“Compressed Sensing for Electron Microscope Data”

April 7, 2011

Master of Science Thesis in Image Processing

Marc Manel Vilà Oliva
mmvo@kth.se

Supervisor: Hamed Hamid Muhammed
Examiner: Fredrik Bergholm

Abstract

Recent work in the field of Compressed Sensing [1, 2, 3] has suggested a great variety of new possibilities in the world of image reconstruction. We have been focused on a novel approach using this kind of algorithms based on CS to solve problems related to limited-angle data (e.g, computed tomography or electron microscope) or when we only dispose from few radon projections, low frequencies of an image, model or figure.

This approach has been based on a variation of the Robbins-Monro stochastic approximation procedure [4] with regularization enabled by a spatially adaptive filter.

The idea consists in exciting an algorithm by injection of random noise in the unobserved portion of the image spectrum and a spatially adaptive image denoising filter, working in the image domain, is exploited to attenuate the noise and reveal new features and details out of the incomplete and degraded observations of the model.

We developed the algorithm and we tested with some variations of the Shepp-Logan phantom[1] and Hansandrey crystallography[2], to prove its viability in an empirical way before applying it to real cases[3]. Our idea after the tests was to apply this procedure in a reconstruction of a 3D-protein crystallography taken in a TEM (Transmission Electronic Microscopy) with limited-angle views that can lead to have missing wedges or cones in the final results.

[1]2-dimensional case

[2]artificially created 3-dimensional model

[3]A crystallography with limited-angle views and a viral structure where we have forced the missing cone

Acknowledgments

Primer de tot vull dedicar aquest projecte final de carrera a la meva tieta Quitèria que recentment ens va deixar i degut a la distància que em trobo de casa no vaig poder anar a dir-li un últim adéu. Donar les gràcies al meu tutor, Hamed, ja que sense ell no hauria tingut aquesta gran oportunitat de realitzar aquest projecte en terres sueques i disfrutar d'aquesta experiència. També recordar al Philip, ja que se'l pot considerar com un segon tutor, sense ser-ho oficialment m'ha dedicat el seu temps i saber. Gràcies Philip. Vull fer un especial nombrament a la meva família, pare, mare, Montsin, "yayes" Pilarin i Manuela, tiets, fillola, etc. que segurament m'han trobat a faltar molt encara que no ho diguin i m'han donat el seu suport en tots els sentits per poder arribar a aquest punt de finalitzar la meva etapa universitària. Per finalitzar demanar-li perdó al senyor Ferran, perqué segurament he estat una persona molt pesada demanant-li consell en tots els petits i grans dubtes que he tingut durant el projecte i que ell molt amablement m'ha ajudat. Gràcies Ferrins. Tampoc em vull deixar als meus amics de la carrera, amb els quals he compartit molts anys a les aules i per Barcelona, gràcies Albert, Eric, Molina, Raúl, Sergi, Didac, Waiki, Aitor...I bueno als meus ex-"jefes" Albert Gil i Josep Ramón, igualment que al Josep Pujal, gràcies per tot el que em vau ensenyar i jo vaig intentar apendre. I nooo Isa, no m'oblido de tú, xD. La frase "ooh!!! You are always working..." l'he sentit molts cops durant els últims mesos, se que he estat treballant més del compte alguns moments i ho sento, però et vull donar les gràcies per haver-me aguantat i recordar-te que tú també formes part d'aquest projecte final de carrera.

First of all I want to dedicate this Master Thesis to my aunt Quitèria who recently passed away and due to the distance I could not go to tell her one last goodbye. To thank my tutor, Hamed, because without him I would not have had this great opportunity to do this project on Swedish land and enjoy this great experience. Also remember to Philip, he could be considered as a second supervisor, although he is not officially but he has shared his time and knowledge with me. Thanks Philip. I want to make a special mention to my family, dad, mom, Montsin, "yayes" Pilarín and Manuela, uncles, goddaughter, etc.. they have surely missed me a lot but they have not told me anything and they have been supportive in every way to reach the end of my time at university. Finally I want to say sorry to Ferran, probably because I have been a very annoying person asking for advice on all big or small doubts that I had during the project and he very kindly helped me. Thanks Ferrins. Nor do I want to forget my friends from university with whom I shared many years in classrooms and in Barcelona, thanks Albert, Eric, Molina, Raúl, Sergi, Tote, Didac, Waiki, Aitor...and to my ex-"bosses" Albert Gil and Josep Ramon, also Josep Pujal, thanks for everything you taught me and I tried to learn. And nooo Isa, I do not forget you, xD. The phrase "ooh!!! You are always working..." I have heard it many times in recent months, I know that I have been working over time and I want to say sorry but I want to thank you for having endured me and remind you that you also form part of this Master Thesis.

Contents

1 Introduction **1**
- 1.1 Inverse Problems in Electron Microscope . . . 1
- 1.2 Limitations . . . 2
- 1.3 Why this approach is needed? . . . 2
- 1.4 Objectives . . . 3
- 1.5 Overview of the thesis . . . 3

2 Background **4**
- 2.1 Compressed Sensing . . . 4
- 2.2 Image Reconstruction Techniques . . . 6
- 2.3 The Data . . . 10
  - 2.3.1 Shepp-Logan Phantom . . . 13
  - 2.3.2 Protein Crystallography Data . . . 14
- 2.4 Statement of the Question or Problem . . . 18

3 Method **20**
- 3.1 Compressed Sensing Image Reconstruction Via Recursive Spatially Adaptive Filtering . . . 20
- 3.2 Filtering . . . 21
- 3.3 How to apply in 3D models . . . 24

4 Implementation **25**
- 4.1 Algorithm . . . 25
- 4.2 Matlab Functions . . . 33

5 Analysis **36**
- 5.1 Shepp-Logan Phantom . . . 37
- 5.2 Peppers and Cameraman . . . 41
- 5.3 Real Data . . . 45
  - 5.3.1 Hansandrey model . . . 46
  - 5.3.2 First model . . . 54
  - 5.3.3 Viral DNA gatekeeper . . . 56

6 Conclusions and future work **59**
- 6.1 Conclusions . . . 59
- 6.2 Future Work . . . 61

7 Bibliography **62**

A Appendix A: Matlab code **64**

B Appendix C: Standard Deviation graphics **79**

C Appendix C: Reconstruction of the Shepp-Logan Phantom **80**

D Appendix D: 3D-data **81**

E Appendix E: Processing Time **86**

List of Figures

2.1 Limited-angle data . . . 7
2.2 Shepp-Logan phantom . . . 11
2.3 FFT Shepp-Logan phantom . . . 11
2.4 Transmission Electron Microscopy . . . 13
2.5 Reconstructions of a.22lines b.11lines c.90º missing data . . . 14
2.6 a.fft_22projections b.fft_11projections c.fft_90º_missing . . . 14
2.7 First protein model, protein.mrc . . . 15
2.8 Hansandrey model, hansandrey.raw . . . 16
3.1 Filtering scheme . . . 22
4.1 Known data (white) and unknown data (black) . . . 26
4.2 Reconstruction 90º degrees missing data . . . 27
4.3 Masks for: a.22projections b.11projections c.90º missing data . . . 31
4.4 Masks: a.128pix b.64pix . . . 31
4.5 3D cone mask . . . 32
4.6 3D wedge mask . . . 32
4.7 Masks: a.xy-slice b.xz-slice . . . 33
4.8 Method diagram . . . 35
5.1 Reconstructions of a.22projections b.11projections c.missing cone . . . 37
5.2 FFT 22 projections . . . 37
5.3 Reconstruction FFT 22 projections . . . 38
5.4 Evolution of the algorithm case a. . . . 38
5.5 FFT 11 projections . . . 39
5.6 Reconstruction FFT 11 projections . . . 39
5.7 Evolution of the algorithm case b. . . . 39
5.8 90º missing data . . . 40
5.9 Reconstruction 90º missing data . . . 40
5.10 Evolution of the algorithm case c. . . . 41
5.11 Peppers and Cameraman . . . 42
5.12 Applied masks: a.128pix b.64pix . . . 42
5.13 Modified peppers and cameraman by mask 128×128 . . . 43
5.14 Reconstruction of modified peppers and cameraman by mask 128×128 . . . 44
5.15 Modified peppers and cameraman by mask 64×64 . . . 45
5.16 Reconstruction of modified peppers and cameraman by mask 64×64 . . . 45
5.17 Hansandrey missing wedge . . . 46
5.18 Hansandrey appearance missing wedge . . . 47
5.19 a.Hansandrey original model b.Hansandrey reconstruction . . . 48
5.20 Hansandrey spectrum reconstruction . . . 49
5.21 Hansandrey missing cone . . . 50
5.22 a.Hansandrey original model b.Hansandrey reconstruction xy-slices . . . 52
5.23 Hansandrey worst case: a.xz slices b.xy slices . . . 53
5.24 a.Hansandrey original model b.Hansandrey reconstruction xz-slices . . . 54
5.25 First model proposal . . . 55
5.26 Comparison between first model and our proposal . . . 56
5.27 Viral DNA gatekeeper (real model) . . . 57

5.28 Viral DNA gatekeeper (missing cone model) 57
5.29 Viral DNA gatekeeper (reconstruction) 58
B.1 Standard deviation projections case 79
B.2 Standard deviation 90º case 79
B.3 Standard deviation low frequency case 80
B.4 Standard deviation 3D case 80
C.1 Shepp-Logan reconstruction. Slices 1, 1000, 2000..., 19000. 81
D.1 Hansandrey xy-slices 15, 20...85, 90. 82
D.2 Hansandrey reconstructed xy-slices 15, 20...85, 90. 82
D.3 Hansandrey xz-slices 15, 20...85, 90. 83
D.4 Hansandrey reconstructed xz-slices 15, 20...85, 90. 83
D.5 Protein slice #26 and #29 84
D.6 protein slice #32 and #35 84
D.7 Protein slice #38 and #41 84
D.8 Protein slice #44 and #47 85
D.9 Protein slice #50 and #53 85
E.1 Processing Time by pixels 86

List of Tables

2.1 Statistic of 3D-data “protein.mrc” 17
2.2 Statistics of 3D-data “hansandrey.raw” 18
4.1 Wavelets combination 28
4.2 Computational Time 29
4.3 α, β values 30
4.4 Noise Amplitude 30
5.1 Missing cone PSNR values of xy slices 51
E.1 Processing time 86

1 Introduction

1.1 Inverse Problems in Electron Microscope

Inverse problems is the aim of a large group of investigators, it is the necessity to solve inverse problems. An inverse problem as we know is the task that often occurs in many branches of science and mathematics where the values of some result, value, model...must be obtained from the observed data.

A common problem in CT[4], other X-ray techniques, TEM[5]...is the limited-angle view. Our case is similar to the third case. We have a 3D crystallography, taken in one of the TEMs which we can be found in KI facilities. Specifically in this model, we can observe a missing/corrupted cone of data in the frequency domain.

But, *what does a limited-angle problem mean?*

Limited-angle problems mean that from a certain angle the data is missing or corrupted, we cannot trust in those results. We have to estimate the missing data. In this case, we have a missing/corrupted 90º cone in the spectrum data. This problem is due to various technical and fundamental limitations on the minimum and maximum attainable tilt angles of our instrument to take the data. However, the data is confined to a limited angular range. We have to mention that if nobody can find a solution for this specific case, the crystallography was done for almost nothing. Actually, it was done with the side-effect to damage the specimen, because in each exposition of a specimen to a TEM, the collision of the electrons damage the specimen. This is one of the main reasons why it is necessary to, at least, try to solve this limited-angle problem, to avoid that the specimen receives more damage and we were able to observe/study the model correctly without any deformation or awkward effect.

The first objective of this work is to explore how the state of the art in this strong Image Processing branch is and second to decide which algorithm is the most suitable to carry this task out. We will take some important characteristics in consideration as the accuracy of the algorithm, speed, complexity and some other characteristics.

The main goal of this project, once we have found and we have decided which algorithm we are going to use, is to reproduce, study and analyze the algorithm. Then, we will apply the algorithm to some test patrons, like Shepp-Logan phantom and Hans Andrey crystallography.

Finally the idea is to apply the algorithm to real data and we expect to succeed in the reconstruction. If we succeed, next time we face this type of troubles, we

[4]Computed Tomography
[5]Transmission Electron Microscope

will be able to solve them by applying the technique explained and developed in this report.

Several efforts have been made to solve this problem with algorithms like POCS, FBP, Bayesian methods, Fourier techniques, algebraic reconstruction techniques (ART), etc. We are going to try to succeed with a novel approach. We have to remind that it is an experimental study. Indeed, no previous studies in this kind of data have been published, so we do not know a priory the results we can get or even if we will get some results.

