL_path.
2. GLOBAL_path is assigned the directory path of the current script using fileparts(mfilename('fullpath')).
3. GLOBAL_useGPU is initialized to 0, indicating that GPU usage is initially disabled.
4. Adding Paths:
5. addpath(genpath(GLOBAL_path)); adds all subdirectories and files in the GLOBAL_path to the MATLAB path. This makes all necessary toolbox files and functions accessible in the current MATLAB session.

- **Loading the Version Number:**

1. The code attempts to open and read a file named 'unlocbox_version' in the script's directory.
2. If the file is successfully opened, it reads the version number from the file and stores it in the unlocbox_version variable.

- **Displaying Banner:**

1. The code creates a banner string that includes the version number and copyright information of the "UnLocBoX" library.
2. The banner is displayed using disp(), providing information about the "UnLocBoX" version and its copyright.

- **GPU Usage Check:**

1. The code checks whether GPU usage is enabled (GLOBAL_useGPU) and whether there is at least one GPU device available (gpuDeviceCount).
2. If GPU usage is enabled and a device is available, it attempts to reset the GPU device and displays a message indicating that the GPU is loaded.
3. If GPU usage is not supported or no GPU devices are available, it displays a warning message and sets GLOBAL_useGPU to 0, disabling GPU usage.
4. Calling 'kbstop' Function:
5. The code makes a function call to 'kbstop' with the argument 'stop.' The specific purpose and implementation of the 'kbstop' function are not provided in this code snippet.

8. Douglous rachford Algorithm

```
%% Initialisation

clear;
close all;

init_unlocbox;

verbose = 2;    % Verbosity level

%% Creation of the problem

% Original image
im_original = Shawdow();


% Creating the problem
A = rand(size(im_original));
A = A > 0.85;

% Depleted image
b = A .* im_original;
%b = im_original;
%% Defining proximal operators

% Define the prox of f2 see the function proj_B2 for more help
operatorA = @(x) A.*x;
epsilon2 = 0;
param_proj.epsilon = epsilon2;
param_proj.A = operatorA;
param_proj.At = operatorA;
param_proj.y = b;
param_proj.verbose = verbose - 1;
f2.prox=@(x,T) proj_b2(x, T, param_proj);
```

```
f2.eval=@(x) eps;

f2.prox = @(x,T) (x - A.*x) + A.*b;


% setting the function f1 (norm TV)
param_tv.verbose = verbose - 1;
param_tv.maxit = 50;

f1.prox = @(x, T) prox_tv(x, T, param_tv);
f1.eval = @(x) norm_tv(x);

%% solving the problem

% setting different parameter for the simulation
paramsolver.verbose = verbose;  % display parameter
paramsolver.maxit = 100;        % maximum iteration
paramsolver.tol = 10e-7;        % tolerance to stop iterating
paramsolver.gamma = 0.1;        % stepsize

% To see the evolution of the reconstruction
fig = figure(100);
paramsolver.do_sol = @(x) plot_image(x,fig);

sol = douglas_rachford(b,f1,f2,paramsolver);

close(100);

%% displaying the result
imagesc_gray(im_original, 1, 'Original image');
imagesc_gray(b, 2, 'Depleted image');
imagesc_gray(sol, 3, 'Reconstructed image');


%% Closing the toolbox
close_unlocbox();
```

- The above code performs image reconstruction from a depleted or corrupted version of the original image using the Douglas-Rachford splitting algorithm. Here's a step-by-step explanation of the code.

- This code initializes the '**unlocbox**' toolbox, creates a problem of image reconstruction from a depleted image, defines proximal operators for sparsity and range constraints, uses the Douglas-Rachford algorithm to solve the problem, displays the original, depleted, and reconstructed images, and then closes the toolbox.

- **Initialization:** The script starts by clearing the workspace and closing any open figures.

- **Loading the Toolbox:** The '**init_unlocbox'** function is called to initialize a toolbox for solving inverse problems with sparsity constraints.

- **Setting Verbosity Level:**The **'verbose'** variable is set to 2, indicating a moderate verbosity level for displaying messages during the execution.

- **Creating the Problem:**

  1. An original image (**'im_original'**) is obtained. However, there seems to be a typo in the code where "Shawdow()" should be replaced with a valid function or image file.
  2. A binary mask '**A**' is created with the same size as the original image. This mask is used to simulate missing or corrupted pixels. It is generated with a probability of 0.85.
  3. The depleted image '**b**' is created by element-wise multiplication of the binary mask '**A**' and the original image. This simulates the effect of missing or corrupted pixels, where missing pixels have a value of zero.