1.2 Limitations

One of the biggest limitations of this work is that we do not know if the results are going to be the desirable ones. It is not possible to predict what will happen. It is possible that we find an algorithm that fits with our problem, we can reproduce and test the algorithm but that finally when we apply it to real data, we do not obtain good results. On the other hand, it is possible that we obtain an exact reconstruction and that we can solve the problem.

There are a lot of different data that have this kind of problem. It is not easy to find a unique solution for all type of data. We have to understand that it is not always possible to satisfy everybody and we must focus our efforts on the data that we possess.

When some theories, models or applications are recent and new, there is always a big discussion and everybody has its own opinion and idea about it. What we want to express is that we have to be opened to read a large quantity of different opinions and that maybe we will find some contradictions between them. It is going to be a hard job to difference between what is useful and what is not; who is right and who is not. We have to be careful and meticulous when we choose our sources.

Another limitation is quite typical of image processing, it is the processing time of reconstructions/simulations, because the work is going to be done in a common-home laptop. We do not dispose of a supercomputer, so we will spend long time waiting for the reconstructions/simulations instead of being working.

1.3 Why this approach is needed?

In many applications it is not possible to collect projection data over a complete angular range of 180º. Examples include electron microscopy, astronomy, geophysical exploration, nondestructive evaluation, and many others. As a brief explanation[6] of why this approach is needed, we can say that problems

[6]Large explanation could be read in section 2.4

of limited-angle view are very common in all kinds of scenarios, as we listed above.

In our specific case, we can say that in electron microscope the image recovery problem refers to the reconstruction of a three dimensional (3D) image distribution of a specimen from its two-dimensional projections measured at a set of angular views. Data at different view angles are collected by tilting the specimen using a goniometer. It is common to obtain limited-angle view in the final reconstruction, due to physic limitations. It is a daily problem that a large group of scientists face every day around the world. It would be a great advance if it is possible to solve or at least contribute to the advancement in finding a solution to this unfortunate effect and we will be helping to make life easier for many researchers.

1.4 Objectives

At the beginning of the project we decided to reach the following potential objectives:

1. Check the literature to establish existing methods which may help to carry out our task.
2. Develop the code of the chosen method.
3. Apply the method in two dimensional cases.
4. Run tests to optimize the method in two dimensional cases.
5. Expand to three dimensional cases.
6. Run tests in three dimensional cases.

The above list was the one on which we based our steps when developing the project. The idea was to complete all objectives and at the end to be able to discuss the results of the applied tests on two and three dimensional cases.

1.5 Overview of the thesis

The first pages of this master thesis are an introduction to relevant theories concerning Compressed Sensing and image reconstruction techniques, focusing on one specific. After that, the theory behind the chosen algorithm is explained in more details, as well as its implementation with MatLab code. Evaluation and analysis of the algorithm results are then presented. Following that a discussion about the results is held. This report ends with the conclusions of the algorithm results. We also propose possible future works related to this project.

2 Background

This part contains only a brief explanation of the fundamental parts of the actual techniques and theories in image reconstruction in order to be able to understand the rest of the work. For further details concerning the theoretical and mathematical background of the following sections of this chapter, feel free to check the next references [1, 2, 3, 8].

We also want to add few definitions that they will help you to understand even better this report:

Definition 1: The meaning for “sparse” when we are referring to an image, consists in an image with few non-zero values (usually with great values) and the rest is filled by values equal to zero or near to zero.

Definition 2: we will understand the verb “sparsify” as the action to get an sparse image, matrix, signal, etc. from a non-sparse image, matrix, signal, etc.

2.1 Compressed Sensing[7]

Since 2004 signal processing has exploded an existing branch called “Compressed Sensing”, “CS” or “Compressive Sampling”, etc. This technique breaks with the sampling rule of Nyquist-Shannon Theorem[8]. The main idea behind CS is to exploit that there is some structures and redundancies in the majority of interesting signals (they are not pure noise). In particular, most signals are sparse as they contain many coefficients close or equal to zero, when they are represented in some domain. So we want to exploit this characteristic.

As we can read in [1], *“our modern technology-driven civilization acquires and exploits ever-increasing amounts of data, ‘everyone’ now knows that most of the data we acquire ‘can be thrown away’ with almost no perceptual loss – witness the broad success of loss compression formats for sounds, images and specialized technical data. The phenomenon of ubiquitous compressibility raises very natural questions: why go to so much effort to acquire all the data when most of what we get will be thrown away? Can't we just directly measure the part that won't end up being thrown away?”* That is the strongest point of the CS theory, a large majority of the data is redundant so if we have an image where a huge amount of data is missing. Maybe this small quantity of data is enough to achieve an accurate or exact reconstruction of the original image.

According to the standard image reconstruction theory in medical/biological imaging, in order to avoid view aliasing artifacts, the sampling rate of the

[7]Also known as CS, compressive sensing, compressive sampling and sparse sampling

[8]The sample frequency has to be at least twice the highest frequency of the signal to obtain a perfect reconstruction

view angles must satisfy the Nyquist-Shannon sampling theorem. The universal applicability of the Nyquist-Shannon sampling theorem lies in the fact that no specific prior information about the image is assumed. However, in practice, some prior information about the image is typically available. When the available prior information is appropriately incorporated into the image reconstruction procedure, an image may be accurately reconstructed even if the Nyquist-Shannon sampling requirement is significantly violated. If a target image is known to be a set of sparsely distributed points, one can imagine that the image may be reconstructed without satisfying the Nyquist-Shannon sampling theorem. Of course, it is not an easy task to formulate a rigorous image reconstruction theory to exploit the sparsity hidden in the signals that we want to reconstruct. Fortunately, a new image reconstruction theory as we mentioned before, CS, was rigorously formulated to systematically and accurately reconstruct a sparse image from an undersampled data set. It has been mathematically proved that an $N \times N$ image can be accurately reconstructed using on the order of $S{\cdot}ln(N)$ samples provided that there are only S significant pixels in the image.

Although the mathematical framework of CS is elegant, the relevance in medical/biological imaging critically relies on the answers to the question:

Are medical/biological images sparse?

If we find that a medical/biological image is not sparse, is it possible to use some transform to make the image sparse? A real medical/biological image is frequently not sparse in the original domain, normally it is the pixel representation. Thus what we want to say is that it is not really common to have a sparse image in pixel domain without any modification previously done. As we know medical/biological imaging physicists and clinicians have proved for a long time that a subtraction operation can make the resultant image much sparser. In the recently proposed CS image/signal reconstruction theory, mathematical transforms have been applied to a single image to make it sparser. We will refer to these transforms as sparsifying transforms, because their task is exclusively sparsify the image. A clear example could be that we can sparsify an image by applying a simple discrete gradient operation, FFT, wavelet transforms, etc. It is demonstrated that a medical/biological image can be made sparser even if the original image is not really sparse.

Thus instead of directly reconstructing our own image, we will work with the sparsified version of the image trying to reconstruct it[9]. Significantly fewer image pixels have significant image values in the sparsified image, so we will have

[9] In this case, we will see in section 3 and 4 that we have the sparsified image in Fourier Domain and we also have another sparsified version after we apply the Block Matching block to the image, because the filter uses the shared information between the image fragments to obtain a sparsified version of the block.

an image with a considerable number of zero values. After that it is possible to reconstruct the sparsified image from an undersampled data set without streaking artifacts or weird artifacts. After the sparsified image is reconstructed, an “inverse” sparsifying transform is used to transform the sparsified image back to the original domain of the first image. In practice, there is no need to have an explicit form for the inverse sparsifying transform. Indeed, only the sparsifying transform is needed in image reconstruction.

Finally, we want to add that techniques typically used in image reconstruction based on the theory of CS rely on convex optimization with a penalty expressed by the l_0 or l_1 norm which is exploited to enable the assumed sparsity. It results in parametric modeling of the solution and in problems that are then solved by mathematical programming algorithms. However there is another way of approaching the reconstruction problems. We can replace the parametric modeling with a nonparametric one implemented by the use of spatially adaptive denoising filtering, like in [4].

2.2 Image Reconstruction Techniques

As we commented in the introduction, when the range of tilt angles, for which projected images of 2-dimensionally specimens can be obtained in TEM, is limited by both technical aspects or by, more fundamental limitation, the thickness of the structure. The lack of a full set of projections could cause a missing cone or wedge in the data of the object, which will give an anisotropic resolution in a three-dimensional reconstruction and may cause some spurious artifacts that will deteriorate the quality of the data. This is a common problem when scientists acquire data from a CT, electron microscope, some other X-ray techniques, etc.

In such limited-angle examples, applying a standard full-data reconstruction algorithm, such as filtered back-projection (FBP), results in poor reconstructions with severe artifacts. Because of the importance of the limited-angle problem, many specialized algorithms have been introduced over the past twenty five years. Approaches have included maximum entropy techniques, Bayesian methods, projection onto convex sets (POCS), Fourier techniques, TV regularization, PICCS, algebraic reconstruction techniques (ART) and many others.

There are a large group of different existing techniques, as seen in the paragraph above, to solve limited-angle problems. In front of the impossibility to comment all methods, we will focus on the following ones:

1. **POCS**[10]**:**

   As you can read in [12, 13], POCS, also called the method of convex projections, is an iterative algorithm that uses the available data and certain types of prior knowledge to recover a signal. It finds a feasible solution consistent with a number of a priory constraints which are defined on the basis of the measured data, a priory information about the degradation operator, the noise statistics and the actual image distribution itself. For each constraint, a closed convex constraint set is defined such that the members of the set satisfy, the given constraint and the ideal solution which is a member of the set. POCS has been applied to extrapolation and interpolation, to computer tomography, to reconstruction from limited views, to electron microscope reconstructions and to computer tomography reconstructions with arbitrary geometries. One of the problems is that this method is the performance in the presence of noise, because it decreases rapidly. Other applications of POCS could be into image coding and to neural nets. We can say that POCS is one of the most used techniques for this kind of problems, limited-angle data (missing cone, wedge, etc).

   POCS results in a phantom case where projection data is limited to view range of (45º, - 45º), like in Figure 2.1[11] are really poor, around 40% of percentage error. The algorithm is not able to fill the whole empty space. The method performance increases when the view range is enlarged.

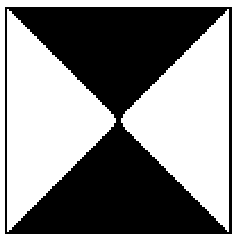

Figure 2.1: Limited-angle data

[10]Projection Onto Convex Sets

[11]Black zone - missing data. White zone - known data

2. **TV[12] minimization:**

   It is a method based on the idea to minimize the following expression

   $$min(E(x,y) + \lambda{\cdot}V(y))$$

   where $E(x,y)$ is the MSE, $V(y)$ is the Total Variation of signal y and λ is the regularization parameter.

   $$E(x-y) = \frac{1}{2}\sum_n |x_n - y_n|^2$$

   $$V(y) = \sum_n |y_{n+1} - y_n|$$

   The signal $x(n)$ is the original signal, then this technique consists of finding an approximation, $y(n)$, that satisfies the first expression. This method works properly for image denoising, interpolation problems and for recovering medical-type images from partial Fourier ensembles. More references of this method and a demo can be found in [7, 16].

3. **PICCS[13]:**

   It is an image reconstruction algorithm and it is implemented by solving

   $$min_x \left[\alpha \left\|\Psi_1(X - X_P)\right\|_{l_1} + (1-\alpha)\left\|\Psi_2 X\right\|_{l_1}\right],\ s.t.\ AX = Y$$

   Ψ_1, Ψ_2 are sparsifying transforms[14].
   α is the control parameter.
   X is the image and X_P is the prior image.
   Y is the line integral values.
   A is the system matrix to describe the x-ray projection measurements.

   This technique is able to reconstruct accurately signals/images from highly undersampled projection data sets.

[12]Total Variation
[13]Prior Image Constrained Compressed Sensing [6]
[14]i.e. discrete gradient

4. **Image Reconstruction Via Recursive Spatially Adaptive Filtering:**

   It is a variation of the Robbins-Monro stochastic approximation procedure [4] with regularization enabled by a spatially adaptive filter. It is useful to recover data from radon sparse projections, low frequencies or with a missing wedge. The idea is to solve the following iterative algorithm:

$$\hat{y}_2 = \begin{cases} \hat{y}_2^{(0)} = 0, & k = 0 \\ \hat{y}_2^{(k)} = \hat{y}_2^{(k-1)} - \gamma \left[\hat{y}_2^{(k-1)} - (1-S).*\Upsilon(\Phi(\Upsilon^{-1}(y_1 + \hat{y}_2^{(k-1)}))) + (1-S).*\eta_k \right], & k \geq 1 \end{cases}$$

   Definition 3: .∗ is a MatLab operator used as a point-wise multiplication between matrices.

   This algorithm is fully explained in subsection 4.1.