- **Defining Proximal Operators:**
  1. Two proximal operators, '**f1**' and '**f2**', are defined to promote sparsity and range constraints during image reconstruction.
  2. '**f2**'is related to a projection operation (**'proj_b2'**) using the binary mask '**A**'. It ensures that the image solution stays within the range of the mask. Parameters for the projection operation are set in the '**param_proj'** struct.
  3. '**f1**' is associated with a total variation (TV) regularization term, which promotes piecewise constant regions in the reconstructed image. The proximal operator '**prox_tv'** is used, and its evaluation is based on the TV norm of the image.

- **Solving the Problem:**
  1. Parameters for the solver are set, including the verbosity level, maximum iterations, tolerance for stopping iterations, and the step size.
  2. A figure ('**fig'**) is created for visualizing the solution during the optimization process.
  3. The '**douglas_rachford'** algorithm is used to solve the optimization problem with the given proximal operators ('**f1**' and '**f2'**) and solver parameters.

- **Displaying the Result:**
  1. The code uses the '**imagesc_gray**' function to display three images:
  2. The original image ('**im_original**') is displayed with the title "Original image."
  3. The depleted image ('**b**') is displayed with the title "Depleted image," showing the effect of missing or corrupted pixels.
  4. The reconstructed image ('**sol**') is displayed with the title "Reconstructed image." This is the result of the image reconstruction process.

- **Closing the Toolbox:** The script closes the '**unlocbox**' toolbox to ensure proper finalization.