From the list above we decided to choose the last option, **Image Reconstruction Via Recursive Spatially Adaptive Filtering.** The main reason of this decision is that one of the cases which is solved by this algorithm is quite similar to a missing wedge. After that if we extrapolate to a 3-dimensional case we could have a pyramid or a cone and it fits exactly with our problem. The performance of this algorithm for radon sparse projections is not as good as the other alternatives, but we do not think that we have to worry about that.

The first option which was POCS was rejected due to the poor results obtained in cases where the view range is (45^{o}, - 45^{o}), because with the ranges of view it is similar to a cone in 3D or a wedge, then we can say that POCS is not running properly for our case. Moreover POCS performance in noisy scenarios is not good, and it is possible to have noisy models from a TEM.

The second and third options, were rejected due to the fact that we had no proof that they could run in cone or wedge case. They are used to solve limited-angle problems but these cases are more similar to radon sparse projections than a missing cone case. Moreover the mathematical solution of their equations is very complex and there is a great amount of literature based on finding methods to solve these equations.

2.3 The Data

What kind of data do we have?

We are going to use two different types of data, one for 2D case and one other for 3D:

- **Shepp-Logan Phantom (2D):**

  The Shepp-Logan phantom is an image test patron which was created as a standard for computerized tomography (CT) image reconstruction simulations. It is very frequently used in image reconstruction literature to evaluate algorithms performance. In MatLab Shepp-Logan phantom with size 256×256 can be easily created as follows:

```
%How to generate a Shepp-Logan Phantom and visualize it
Ph = phantom('Modified Shepp-Logan',256);
figure(1), imagesc(Ph);
```

The visualization of Shepp-Logan phantom in pixel domain is presented in the following figure:

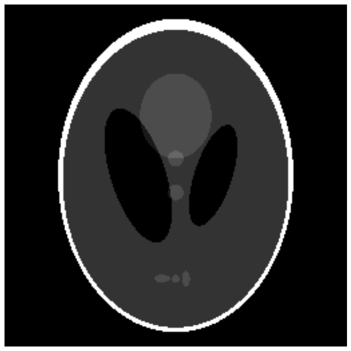

Figure 2.2: Shepp-Logan phantom

On the other hand, its spectrum looks like as follows:

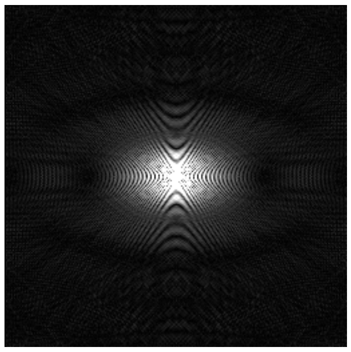

Figure 2.3: FFT Shepp-Logan phantom

- **Artificial Protein Crystallography Data (3D):**

  The data in this case reproduces a traditional crystallography but the results were not obtained from an Electron Microscope, they were reproduced artificially in a computer. This model is known as Hansandrey[15]

[15]In Appendix D is present a group of slices

crystallography and it consists of a $100x100x100$ cube of data that contains a protein model. Snapshot of the model is available in subsection 2.3.2.

- **Real Protein Crystallography Data[16] (3D):**

  The data was collected by method X-ray protein crystallography which we can be defined as a technique of determining the arrangement of protein atoms within a crystal, in which a beam of X-rays strikes the crystal and diffracts into many specific directions. From the angles and intensities of these diffracted beams, in our case, a TEM can produce a three-dimensional picture of the density of electrons within the crystal. From this electron density, the mean positions of the atoms in the crystal can be determined. Snapshot of the model is available in subsection 2.3.2.

  We are also going to give a brief explanation about what is a TEM:

  It is defined as a microscopy technique whereby a beam of electrons is transmitted through an ultra thin specimen, in this case a protein crystallography, interacting with it. An image is formed from the interaction of the electrons transmitted through the specimen. After that the image is magnified and focused onto an imaging device, some examples of imaging devices could be a fluorescent screen, a layer of photographic film, or there is another way to be detected by a sensor such as a CCD camera.

[16] In Appendix D, you can find some slices of the protein model

A typical scheme of a microscopy is presented below (2.4):

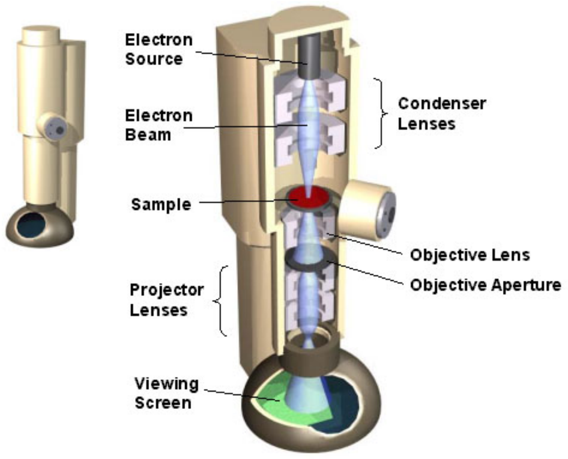

Figure 2.4: Transmission Electron Microscopy

2.3.1 Shepp-Logan Phantom

As we explained in the section above, the Shepp-Logan phantom is used as a test patron in image reconstruction and we are going to use it as well. For the first stage of the algorithm analysis we will use three different variations of Shepp-Logan phantom:

1. 22 Radon sparse projections[17] (a)
2. 11 Radon sparse projections (b)
3. 90º degrees missing data (c)

It is presented in (2.5) the three modifications, in pixel domain, of
Shepp-Logan phantom that we are going to use:

[17]The Fourier Transform of the Radon transform with respect to the projection coordinate equals a radial FFT line (of the unknown function f), according to the Fourier Slice theorem.

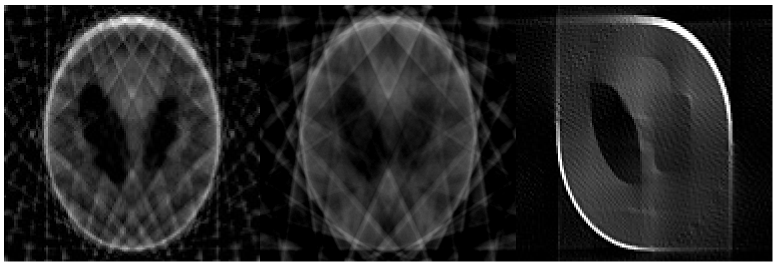

Figure 2.5: Reconstructions of a.22lines b.11lines c.90º missing data

It is presented in (2.6) the three modifications, in frequency domain, of the Shepp-Logan phantom that we are going to use:

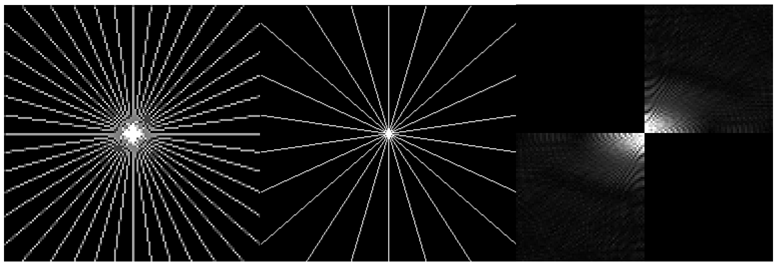

Figure 2.6: a.fft_22projections b.fft_11projections c.fft_90º_missing

The reconstruction of these three modifications of Shepp-Logan phantom helped us to evaluate the robustness and effectiveness of the algorithm.

2.3.2 Protein Crystallography Data

The electron crystallography is the study of 2D crystals by electron microscopy, in this case a Transmission Electron Microscopy. Such crystals typically consist in only one thick molecule thick but many molecules across. Hence, they are fragile, deformable and need to be supported by a flat electron-translucent but it should be a sufficient stable surface, which is typically a carbon support film or similar in conjunction with an electron microscopy grid. Our first model is a crystallography data obtain in one of Karonlinska Institutet TEMs and the second one, test model, is an artificial crystallography.

▷ The first one[18] contains the model which we want to reconstruct. There is a missing/corrupted cone of data, in frequency domain. The model has a cube structure and size of $80 \times 80 \times 80$. In the next figure is shown the protein model. Read and visualized by Chimera:

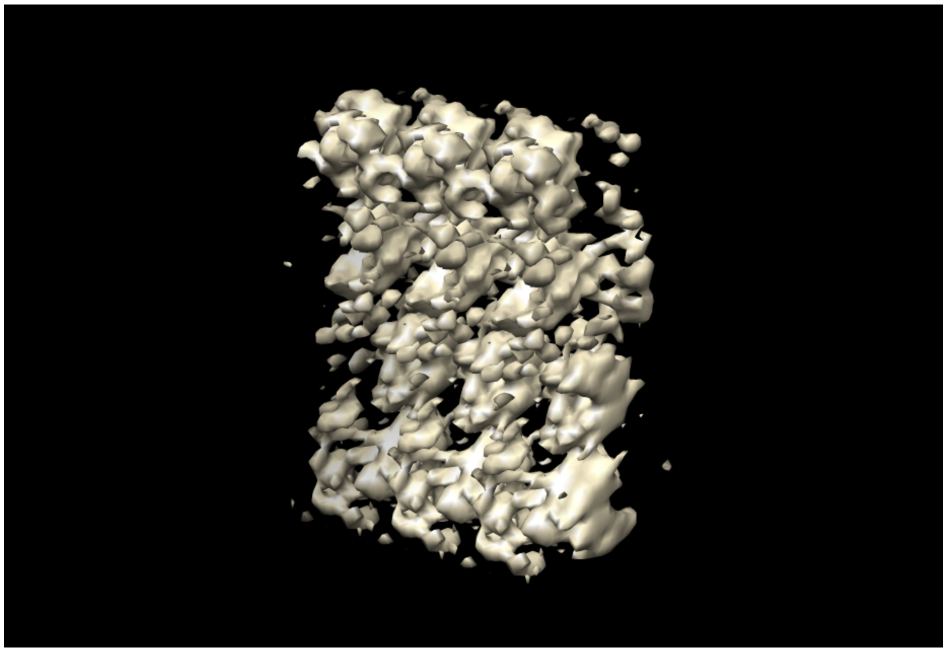

Figure 2.7: First protein model, protein.mrc

[18]Name file: "protein.mrc"

▷ The second file[19] consists in a patron model test to analyze the reconstruction results, this model has the same cube shape as the other model but its size is $100 \times 100 \times 100$.

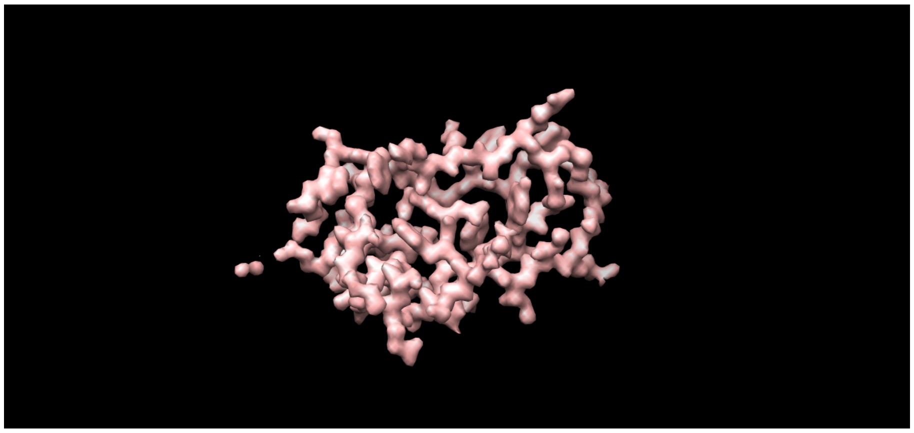

Figure 2.8: Hansandrey model, hansandrey.raw

[19]Name file: "hansandrey.raw"

▷ In Tables 2.1 and 2.2 are shown the statistics of both 3D models "protein.mrc"[20] and "hansadrey.raw"[21]:

Statistics of 3D-data "protein.mrc"			
Object dimension	w=80	d=80	h=80
Maximum	0.0175607		
Minimum	-0.0184695		
Mean	-6.16801e-13		
Variance	2.72724e-06		
Std Dev	0.00165144		
RMS	0.00165143		
Skewness	0.0862649		
Kurtosis	30.7101		
First minima at location	w=19	h=26	d=40
First maxima at location	w=29	h=34	d=35
Total Integral	-3.15802e-07		
Positive Integral	110.089		
Negative Integral	-110.089		
Total Contributing Points	512000		
Contributing Positive Points	34950		
Contributing Negative Points	35610		
Contributing Zero-Valued Pts	441440		

Table 2.1: Statistic of 3D-data "protein.mrc"

[20]It is the real data which we want to reconstruct
[21]It is the reference data to compare with the results that we obtain

Statistics of 3D-data “hansandrey.raw”			
Object dimension	w=100	d=100	h100
Maximum	51662.6		
Minimum	-123.313		
Mean	5203.15		
Variance	4.48678e+06		
Std Dev	2118.2		
RMS	5617.79		
Skewness	8.16291		
Kurtosis	75.1485		
First minima at location	w=24	h=52	d=72
First maxima at location	w=32	h=55	d=51
Total Integral	5.20315e+09		
Positive Integral	5.20315e+09		
Negative Integral	-123.313		

Table 2.2: Statistics of 3D-data “hansandrey.raw”

2.4 Statement of the Question or Problem

In the world of crystallography it is very common to obtain limited-angle effects in the final model because of physic limitations. To explain this obstacle in crystallography, we have to remember that a Transmission Electron Microscope produces projections of 3D objects. To arrive at a 3D structure, different projections must be combined, with 2D crystals. The different projections are generated by tilting the crystals relative to the incident electron beam. However, in practice, virtually no data can be obtained with the specimen tilted beyond 70^{o} because the bars of the support grid begin to occlude the specimen beyond this angle, and in most cases tilt angles up to only 55^{o}–60^{o} are recorded. There are therefore information deficits along one direction in space (the z-axis, normal to the crystal x,y plane), which can lead to distortions, loss of resolution and the artifactual introduction or omission of densities. This is called as we mentioned as “missing cone” or, less commonly, as “dead zone” and it can be understood if one thinks about the unsampled wedge of information (i.e. a 30^{o} wedge when tilting is restricted to $\pm 60^{o}$) that is rotated by 180^{o} to simulate the unrestricted rotation of a specimen in the plane – thereby creating a cone.

How serious the problem of missing cone is also depends on the individual structural features of a protein. The impact of the missing cone is particularly severe when it excludes data that relates to the most prominent feature of a structure. The other important disturbance is the introduction of artifactual densities. However, although this needs to be of particular concern, it seems to be only relevant for structures in which the densities are diffuse and insufficiently separated owing to resolution. To this end, high-resolution structural data (to <4 A with a missing cone not larger than 30^{o}) with well-separated densities seem to be more robust than are lower-resolution data. However, to arrive at a robust high-resolution data-set in which all features are resolved, the missing cone impact has to be minimized. Thus we can conclude that the impact of a missing cone depends in big measure on the kind of data that we have.

We can assure that it is a big problem with the above arguments and that this missing cone, wedge or other limited-angle effect remains without a proper answer of how to recover this missing/corrupted data. It would be a great contribution to world science, not only for crystallography. It is one of the active research topics in image reconstruction field and some techniques are commonly used like POCS[22], Bayesian methods, Fourier techniques, TV regularization, PICCS, ART, etc.

Finally, we think it is worth to spend our time and resources on trying to find a solution to this common problem that physicists around the world have to face every day when they are trying to obtain the 3D model of a protein. Thus we sincerely see that this trouble has to be solved or at least we would like to contribute to its future solution.

[22]Project Onto Convex Sets

3 Method

3.1 Compressed Sensing Image Reconstruction Via Recursive Spatially Adaptive Filtering

The first goal of this project was to check the literature, find and reproduce a reconstruction algorithm in Matlab code. In order to accomplish this task, an exhaustive study of the actual literature[23] related to the topic was clearly needed. Our choice was a variation of the Robbins-Monro stochastic approximation procedure with regularization enabled by a spatially adaptive filter [4].

In the publications [1, 2, 3], it is shown that under CS assumptions, stable reconstruction of unknown signal is possible and that in some cases the reconstruction can be exact. These techniques typically rely on convex optimization with a penalty expressed by the l_0 or l_1 norm which is exploited to enable the assumed sparsity [8]. It results in parametric modeling of the solution and in problems that are then solved by mathematical programming algorithms. In this report it is proposed to replace the traditional parametric modeling used in CS by a nonparametric[24] one.

The nonparametric modeling is implemented by the use of spatially adaptive filters. The regularization imposed by the l_0 or l_1 norms is essentially only a tool for design of some nonlinear filtering. This implicit regularization is replaced by explicit filtering, exploiting spatially adaptive filters sensitive to image features and details. If these filters are properly designed we have reasonable hopes to achieve better results than it can be achieved by the formal approach based of formulation of imaging, as the variational problem with imposed global constraints. In imaging, the regularization with global sparsity penalties (such as l_p norms in some domain) often results in inefficient filtering. It is known that a higher quality can be achieved when regularization criterias are local and adaptive.

How does the algorithm work?

This method of signal reconstruction is realized by a recursive algorithm based on spatially adaptive image denoising. Each iteration provides the block matching spatially adaptive filtering algorithm with data by the injection of random noise in the unobserved portion of the spectrum. The denoising filter working in the image domain, attenuates the noise and reveals new features and details out of the incomplete and degraded observations. Roughly speaking, we seek for the solution (reconstructed signal) by stochastic approximations whose search direction is driven by the denoising filter. This method is applicable for four important inverse problems which are not all from electron microscope data:

[23] All literature references can be found in Bibliography

[24] The model structure is not specified a priory but is instead determined from data. The number and nature of the parameters are flexible and not fixed in advance.

1. Missing cone effect.
2. Missing wedge effect.
3. Recovery from sparse projections.
4. Recovery from low frequencies.

It is fully explained in the Analysis section the reconstruction characteristics of the cases above.

3.2 Filtering

Filtering is one of the central blocks. It is a very important part of the algorithm, because it is responsible for changing the estimated image, to fill the unobserved/missing data. To carry this task out we have used the same filter as in [5] with some modifications.

The idea relies on the fact that if the filter is well-designed, we have great chances to achieve better results by filtering than through the common approach based on a variational problem with imposed global constraints. This filtering consists in a block-matching and 3-dimensional denoising filtering, in order to obtain a highly sparse representation of the data, recall that is one of main conditions of Compressed Sensing theory.

In reference [5], they explained in the abstract that *"We propose a novel image denoising strategy based on an enhanced sparse representation in transform domain. The enhancement of the sparsity is achieved by grouping similar 2D image fragments (e.g. blocks) into 3D data arrays which we call "groups". Collaborative filtering is a special procedure developed to deal with these 3D groups. We realize it using the three successive steps: 3D transformation of a group, shrinkage of the transform spectrum, and inverse 3D transformation. The result is a 3D estimate that consists of the jointly filtered grouped image blocks. By attenuating the noise, the collaborative filtering reveals even the finest details shared by grouped blocks and at the same time it preserves the essential unique features of each individual block. The filtered blocks are then returned to their original positions. Because these blocks are overlapping, for each pixel we obtain many different estimates which need to be combined. Aggregation is a particular averaging procedure which is exploited to take advantage of this redundancy".*

They are taking advantage of shared information between blocks and attenuating noise, they reveal new details and features of the original image. To understand better this procedure we can look at the scheme:

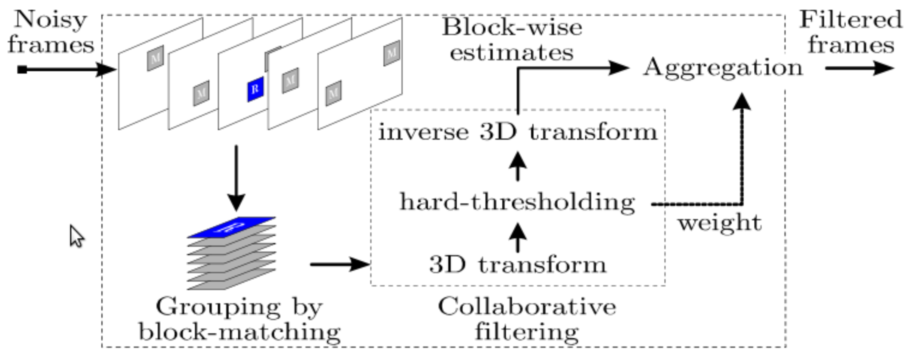

Figure 3.1: Filtering scheme

We have used the “Filtered Frames” as our reconstructions, we eliminated a second block that the original *BM3D* [5] filter contains. This block consists in a Wiener filter (that it is not present in the scheme), the reason why we removed this block is due to the large amount of processing time that it needs to deliver a new estimator and the improvement on the estimator is not very important (around $0.5dB$).

As you can see, we introduce a “Noisy image”, in our case the image that we want to reconstruct, and the first block (Block Matching) starts to join similar fragments of the image into “groups” to exploit their shared information. To select if a fragment is valid or not for a block is used the $l_2 norm$ to calculate the distance between the reference and the possible candidates. This task to join image fragments into a block is done in *“Grouping by block-matching”*.

After that when we have the “groups”, it is applied the *“Collaborative filtering”* which consists to obtain the 3D transform[25] of the group, immediately after a hard-thresholding is applied to the group, in the transformed domain, to attenuate or enhance the noise which is the responsible to create the missing coefficients in the unobserved part of the image transform. Finally we anti-transform the group and we apply the block *“Aggregation”* that computes the basic estimate of the true-image by weighted averaging all of the obtained block-wise estimates that are overlapping.

The implementation of the filter (BM3D_variant.m) is available in Appendix A, the main differences with the original code are commented below:

```
%%%% Select transforms ('dct', 'dst', 'hadamard', or anything that is listed by 'help wfilters'):
transform_2D_HT_name     = 'haar' or 'bior1.5'; %% WE WERE USING BOTH TRANSFORMS
                                                   DEPENDING ON THE CASE %%
```

[25]Applying a 2D transform first and after that a 1D transform for the third dimension

```
transform_2D_Wiener_name = 'haar'; %%WE DO NOT USE THIS PARAMETER, BECAUSE WE
                                     ELIMINATED THE WIENER FILTERING BLOCK %%
transform_3rd_dim_name   = 'haar'; %%WE HAVE USED Haar WAVELET AS THE 3-rd DIMENSION TRANSFROM,
                                     BECAUSE IS THE FASTEST %%

%%%% Hard-thresholding (HT) parameters:
N1                  = 8;   %% N1 x N1 is the block size used for the hard-thresholding (HT) filtering,
                           WE USED 4 INSTEAD OF 8 FOR THE 3D MODELS, BECAUSE ITS SIZE WERE SMALLER
                           THAN THE PHANTOMS %%
Nstep               = 3;   %% sliding step to process every next reference block
                           %% WE KEPT THE SAME VALUE FOR ALL CASES
N2                  = 16;  %% maximum number of similar blocks (maximum size of the 3rd dimension of a 3D array)
                           %% IN 3D MODELS WE REDUCED THIS VALUE TO 12, TO INCREASE THE SPEED OF THE ALGORITHM
Ns                  = 39;  %% length of the side of the search neighborhood for full-search block-matching (BM), must be odd
                           %% IN 3D MODELS WE REDUCED THIS VALUE TO 31, TO INCREASE THE SPEED OF THE ALGORITHM
tau_match           = 3000;%% threshold for the block-distance (d-distance)
                            %% IN 3D MODELS WE REDUCED THIS VALUE TO 1800, TO HAVE MORE SIMILAR NEIGHBOURS IN THE BLOCK

%% WE NOT USE THE SECOND STEP OF THE ALGORITHM: WIENER FILTERING %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Step 2. Produce the final estimate by Wiener filtering (using the
%%%%  hard-thresholding initial estimate)
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%WE DO NOT USE THE WIENER FILTERING PART%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%%%%%
%tic;
%y_est = bm3d_wiener(z, y_hat, hadper_trans_single_den, Nstep_wiener, N1_wiener, N2_wiener, ...
%    'unused arg', tau_match_wiener*N1_wiener*N1_wiener/(255*255), (Ns_wiener-1)/2, (sigma/255),...
%    'unused arg', single(TforW), single(TinvW)', inverse_hadper_trans_single_den, Wwin2D_wiener,...
%    smallLNW, stepFSW, single(ones(N1_wiener)) );
%wiener_elapsed_time = toc;

y_est = y_hat; %% WE USE y_hat FROM THE FIRST STEP, hard-thresholding (HT), as our estimator y_est

end
return;
```

The final estimator y_est is obtained from the first step, hard-thresholding (HT). In that step the function `bm3d_thr()` is the responsible to apply the threshold, but we were not able to access to the source code. The reason of that is because the function `bm3d_thr()` is contained in a *black box*[26], actually is contained in the library *"bm3d_thr.dll"* which is not possible to read. It is a little bit disappointing not to have the possibility to read the most exciting part of the code and not to have the chance to change some parts of this function to try new scenarios for our reconstructions.

[26]The source code is not available

3.3 How to apply in 3D models

One of the main goals of this thesis is to apply the chosen algorithm to 3D models of data. In this case, the data comes from an artificially made crystallography and from a protein crystallography[27] taken from a TEM.

The first model of data, as we explained before, consists of a raw data cube[28] with a dimension of $100 \times 100 \times 100$ pixels. The second one, consists on a MRC file with $80 \times 80 \times 80$ pixels.