- **Barbara Image**

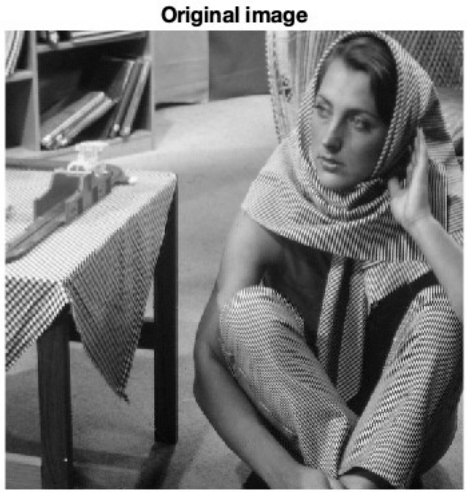

Fig 8.1.1: Barbara Image-1

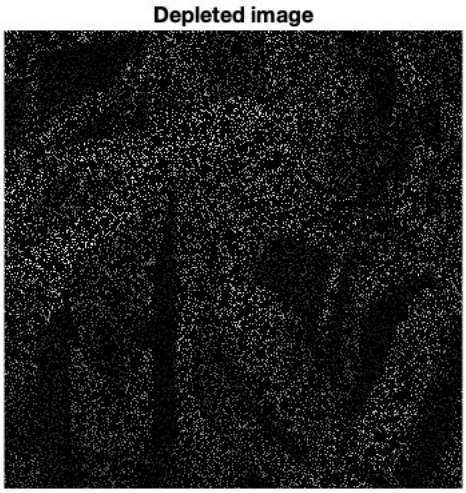

Fig 8.1.2 : Barbara Image-2

Fig 8.1.3: Barbara Image-3

- **Camera Man Image**

Fig 8.2.1: Camera Man Image-1

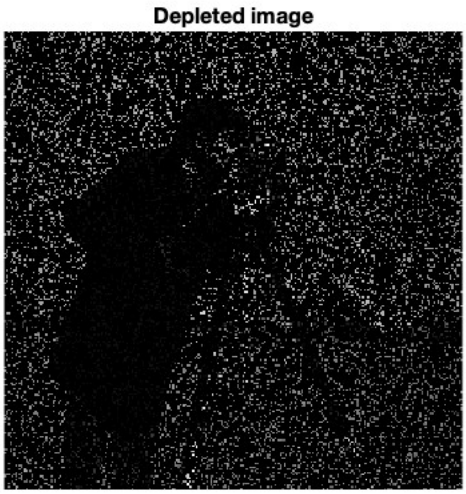

Fig 8.2.2: Camera Man Image-2

Fig 8.2.3: Camera Man Image-3

- **Checker Board Image**

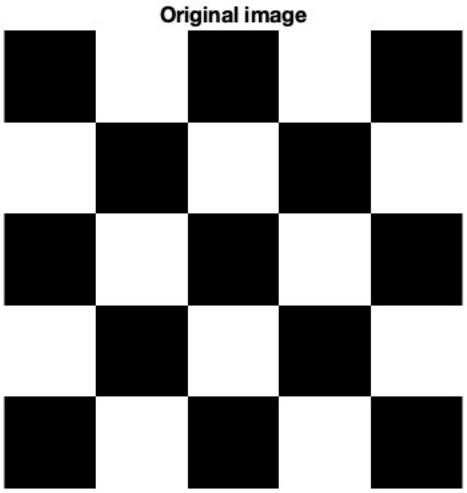

Fig 8.3.1: Checker Board Image-1

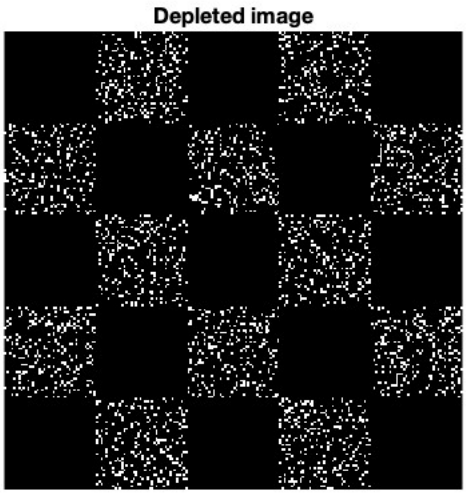

Fig 11.3.2: Checker Board Image-2

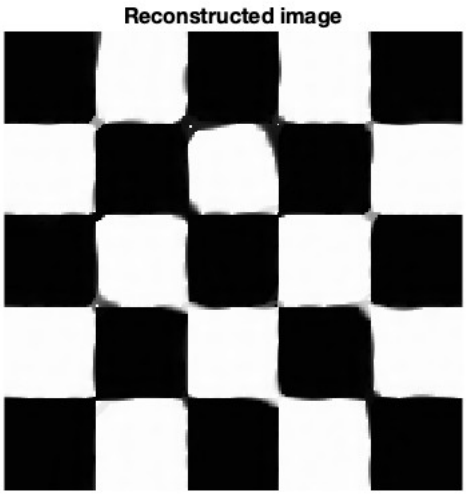

Fig 11.3.3: Checker Board Image-3

9. Initialization of LTFAT

```
function ltfatstart(varargin)
%LTFATSTART   Start the LTFAT toolbox
%   Usage:  ltfatstart;
%
%   `ltfatstart` starts the LTFAT toolbox. This command must be run
%   before using any of the functions in the toolbox.
%
%   To configure default options for functions, you can use the
%   |ltfatsetdefaults| function in your startup script. A typical startup
%   file could look like::
%
%     addpath('/path/to/my/work/ltfat');
%     ltfatstart;
%     ltfatsetdefaults('sgram','nocolorbar');
%
%   This will add the main LTFAT directory to you path, start the
%   toolbox, and configure |sgram| to not display the colorbar.
%
%   The function walks the directory tree and adds a subdirectory
%   to path if the directory contain a `[subdirectory,init.m]`
%   script setting a `status` variable to some value greater than 0.
%   `status==1` identifies a toolbox module any other value just a
%   directory to be added to path.
%
%   `ltfatstart(0)` supresses any status messages.
%
%   !!WARNING for MATLAB users!!
%   ----------------------------
%
%   The function indirectly calls `clear all`, which clears all your global
%   and persistent variables. It comes with calling `javaaddpath` in
%   blockproc/blockprocinit.m. You can avoid calling it by passing
%   additional `'nojava'` flag.
%
%   See also:  ltfatsetdefaults, ltfatmex, ltfathelp, ltfatstop

%   AUTHOR : Peter L. Søndergaard, Zdenek Prusa
%   TESTING: NA

%% PKG_ADD: ltfatstart(0);

do_java = 1;
ltfatstartprint=1;
if nargin>0
   scalarIds = cellfun(@isscalar,varargin);
   nojavaIds = strcmpi('nojava',varargin);
```

```
    if ~all(scalarIds | nojavaIds)
        error(['LTFATSTART: Only a single scalar and flag, '...
              '"nojava" are recognized']);
    end

    if any(nojavaIds)
        do_java = 0;
    end

    scalars = varargin(scalarIds);
    if numel(scalars)>1
        error('LTFATSTART: Only a single scalar can be passed.');
    elseif numel(scalars) == 1
        ltfatstartprint=scalars{1};
    end
end;

% Sometimes the run command used further does not return back to the
% current directory, here we explicitly store the current directory and
% cd to it at the end or is something goes wrong.
currdir = pwd;

% Get the basepath as the directory this function resides in.