Before applying the algorithm was necessary to develop a tool which was able to read raw data, because we had to read structures with extension *".mrc" and ".raw"*. This kind of files are unreadable by Matlab, so we developed the MatLab function ``readraw()'', which is in charge of this task and gives us a useful output structure. From this point, we were able to use this data and modify, reconstruct, test, etc. in Matlab.

Then, we decided that the best way to work with the cube was to obtain 2D slices from the 3D model, and apply the algorithm separately to each one, exploiting the properties of Fourier transform which says that the Fourier transform of a cube is the same than the Fourier transform of the slices separately, they are independent between them. The function ``slices()'' was in charge of this task.

After the slices were obtained we were already able to apply the algorithm, implemented in function ``recons()''. Thus one of the latest steps was to apply the optimized algorithm to every slice of the protein model, save the results and visualize the 3D model.

The last point which was visualization, was one of the most complicated, because 3D representation of protein structures in Matlab is not a very easy task and its results are not the best ones. What we did was to check which program was the most used in protein representation. Our research ended with the program **Chimera**, but our work continued, because a tool was needed to convert a MatLab structure into a Chimera file. To carry this task out we used the function ``WriteMRC()'' which converts a MatLab structure into a .mrc file, readable by Chimera.

[27] Slices of the data are available in Appendix D

[28] Characteristics of the data are explained in Appendix D: 3D-data

4 Implementation

4.1 Algorithm

As we previously said, the algorithm used in this project consists in a variation of the Robbins-Monro stochastic approximation procedure [4] with regularization enabled by a spatially adaptive filter (Φ).

We are going to proceed to explain how this iterative system it works:

$$\hat{y}_2 = \begin{cases} \hat{y}_2^{(0)} = 0, & k = 0, \\ \hat{y}_2^{(k)} = \hat{y}_2^{(k-1)} - \gamma \left[\hat{y}_2^{(k-1)} - (1-S).*\Upsilon(\Phi(\Upsilon^{-1}(y_1 + \hat{y}_2^{(k-1)}))) + (1-S).*\eta_k \right], & k \geq 1, \end{cases} \tag{4.1}$$

DEFINITION 4: $.*$ is a MatLab operator used as a point-wise multiplication between matrices.

Parameters:

$\hat{y}_2 \equiv estimation\,of\,unknown\,data\,in\,Fourier\,domain$
$y_1 \equiv known\,data\,in\,Fourier\,domain$
$\gamma \equiv speed\,step\,of\,the\,algorithm$
$(1-S) \equiv mask\,to\,select\,the\,region\,of\,the\,unknown\,data$
$\Upsilon \equiv Fast\,Fourier\,Transform,\ R^n,\ C^n,\ n=2$
$\Phi \equiv filtering\,block$
$\Upsilon^{-1} \equiv Inverse\,Fast\,Fourier\,Transform,\ R^n,\ C^n,\ n=2$
$\eta_k \equiv Gaussian\,noise$

We have to explain that we divide the images as:

$$y = S.*y + (1-S).*y = y_1 + y_2$$

Where S is a mask filled up with 0_s and 1_s to select the desired region of the image. In the following figure (4.1) you can see what we understand as known data, y_1, and unknown data, y_2, for the 90º missing data case:

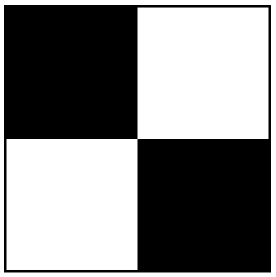

Figure 4.1: Known data (white) and unknown data (black)

Parts of the algorithm:

$$\hat{y}_2^{(k-1)} \Rightarrow estimated\, unknown\, data\, in\, the\, iteration\; k-1 \tag{4.2}$$

$$(1-S).*\Upsilon(\Phi(\Upsilon^{-1}(y_1+\hat{y}_2^{(k-1)}))) \Rightarrow new\, estimation\, for\, iteration\; k \tag{4.3}$$

$$(1-S).*\eta_k \Rightarrow noise\, in\, the\, unknown\, data\, region\, and\, iteration\; k \tag{4.4}$$

$$\hat{y}_2^{(k-1)} - (1-S).*\Upsilon(\Phi(\Upsilon^{-1}(y_1+\hat{y}_2^{(k-1)}))) + (1-S).*\eta_k \tag{4.5}$$

We can observe in (4.2) that we have the estimated unknown info of the spectrum in the iteration k-1, then with the help of (4.3) and (4.4) we obtain the difference between the unknown data in the previous iteration and the new estimation (4.3) plus Gaussian noise around the unknown data region, to achieve (4.5). If we consider the whole block of (4.5) we can see it, as an update block, it is responsible for the new changes of $\hat{y}_2^{(k)}$. Then we make the comparison with the previous iteration $\hat{y}_2^{(k-1)}$ again, to finally obtain (4.1), $\hat{y}_2^{(k)}$.

During the tests to find the best performance of the algorithm I found pretty tough problems of choosing the variable parameters, exciting noise, algorithm-step, filtering block and the masks.

A reconstruction example is shown in the following figure, which consists in the reconstruction of the 90º degrees missing data (c) of size 128×128.

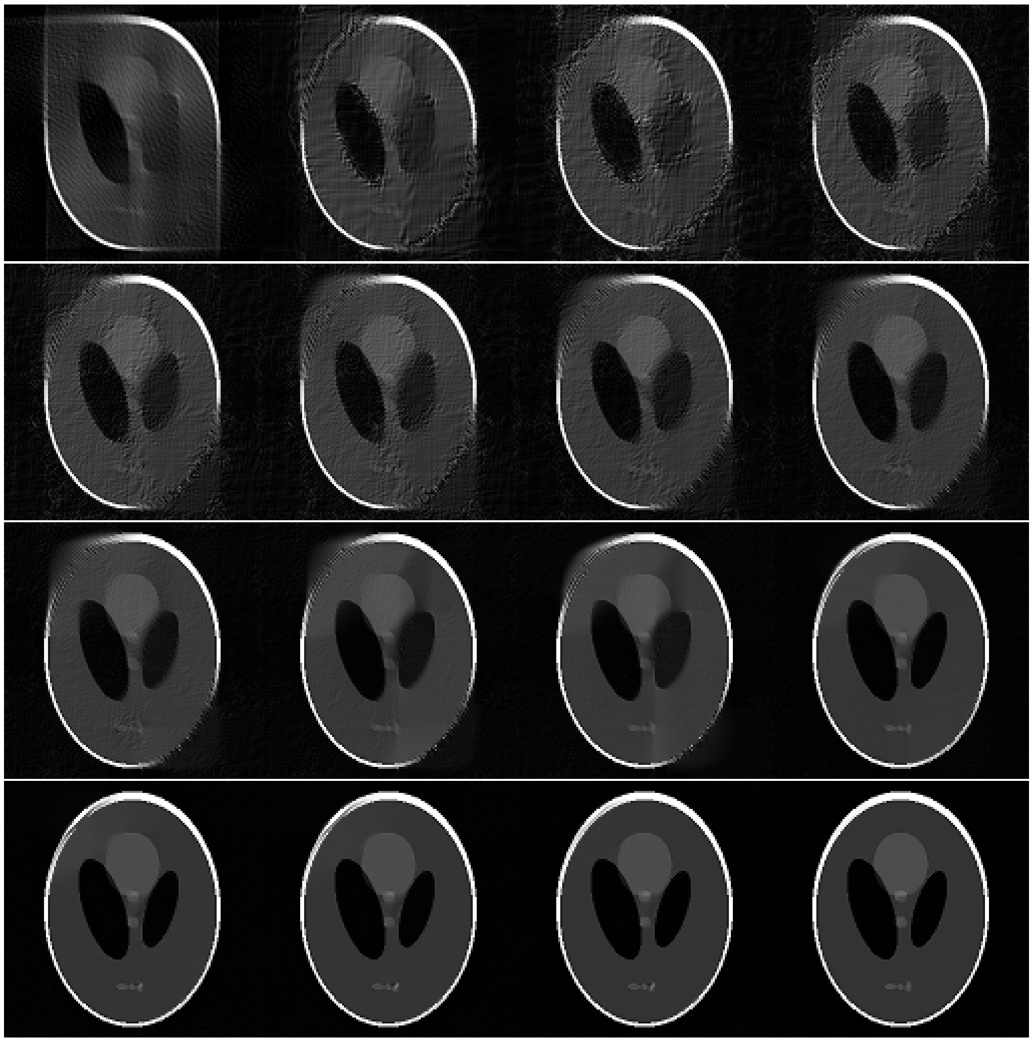

Figure 4.2: Reconstruction 90º degrees missing data

The above example consists in a reconstruction of case c). In this case the reconstruction has a great quality, but it is not perfectly exact. We obtained a PSNR of $46.08dB$ after 20000 iterations.

Note: *in the Appendix C there is another reconstruction example of case c with size* 256×256.

To achieve the results of last reconstruction, we had to run several tests to determinate the optimum value of the algorithm parameters[29]. We followed the next steps to decide the correct value of the parameters:

▷ The **first parameter** we run on test, was the type of wavelets to use in the filtering block (BM3D). We are referring to this part of the BM3D.m code:

```
transform_2D_HT_name     = 'haar';    %% transform used for the HT filt. of size N1 x N1
transform_2D_Wiener_name = 'dct';     %% transform used for the Wiener filt. of size N1_wiener x N1_wiener
transform_3rd_dim_name   = 'haar';    %% transform used in the 3-rd dim, the same for HT and Wiener filt.;
```

Our duty was to choose the best combination of *"transform_ 2D_ HT_ name"* and *"transform_ 2rd_ dim_ name"*, the second transform *"transform_ 2D_ Wiener_ name"* is not used in the code, because the part of the wiener filter requires a big amount of computation time and we decide to eliminate it.

We made the choice based on the following results:

Combination	Reconstruction Accuracy[30]	Computational Time
Haar-Haar	88.31%	Low
dct-Haar	91.83%	Low-Medium
Haar-dct	88.10%	Medium-High
bior1.1-Haar	87.70%	Medium
bior1.3-Haar	90.16%	Medium
bior1.5-Haar	90.16%	Low
bior2.2-Haar	79.62%	Medium
bior2.8-Haar	87.50%	Medium
bior3.1-Haar	100%	Medium
bior3.3-Haar	88.31%	Medium
bior3.5-Haar	87.30%	Medium
bior3.7-Haar	89.33%	Medium
bior5.5-Haar	88.72%	Medium
bior6.8-Haar	85.11%	Medium
db2-Haar	87.10%	Medium
db12-Haar	90.78%	Medium-High
Haar-bior1.5	88.72%	High
bior1.5-Haar*[31]	90.78%	Medium-High

Table 4.1: Wavelets combination

[29] Amplitude of exciting noise, algorithm step size, decaying rule, type of mask and type of wavelets

As we can see in Table 4.1, the best wavelets combination is bior3.1-Haar but it takes a medium computational time. The second best in reconstruction accuracy is dct-Haar but it takes a low-medium computational time. Then if we checked the third best option in reconstruction accuracy is bior1.5-Haar. In this case, the computational time is low. Thus if we take into consideration all factors, we decided that the most accurate one was this last option. The computational time that we save is possible to use to run the algorithm longer time. To have an idea about the computational time we attach a table with reference times:

Computational Time	Minutes
Low	~ 19
Low-Medium	~ 23
Medium	~ 28
Medium-High	~ 36
High	~ 70

Table 4.2: Computational Time

▷ The **second parameter** we decided to optimize, was the exponentially decreasing variance of the exciting noise $var\{\eta_k\}$. This parameter controls the level of smoothing in the recursive procedure and hence the rate evolution of the algorithm. To carry out this selection, we prepared some exponential decreasing variance and we evaluated their performances.

The mathematical expression of the standard deviation for this case is:

$$std_{32} = var\{\eta_k\}^{\frac{1}{2}} = \alpha^{\frac{1}{2}(-i-\beta)}$$

Where i is the actual iteration of the algorithm. Our modifications were focused on α and β values. After applying a series of tests, we obtained the best results with the following values:

[32]std - standard deviation

Case	α	β
22 Projections	1.008	750
11 Projections	1.008	750
90º missing data	1.0008	9000
Low frequency	1.008	1250
3D data[33]	1.0008	8500

Table 4.3: α, β values

In Appendix C are attached the graphics of every Standard Deviation case. If we observe the evolution of each std, we can observe that if there is no missing data all around the spectrum (90º missing data or 3D data), it is better if the decay of std is more relaxed. In contrast to other cases it is better if the decay is more abrupt and quickly reaches a value close to zero.

▷ The **third parameter** was the amplitude of the exciting noise.