% The 'which' solution below is more portable than 'mfilename'
% becase old versions of Matlab does not have "mfilename('fullpath')"
basepath=which('ltfatstart');
% Kill the function name from the path.
basepath=basepath(1:end-13);

pathCell = regexp(path, pathsep, 'split');
if ispc  % Windows is not case-sensitive
   bponPath = any(strcmpi(basepath, pathCell));
else
   bponPath = any(strcmp(basepath, pathCell));
end

if ~bponPath % To avoid recursion during pkg load
    addpath(basepath);
end

bp=basepath;

% Load the version number
[FID, MSG] = fopen ([bp,filesep,'ltfat_version'],'r');
if FID == -1
    error(MSG);
else
```

```
    ltfat_version = fgetl (FID);
    fclose(FID);
end

ignored_dirs_common = {[filesep(),'mat2doc'],...
                       [filesep(),'src']};

ignored_inits = {};
if ~do_java
    ignored_inits{end+1} = {'blockprocinit.m',1};
end

%% --- Check for old versions of Octave and Matlab
if isoctave
   major_rq=3;
   minor_rq=6;
   intp='Octave';
   req_versionname='3.6.0';
   ignore_dirs = [{[filesep(),'mex']},...
                  ignored_dirs_common];
else
   major_rq=7;
   minor_rq=9;
   intp='Matlab';
   req_versionname='2009b';
   ignore_dirs = [{[filesep(),'oct']},...
                  ignored_dirs_common];
end;

% Split into major and minor version
s=version;
stops=find(s=='.');
major_no  = str2num(s(1:stops(1)));
if numel(stops)==1
  minor_no  = str2num(s(stops(1)+1:end));
  bugfix_no = 0;
else
  minor_no  = str2num(s(stops(1)+1:stops(2)));
  bugfix_no = str2num(s(stops(2)+1:end));
end;

% Do the check, multiply by some big number to make the check easy
if major_rq*1000+minor_rq>major_no*1000+minor_no
  warning(['Your version of %s is too old for this version of LTFAT ' ...
        'to function proberly. Your need at least version %s of %s.'],...
        intp,req_versionname,intp);
end;
```

```
%% -----------  install the modules -----------------

modules={};
nplug=0;

% List all files in base directory
d=dir(basepath);

% Pick only valid directories and wrap it in a cell array
d={d(arrayfun(@(dEl) dEl.isdir && ~strcmp(dEl.name(1),'.'),d))};
basedir = {filesep};

while ~isempty(d)
  for ii=1:length(d{1})
    name=d{1}(ii).name;

    % Skip ignored directories
    if any(cellfun(@(iEl) strcmp([basedir{1},name],iEl),ignore_dirs))
      continue;
    end

    % Store only valid subdirectories, we will go trough them later
    dtmp = dir([bp,basedir{1},name]);
    dtmp = dtmp(arrayfun(@(dEl) dEl.isdir && ~strcmp(dEl.name(1),'.'),dtmp));
    if ~isempty(dtmp)
      d{end+1} = dtmp;
      % Store base directory too
      basedir{end+1} = [basedir{1},name,filesep];
    end

    % The file is a directory and it does not start with '.' This could
    % be a module
    initfilename = [lower(name),'init.m'];
    initfilefullpath = [bp,basedir{1},name,filesep,initfilename];
    if ~exist(initfilefullpath,'file')
      continue
    end;

    % Now we know that we have found a module

    % Set 'status' to zero if the module forgets to define it.
    status=0;

    module_version=ltfat_version;

    % Add the module dir to the path
    addpath([bp,basedir{1},name]);
```

```
    iffound = cellfun(@(iEl) strcmpi(initfilename,iEl{1}),ignored_inits);

    if any(iffound)
        status = ignored_inits{iffound}{2};
    else
        % Execute the init file to see if the status is set.
        % We are super paranoid co we wrap the call to a try block
        try
            run(initfilefullpath);
        catch
            % If the run command breaks, it might not cd back to the
            % original directory. We do it manually here:
            cd(currdir);
        end
    end

    if status>0
      % Only store top-level modules
      if status==1 && strcmp(basedir{1},filesep)
        nplug=nplug+1;
        modules{nplug}.name=name;
        modules{nplug}.version=module_version;
      end;
    else
      % Something failed, restore the path
      rmpath([bp,basedir{1},name]);
    end;
  end;
  % Remove the just processed dir from the list
  basedir(1) = [];
  d(1) = [];
end


% Check if Octave was called using 'silent'
%if isoctave
%  args=argv;
%  for ii=1:numel(args)
%    s=lower(args{ii});
%    if strcmp(s,'--silent') || strcmp(s,'-q')
%      printbanner=0;
%    end;
%  end;
%end;

if ltfatstartprint
  try
```

```
    s=which('comp_pgauss');
    if isempty(s)
      error('comp_pgauss not found, something is wrong.')
    end;

    if strcmp(s(end-1:end),'.m')
      backend = 'LTFAT is using the script language backend.';
    else
      if isoctave
        backend = 'LTFAT is using the C++ Octave backend.';
      else
        backend = 'LTFAT is using the MEX backend.';
      end;
    end;
  catch
    backend = 'Error with backend, consider running "ltfatmex clean" immediately.';
  end;

  banner = sprintf(['LTFAT version %s. Copyright 2005-2023 Peter L. Søndergaard. ' ...
                    'For help, please type "ltfathelp". %s'], ...
                   ltfat_version,backend);

  disp(banner);

  if ~isoctave() && do_java
     disp('(Your global and persistent variables have just been cleared. Sorry.)');
  end

  if exist('ltfat_binary_notes.m','file')
    ltfat_binary_notes;
  end;

end;

if isoctave()
    % On Windows the run command might not change back to the original path
    cd(currdir);
end

%% ---------- load information into ltfathelp ------------
clear ltfatarghelper;
% As comp is now in the path, we can call ltfatarghelper
ltfatsetdefaults('ltfathelp','versiondata',ltfat_version,...
                 'modulesdata',modules);

%% ---------- other initializations ---------------------

% Force the loading of FFTW, necessary for Matlab 64 bit on Linux. Thanks
```

```
% to NFFT for this trick.
fft([1,2,3,4]);
```