After we decided the std in all our different cases we wanted to study the effect that produced the change in the amplitude of the noise in the reconstructions. Thus to carry this test out we varied the value of the amplitude of the noise to know which was the most appropriate value and we obtained these final results:

Case	Amplitude
22 Projections	150
11 Projections	150
90º missing data	15
Low frequency	125
3D data[34]	6

Table 4.4: Noise Amplitude

From now we have defined the noise, η_k, to be used in all cases.

▷ The **fourth parameter** was the step size of the algorithm, which controls the speed of the algorithm.

In the decision to choose the best magnitude for the step size of the algorithm, we had to take into account the restriction that the value of this parameter was between 1 and 2. After this clearance, our next steps were to test values between 1 and 2. The best performance was achieved with $\gamma = 1.5$.

▷ The **fifth parameter** was the mask to use in each case.

This parameter depends exclusively on the spectrum that we have, so it is going to be different in each case. For each case we decided these masks:

- Cases 22 projections, 11 projections and 90º missing data:

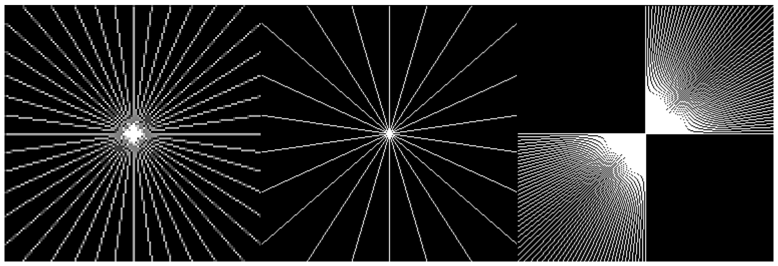

Figure 4.3: Masks for: a.22projections b.11projections c.90º missing data

- Low frequency:

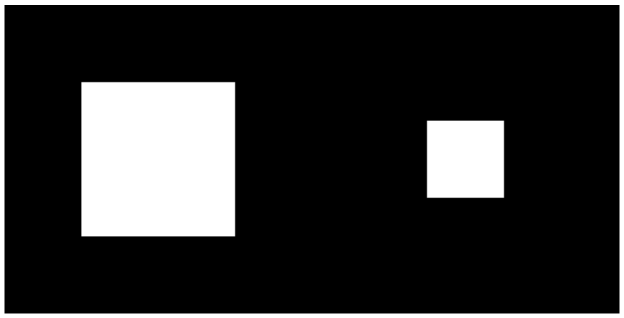

Figure 4.4: Masks: a.128pix b.64pix

- 3D data: in this case we need a group of 2D masks to create a mask for the cone case and the wedge case. Our final 3D masks are shown in the figures below.

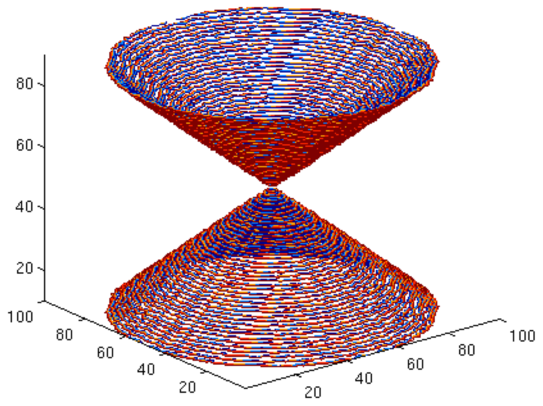

Figure 4.5: 3D cone mask

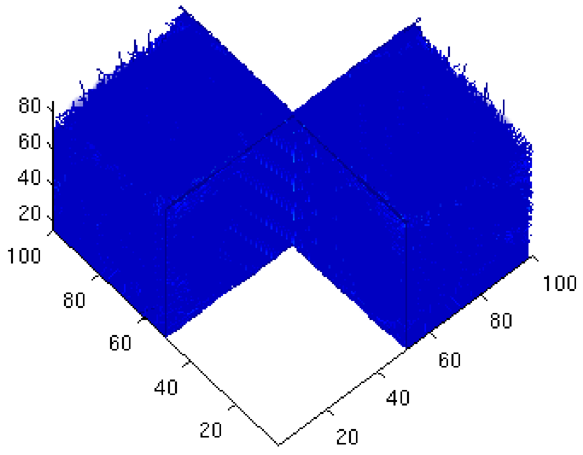

Figure 4.6: 3D wedge mask

In the wedge case all xy-slices are the same, thus we are going to use the same 2D mask in every slice. In contrast, in the other case we have different mask slices for each case. We used xy-slices and xz-slices to create different type of masks for the missing cone, an example of one xy-slice and xz-slice masks is shown below:

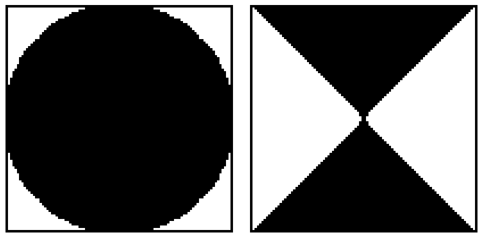

Figure 4.7: Masks: a.xy-slice b.xz-slice

From this point, everything is ready to apply the algorithm to each case.

4.2 Matlab Functions

All the code[35] has been written with MatLab R2010b running on Linux 2.6.32-28-generic Ubuntu x64.

In this section the most important functions are listed and we give a brief explanation of these functions that are contributing in this application somehow to be able to apply the algorithm into the data. Scripts or minor functions are not listed.

▷ List of functions:

- MasterThesis.m ≡ this function performs the main task which is the algorithm[36] itself.

- Filtering.m ≡ it is the filtering block of the algorithm.

- test.m ≡ function that helps to prove that the filter block "Filtering" works properly.

[35]Code is available in Appendix A
[36]Explained in the above subsection

- readraw.m ≡ it is responsible for reading the raw data (*i.e: "protein.mrc" and "hansandrey.raw"*) and gives a readable MatLab format.

- oneDto3Ddata.m ≡ once we have the data in MatLab format, we have to reshape the data from 1D (vector form) to 3D data (cube form), this function does this task.

- slices.m ≡ it is able to extract slices in z, y or x axis of the data cube (3D data).

- fft_slice.m ≡ it does the same as slices.m but instead of giving a slice in the pixel domain, it gives the Fourier transform of the slice.

- fft_slice_fromdisk.m ≡ it does the same as fft_slice.m but instead of using a 3D data as input data, it uses pictures saved in the hard disk.

- cube.m ≡ it is able to create a 3D structure from slices in z, y or x direction.

- fix_mean_noise ≡ with a 2D input data it eliminates background noise from the image.

- mask_creator.m ≡ it is responsible for creating the masks for the unknown region and the known region.

- represent_contours.m ≡ it creates a 3D representation in MatLab of a 3D model input.

- WriteMRC ≡ it writes a Matlab structure, in our case a 3D matrix, in MRC format which one is readable by Chimera.

Method Scheme:

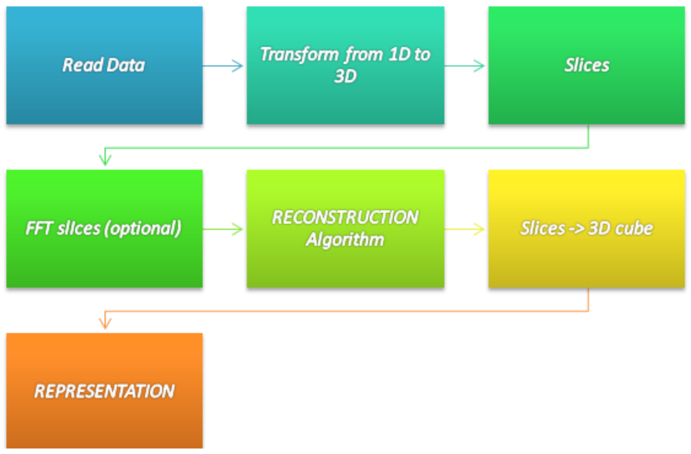

Figure 4.8: Method diagram

5 Analysis

The result of this project is the implementation of a reconstruction algorithm for limited-angle problems like it could be a missing wedge or a missing cone, based in the recent new ideas of Compressed Sensing theory. To evaluate the developed algorithm a number of tests have been conducted. The subjects of these tests differ in what image, object or data we are reconstructing, but the idea is quite similar. To evaluate the reconstruction quality we calculated the PSNR value:

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

MAX_I is the maximum possible pixel value of the image and MSE is the Mean Square Error defined as:

$$MSE = \frac{1}{m \cdot n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left[X(i,j) - X_N(i,j)\right]^2$$

X is the original image and X_N is a reconstruction of the original.

The first round of tests consisted in the performance evaluation of the Shepp-Logan phantom reconstruction, with three different modifications[37] of it. As presented in the above sections, these reconstructions helped to reveal strengths but also imperfections in the method, because in some of the examples different values of one parameter[38] help to obtain a better reconstruction, that is why we did not obtain fixed parameter values for all cases. In this chapter, we describe how these tests were processed and how they revealed important information about the algorithm that we developed.

The next step was to conduct the reconstruction tests using low frequencies. What we wanted to measure was the performance of the algorithm reconstructing high-frequencies. We used some typical pictures of the image processing world, like “cameraman” or “peppers”.

Finally, we applied the algorithm to 3D models of data, one crystallography pattern test[39] and the data we wanted to reconstruct since the beginning.

[37] 1. 22 sparse projections. 2. 11 sparse projections. 3. 90º degrees missing data

[38] Like exciting noise amplitude, decaying noise rule, algorithm step, mask...

[39] several modifications of Hans Andrey protein

5.1 Shepp-Logan Phantom

The first round of tests consist in the reconstruction of the following Shepp-Logan phantom modifications:

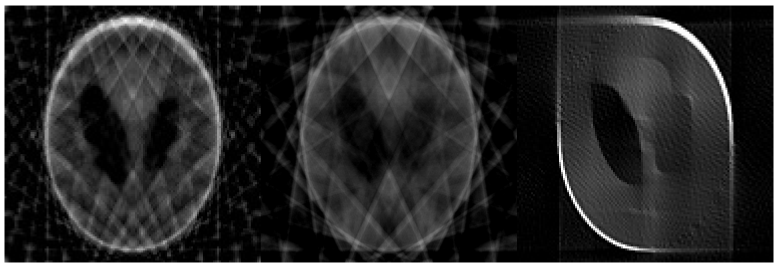

Figure 5.1: Reconstructions of a.22projections b.11projections c.missing cone

To carry out these tests we used the values of the parameters explained in section 4.1, with an image size of 256×256 pixels.

▷ The simplest case is the first one, a. 22lines. We can observe that the spectrum, (5.2), consists in 22 lines equally spaced. Each line represents one of the 22 projections of the specimen.

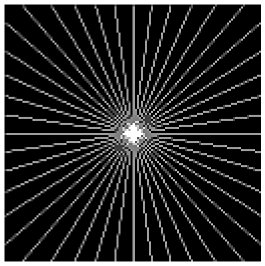

Figure 5.2: FFT 22 projections

Most part of the spectrum is missing, but we have a large amount of energy around zero frequency and 22 contributions throughout the course of frequency. The result in this case was the following one:

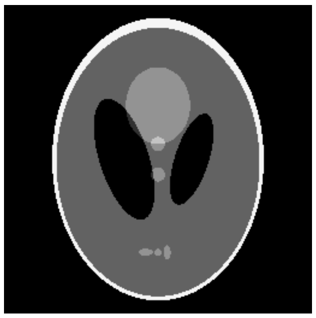

Figure 5.3: Reconstruction FFT 22 projections

The PSNR evolution during the algorithm is shown in the next graph:

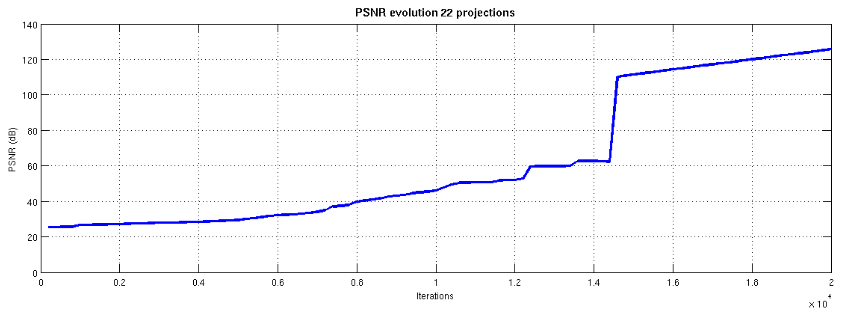

Figure 5.4: Evolution of the algorithm case a.

We can observe that the final value is over $125dB$. With this rate, we can say that the reconstruction is exact. Indeed, it is impossible to detect any difference between the original image and the reconstruction.