- The above code is an initialization script for the LTFAT (Large Time-Frequency Analysis Toolbox) toolbox. It sets up the toolbox, adds relevant directories to the MATLAB path, and performs various checks. Here's a step-by-step explanation of the code.

- This script prepares the LTFAT toolbox for use, adds relevant directories to the path, and ensures compatibility with the user's MATLAB or Octave version. It also displays a version banner and other relevant information.

- **Function Header:** The script defines a function called ltfatstart that initializes the LTFAT toolbox. It can be used to configure default options for functions within the toolbox.

- **Input Arguments:** The function can accept an optional scalar input argument to control verbosity. If not provided, it defaults to 1, meaning that it will display status messages. If the argument is 0, it suppresses status messages.

- **Path Initialization:** The script sets up initial variables do_java and ltfatstartprint based on the input arguments. The do_java variable controls whether Java should be used (it's enabled by default).

- **Current Directory Saving:** The current directory is stored in the currdir variable to ensure it can be restored at the end of the script. This is done because certain commands may change the current directory.

- **Determine Base Path:** The base path of the script is determined, which is the directory containing the ltfatstart function.

- **Version Number Loading:** The script reads the version number of the LTFAT toolbox from a file named 'ltfat_version' in the toolbox directory.

- **Platform and Version Compatibility Checks:** The script performs checks to ensure compatibility with the user's MATLAB or Octave version. It compares the version of the interpreter with the required version and issues a warning if the interpreter version is older than required.

- **Initialization of Modules:**
  1. The script scans the toolbox directory for subdirectories that contain an init.m script. Each subdirectory with an init.m script is considered a module. The init.m script can set a status variable to indicate its status.
  2. The script adds the module directories to the MATLAB path and executes the init.m script to determine the module's status.
  3. The module information (name and version) is stored in the modules array for top-level modules.

- **Banner Display:**A banner is generated with version information and backend details and is displayed to the user.

- **Clearing Variables (for MATLAB):** In MATLAB, a message is displayed that warns the user that global and persistent variables have been cleared. This is necessary to ensure proper initialization.
- **Setting Help Data:** The script clears ltfatarghelper and sets data for the LTFAT help system.
- **Other Initializations:** The script forces the loading of FFTW (Fastest Fourier Transform in the West), which is necessary for MATLAB 64-bit on Linux.