▷ The second case, b. 11lines. We can observe that the spectrum, (5.5), consists in 11 lines distributed all around the spectrum which one represents one of the 11 projections of the specimen.

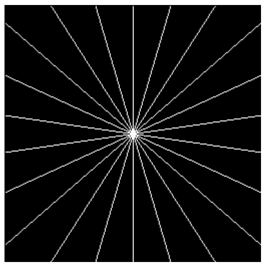

Figure 5.5: FFT 11 projections

Most part of the spectrum is missing, but we have a large amount of energy around zero frequency and 11 contributions throughout the course of frequency. This case is harder than the last one, because it is evident that we have less contributions of known data. The result in this case was the following one:

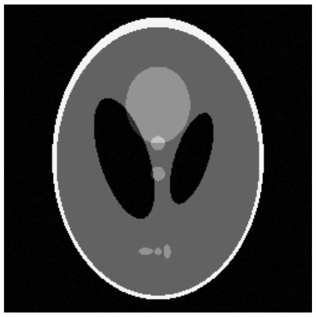

Figure 5.6: Reconstruction FFT 11 projections

The PSNR evolution during the algorithm is shown in the next graph:

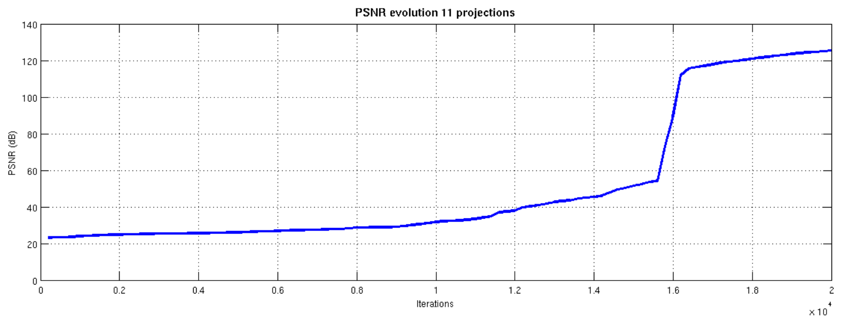

Figure 5.7: Evolution of the algorithm case b.

As a last case, we can observe that the final value is near $125dB$. With this rate, we can say that the reconstruction is exact, because it is impossible to detect any difference between the original image and the reconstruction. But one difference that can be seen immediately that the algorithm takes more time to achieve its best performance. In the last case, the PSNR was rising very fast just after 14000 iterations, in this one that situation happens near 16000 iterations, thus the algorithm is slower.

▷ The last case, c. missing cone, consists in a 90º degree missing data. We can observe in the spectrum, (5.8), that the absence of this part of the spectrum produces on the image, in pixel domain, an elongation by the diagonal which joins the first vertex of the square with its opposite vertex, in mathematical words the line that meets with $x = -y$.

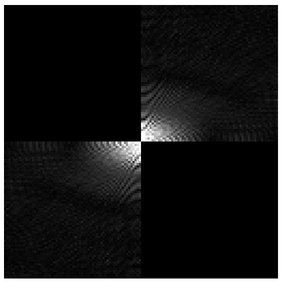

Figure 5.8: 90º missing data

As seen in the other last two cases, a huge part of the spectrum is missing. Moreover the known data is not equally spaced and we do not have any information for $F(-w_x, w_y)$ and $F(w_x, -w_y)$. This case is the hardest to reconstruct. Indeed, we do not have all the energy around frequency zero and usually the greater contribution of energy comes from there. The result is shown below:

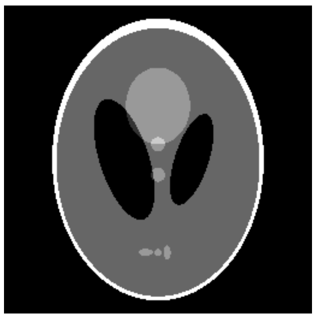

Figure 5.9: Reconstruction 90º missing data

The PSNR evolution during the algorithm is shown in the next graph:

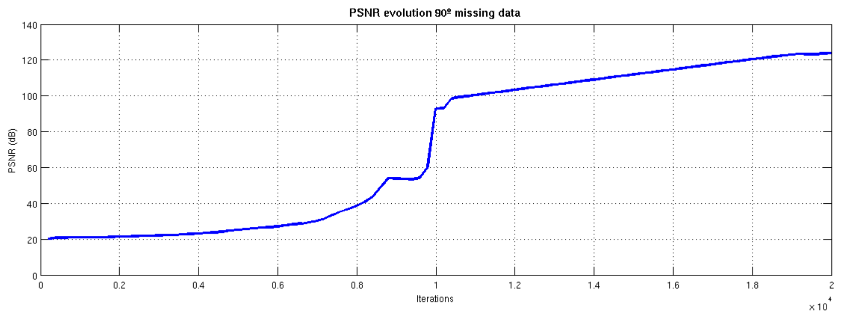

Figure 5.10: Evolution of the algorithm case c.

The results of this last case are almost the same as the other cases. The reconstruction is exact ($\sim 125dB$) and the algorithm in this case is even faster than in the previous cases. As this case is the closest to our problem, we attached in Appendix D the sequence, of the reconstruction. If we observe the reconstruction sequence we can see that between image 9000 and image 10000 the last imperfection of the image is gone. After that, the image quality increases rapidly. As we can see in the graphic, there is a strong ramp between iteration 9000 and 10000 due to the fact we have just explained. Another observation is that near iteration 20000, the algorithm relaxes and the gain is very small.

We can conclude that our method has a very good performance for these last three cases, since as we explained, we obtained accurate reconstructions for all of them. We can check the final reconstructions and see that they are identical. Thanks to these tests we decided to run the algorithm during 20000 iterations for future cases, because from that point the algorithm is near its gain limit.

Finally, we can say that we have reached our first four objectives, we have found a method, we have typed the code to apply the algorithm in 2-dimensional cases and we have run tests to optimize the method and extract conclusions.

5.2 Peppers and Cameraman

The following tests are not included in our first objectives, but we considered that this kind of application is also interesting for some other fields. These group of tests consist in the reconstruction of high frequencies. In order to accomplish this task we used the following pictures applying multiple masks on their spectra:

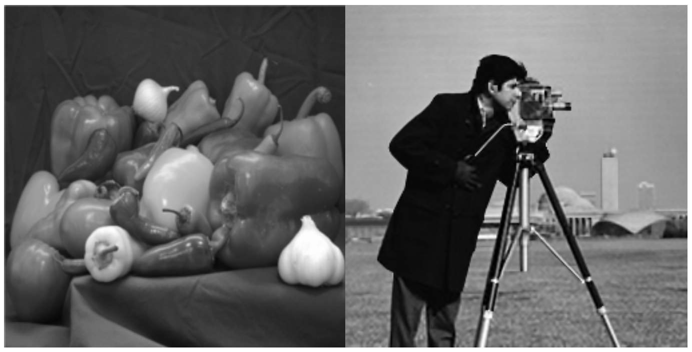

Figure 5.11: Peppers and Cameraman

Our idea was to assess the algorithm performance, reconstructing from a low frequency spectrum the original image. To carry out this test, we applied the masks below, like we presented in section 4.1, on each original spectrum to only obtain the low frequencies of them:

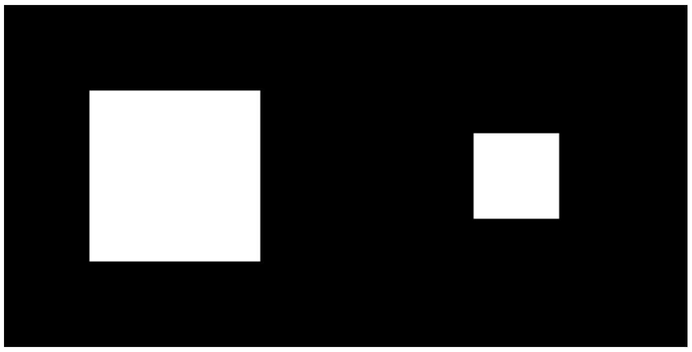

Figure 5.12: Applied masks: a.128pix b.64pix

The white zone represents the frequencies that we select and the black zone represents the frequencies that we neglect.

▷ In the first case, we apply the mask of 128×128 white square on each picture, so we are neglecting around 75% of the spectrum. The images that we obtain are shown below together with their respectives spectra:

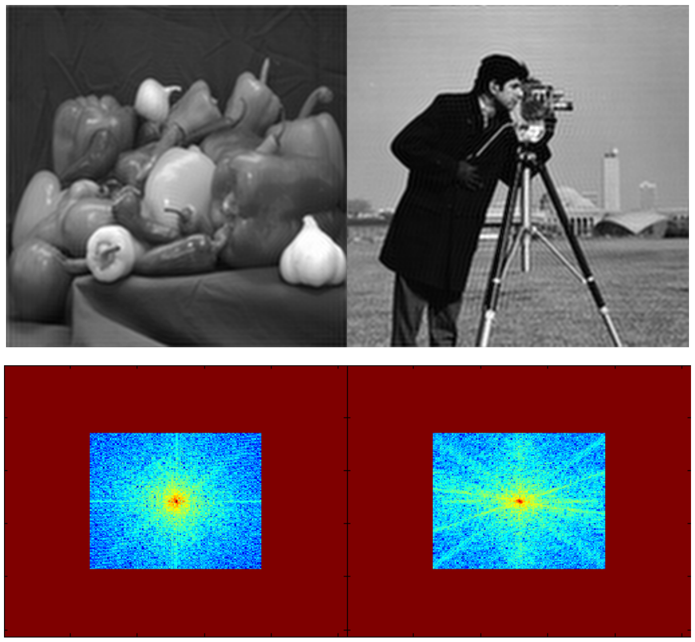

Figure 5.13: Modified peppers and cameraman by mask 128×128

We can observe in above images that around contours and edges appears a strange effect, like a replica of the edge or contour. The reason for this, is the elimination of high frequencies, in high frequencies the information about variations of the contours or edges is stored. Thus it is reasonable that if we eliminate high frequencies, this kind of characteristics may have been altered.

After applying the algorithm during 400 iterations, we obtained an accurate reconstruction of the images:

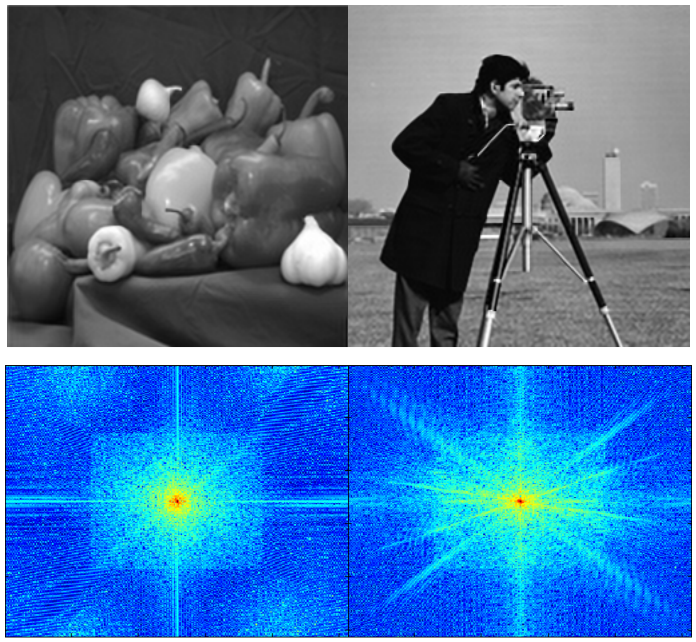

Figure 5.14: Reconstruction of modified peppers and cameraman by mask 128×128

The above reconstructions have a PSNR around 40dB, we did not run the algorithm more iterations because it is hardly difficult to distinguish minor changes in the new reconstructed images. Then, it is proved that we can use this algorithm for the reconstruction from low frequencies like it was seen in the above examples, if we have in our possession a low frequencies square which represents 25% of the data.

$\triangleright$ Using the small mask[40], the algorithm does not succeed in reconstructing the images. It is able to eliminate contours and edges effects. These effects are observable on 5.2, but the reconstructed image definition is not very accurate.