10. ADMM

```
%% Initialisation

clear;
close all;

% Loading toolbox
init_unlocbox();
ltfatstart();

verbose = 2;    % verbosity level

% Regularization parameter: weight of the fielity term
tau = 50;
% Noise level
sigma = 0.1;
% Percent of missing pixels
p = 50;

%% Defining the problem

% Original image
im_original = barbara();

% Depleted image
mask = rand(size(im_original))>p/100;
z = mask .* im_original + sigma * rand(size(im_original));

%% Defining proximal operators

% Define the wavelet operator
L = @(x)  fwt2(x,'db8',6);
Lt = @(x)  ifwt2(x,'db8',6);

% setting the function tau * || Mx - y ||_2^2
f1.proxL = @(x, T) (1+tau*T*mask).^(-1) .* (Lt(x)+tau*T*mask.*z);
```

```
f1.eval = @(x) tau * norm(mask .* x - z)^2;

% setting the function || L x ||_1 using ADMM to move the operator ot of
% the proximal
param_l1.verbose = verbose - 1;
f2.prox = @(x, T) prox_l1(x, T, param_l1);
f2.eval = @(x) norm(L(x),1);
f2.L = L;
f2.Lt = Lt;
f2.norm_L = 1;


%% solving the problem

% setting different parameter for the solver
paramsolver.verbose = verbose;     % display parameter
paramsolver.maxit = 100;           % maximum number of iterations
paramsolver.tol = 1e-3;            % tolerance to stop iterating
paramsolver.gamma = 1;             % stepsize
% Activate debug mode in order to compute the objective function at each
% iteration.
paramsolver.debug_mode = 1;
fig=figure(100);
paramsolver.do_sol=@(x) plot_image(x,fig);

sol = admm(z, f1, f2, paramsolver);

%% displaying the result
imagesc_gray(im_original, 1, 'Original image');
imagesc_gray(z, 2, 'Depleted image');
imagesc_gray(sol, 3, 'Reconstructed image');


%% Closing the toolbox
close_unlocbox();
```

- This code to perform image reconstruction using a sparse regularization technique, specifically using the Alternating Direction Method of Multipliers (ADMM). Here's a step-by-step explanation of the code.

- The above code sets up a sparse image reconstruction problem, defines proximal operators for the data fidelity and sparsity regularization terms, and uses the ADMM algorithm to solve the optimization problem. It then displays the original, degraded, and reconstructed images.The **'unlocbox'** and '**ltfat'** toolboxes are utilized for various operations in this image processing task.

- **Initialization:** The script begins by clearing the workspace and closing any open figures.

- **Loading Toolboxes:** It loads two toolboxes: unlocbox and ltfat. unlocbox seems to be a toolbox for solving inverse problems with sparsity constraints, and ltfat is the Large Time-Frequency Analysis Toolbox.

- **Parameter Setup:** Several parameters are set are
  1. '**verbose**': A verbosity level for displaying messages.
  2. **tau**: Regularization parameter, which controls the weight of the fidelity term in the objective function.
  3. **sigma**: Noise level in the image.
  4. **p**: The percentage of missing pixels in the original image.

- **Defining the Problem:** The code defines the problem by creating a degraded image from an original image. It uses a random binary mask to simulate missing pixels and adds random noise to it.
  1. **'im_original'** is the original image.
  2. **'mask'** is the binary mask indicating missing pixels.
  3. **'z'** is the degraded image with missing pixels and noise.

- **Defining Proximal Operators:**
  1. The code defines two proximal operators and associated evaluation functions.
  2. The first operator (**'f1'**) handles the data fidelity term and is responsible for reconstructing the image. It uses the '**mask**' and '**z**' information.
  3. The second operator (**'f2'**) deals with sparsity regularization using the wavelet transform.
  4. The **'L'** and **'Lt'** functions represent the wavelet operator and its adjoint (inverse wavelet transform).
  5. The **'norm_L'** parameter is set to 1, indicating the regularization term's weight.

- **Solving the Problem:**
  1. Parameters for the solver are set, including the maximum number of iterations (**'maxit'**), tolerance to stop iterating (**'tol'**), step size (**'gamma'**), and debug mode for computing the objective function.
  2. The **'sol'** variable stores the result of solving the optimization problem using the ADMM method. The **'admm'** function is used to solve the problem with the defined proximal operators and solver parameters.
  3. During the solution process, the **'plot_image'** function is called at each iteration to display the current solution. It appears that the solution progress can be visualized.

- **Displaying the Result:** The script uses the **'imagesc_gray'** function to display the original image, degraded image, and reconstructed image. Each image is displayed with a title.

- **Closing the Toolbox:** The script closes the **'unlocbox'** toolbox. This ensures that the toolbox is properly finalized and any resources are released.

- **Barbara Image**

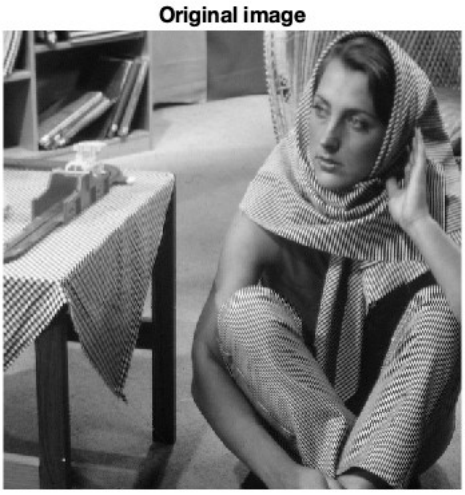

Fig 10.1.1: Barbara Image-1

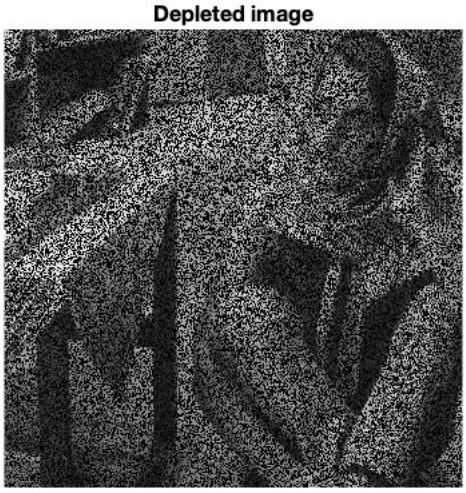

Fig 10.1.2: Barbara Image-2

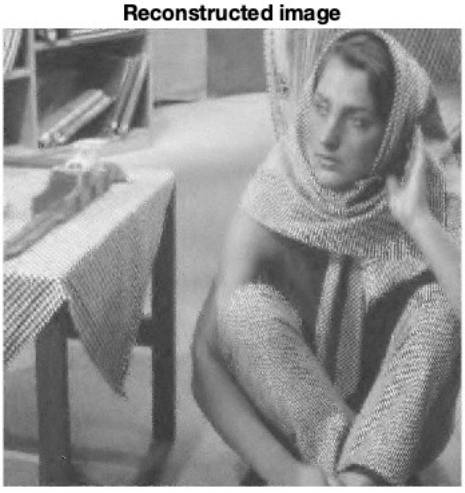

Fig 10.1.3: Barbara Image-3

- **Checker Board Image**

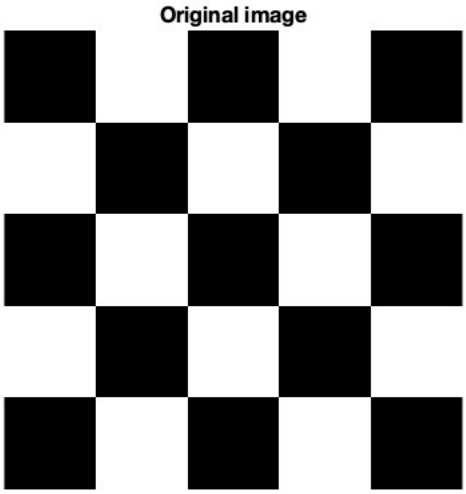

Fig 10.2.1: Checker Board -1

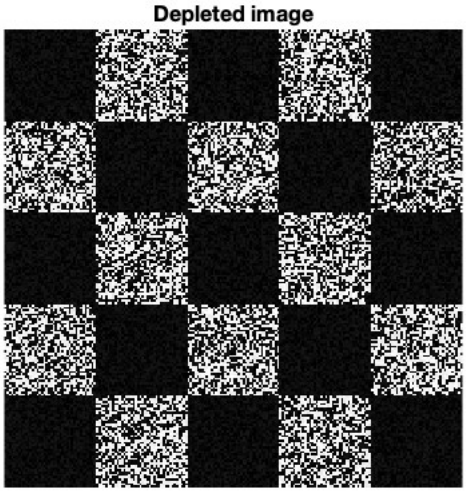

Fig 10.2.2: Checker Board -2

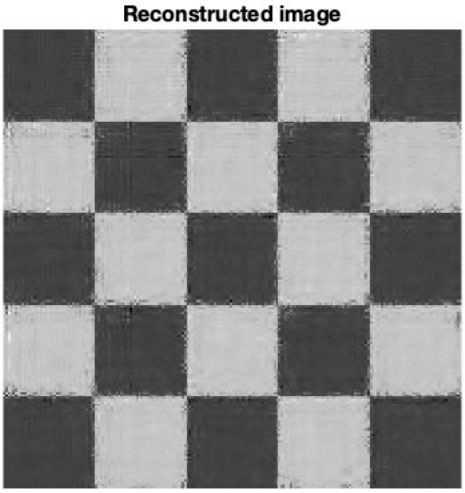

Fig 10.2.3: Checker Board -3

- **Camera Man Image**

Fig 10.3.1: Camera Man-1

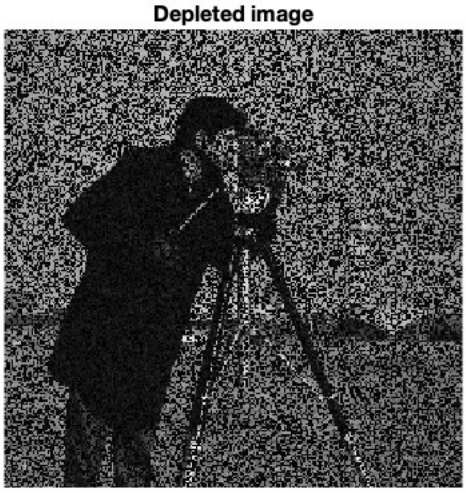

Fig 10.3.2: Camera Man-2

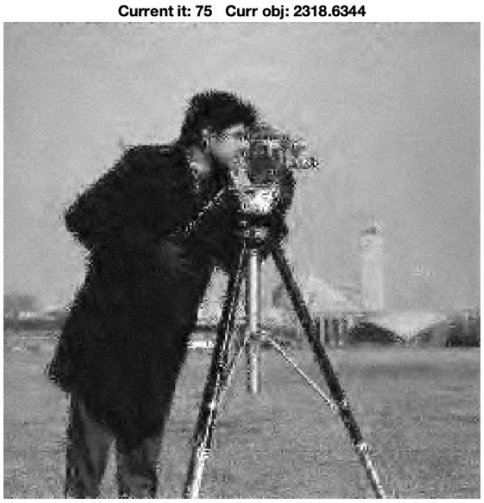

Fig 10.3.3: Camera Man -3

11. Future Prospects

- We are introducing a new weighted variational model for the estimation of reflectance and illumination in images. Unlike the commonly used log-transformed images, our model is designed to offer better performance by incorporating a more suitable prior representation in the regularization terms. What sets our approach apart is its ability to retain finer details in reflectance and to partially mitigate noise. We employ an alternating minimization technique to solve this model, and our experiments confirm its effectiveness. When compared to traditional variational methods, our proposed approach consistently achieves comparable or superior results in both subjective and objective evaluations.

12. References

1. A Weighted Variational Model for Simultaneous Reflectance and Illumination Estimation Xueyang Fu; Delu Zeng; Yue Huang; Xiao-Ping Zhang; Xinghao Ding [1]

2. A non-negative sparse promoting algorithm for high resolution hyperspectral imaging Eliot Wycoff; Tsung-Han Chan; Kui Jia; Wing-Kin Ma; Yi Ma [2]

3. High-resolution hyperspectral imaging via matrix factorization Rei Kawakami**;** Yasuyuki Matsushita; John Wright; Moshe Ben-Ezra; Yu-Wing Tai; Katsushi Ikeuchi [3]

4. On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators Jonathan Eckstein & Dimitri P. Bertsekas [4]

5. A Weighted Variational Model for Simultaneous Reflectance and Illumination Estimation Xueyang Fu; Delu Zeng; Yue Huang; Xiao-Ping Zhang; Xinghao Ding [5]

6. Removing Rain from Single Images via a Deep Detail Network Xueyang Fu; Jiabin Huang; Delu Zeng; Yue Huang; Xinghao Ding; John Paisley [6]