[40]Size of $64x64$ pixels

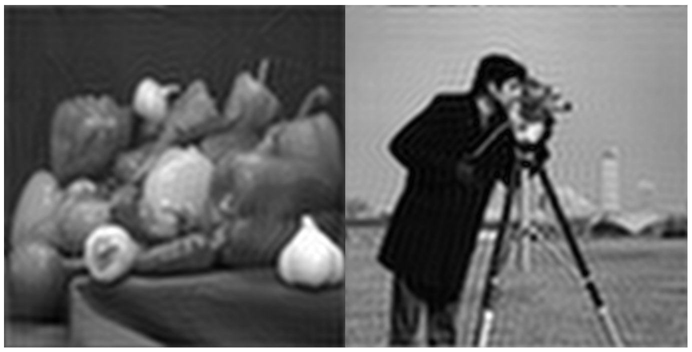

Figure 5.15: Modified peppers and cameraman by mask 64 × 64

Reconstructed images:

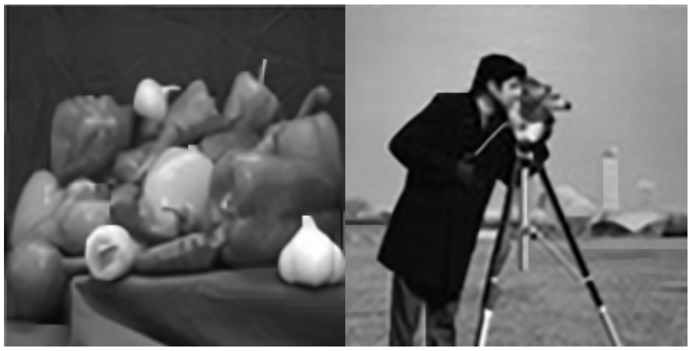

Figure 5.16: Reconstruction of modified peppers and cameraman by mask 64 × 64

We can say that with the small mask, it is not enough to reconstruct the original images. We are able to distinguish between the images, but they present symptoms of blurring and indefinite. The results are not good but at least we deleted the artifacts around the edges and contours.

5.3 Real Data

The last round of tests consisted basically in the reconstruction of a reference model. We based our test in the artificially created Hansandrey crystallography. We modified the original model to have some similar cases as missing cone or missing wedge. To evaluate the results, apart from using the PSNR calculation, we used MatLab. When we could see the models properly, to display the 3D models. When representation was not good enough, we decided to use program

"Chimera" to have better visualization conditions.

After these tests, we showed a possible reconstruction of the first 3D model that we wanted to reconstruct since beginning. As mentioned in previously sections, we can not assure that it is the correct reconstruction. Finally we will show a reconstruction of a viral DNA gatekeeper, where we force the extraction of a 90º cone and then we reconstruct this modification:

5.3.1 Hansandrey model

▷ **Hansandrey missing wedge case:**

In this case we introduced a missing wedge in the frequency domain and we applied the algorithm to each 2D slice in the z-axis to reconstruct the 3D model. The first cube of spectrum that we had, is shown in the next figure:

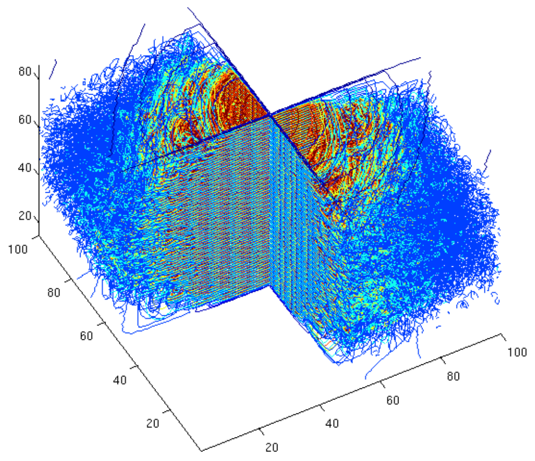

Figure 5.17: Hansandrey missing wedge

The model for this spectrum has a very curious form because of these great transitions caused by the missing wedge:

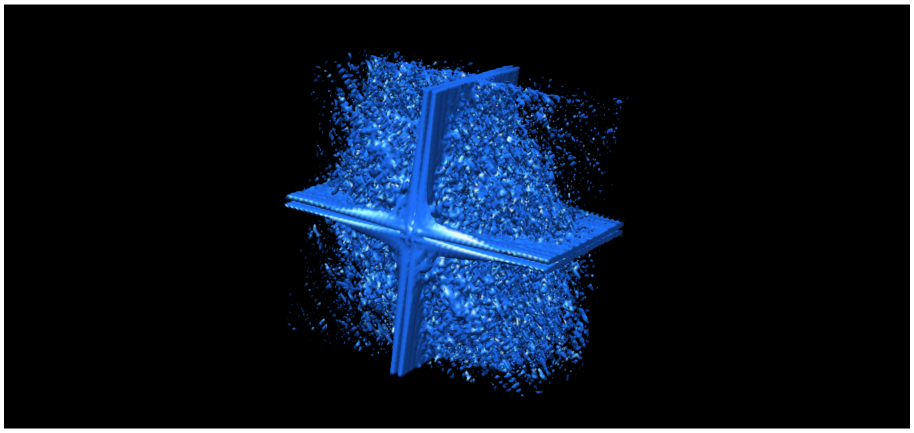

Figure 5.18: Hansandrey appearance missing wedge

Applying the algorithm for this case, we obtained the same problem for every 2-dimensional slice. Indeed if you have a missing wedge around z-axis and you are taking slices around z-axis to apply the algorithm right away, you are always having the Shepp-Logan modification c. 90^{o} missing data.

After running the algorithm, we obtained a reconstruction with a $PSNR = 47.58dB$. We attach the final reconstruction model with the original one:

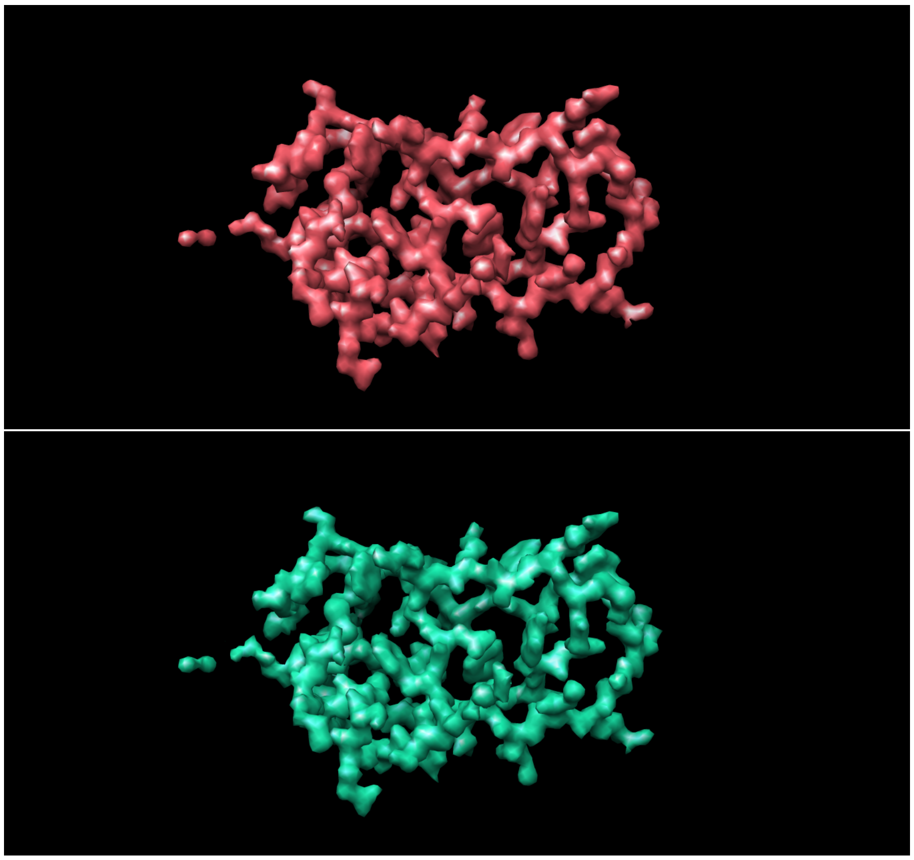

Figure 5.19: a.Hansandrey original model b.Hansandrey reconstruction

We can not really appreciate any difference between the original structure and the reconstructed one. We have achieved a good reconstruction for this case. Finally we are going to present the reconstructed spectrum:

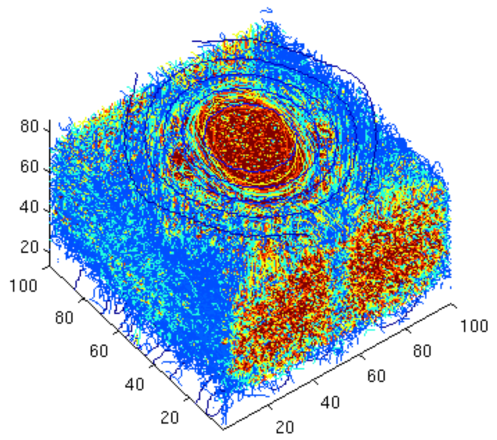

Figure 5.20: Hansandrey spectrum reconstruction

▷ **Hansandrey missing cone case:**

In this case we introduced a missing cone in the frequency domain and we applied the algorithm to each 2-dimensional slice in the z-axis to reconstruct the 3D model. The first cube of data that we had, is shown in the next figure:

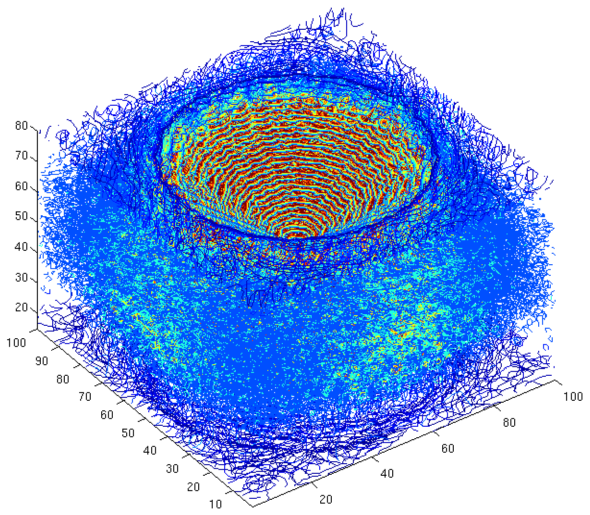

Figure 5.21: Hansandrey missing cone

The main problem in this case is that when you are going further (in the z-axis direction) from the origin, the hole of missing data is getting larger. Thus a great part of information is missing and it is almost impossible for the algorithm to reconstruct anything. The reason lies in the fact that the available known data is not enough to achieve a good reconstruction. The results of this case are shown in the following table[41] with PSNR values of each slice:

[41]The slice number 1 starts in the middle of the structure, where the two missing cones intersect. The model consists of 100 slices from [-50...-1] and from [1...50]

Slice	PSNR(dB)	Slice	PSNR(dB)	Slice	PSNR(dB)	Slice	PSNR(dB)
-50-40	Exact[42]	**-20**	34.95	**1**	124.38	**21**	29.91
-39	Exact	**-19**	33.64	**2**	100.85	**22**	30.07
-38	Exact	**-18**	35.02	**3**	86.91	**23**	29.57
-37	Exact	**-17**	39.99	**4**	40.17	**24**	29.04
-36	Exact	**-16**	49.32	**5**	47.84	**25**	28.83
-35	60.13	**-15**	42.40	**6**	46.55	**26**	30.05
-34	47.34	**-14**	43.79	**7**	45.54	**27**	32.18
-33	42.39	**-13**	40.17	**8**	45.88	**28**	34.38
-32	37.96	**-12**	39.04	**9**	43.38	**29**	36.83
-31	38.36	**-11**	40.23	**10**	39.68	**30**	38.56
-30	38.10	**-10**	40.39	**11**	39.85	**31**	40.30
-29	38.11	**-9**	46.80	**12**	51.47	**32**	41.28
-28	38.01	**-8**	45.09	**13**	42.13	**33**	41.84
-27	35.61	**-7**	42.78	**14**	34.93	**34**	42.02
-26	34.70	**-6**	42.11	**15**	34.89	**35**	46.20
-25	35.64	**-5**	44.27	**16**	33.70	**36**	52.89
-24	35.53	**-4**	95.12	**17**	31.00	**37**	60.33
-23	34.39	**-3**	92.32	**18**	30.52	**38**	Exact
-22	33.57	**-2**	99.10	**19**	30.44	**39**	Exact
-21	34.35	**-1**	86.99	**20**	30.33	**40-50**	Exact

Table 5.1: Missing cone PSNR values of xy slices

After checking the results something caught our attention, the results are improving from slices 27 until the end, although the magnitude of the missing data is increasing (the radius of the missing hole is increasing). The reason of this phenomenon is due to the fact that the spectrum varies widely in each case. While we are approaching to the ends of the structure, the quantity of energy in the image is decreasing, the images are almost completely black[43], i.e they do not contain much energy and it is more feasible to approach, in terms of energy, to the solution. The turning point to change the trend of PSNR is produced in the aforementioned slice number 27. Even if we got good results in the center of the structure and at the end of the structure, the final PSNR of the whole reconstruction was $36.46dB$. This result is not good enough, we can not accept values under $35-40dB$. At the moment, we can not propose this method as a solution.

You can observe a great difference between the original model and the reconstruction:

[43]In appendix E you can find these images

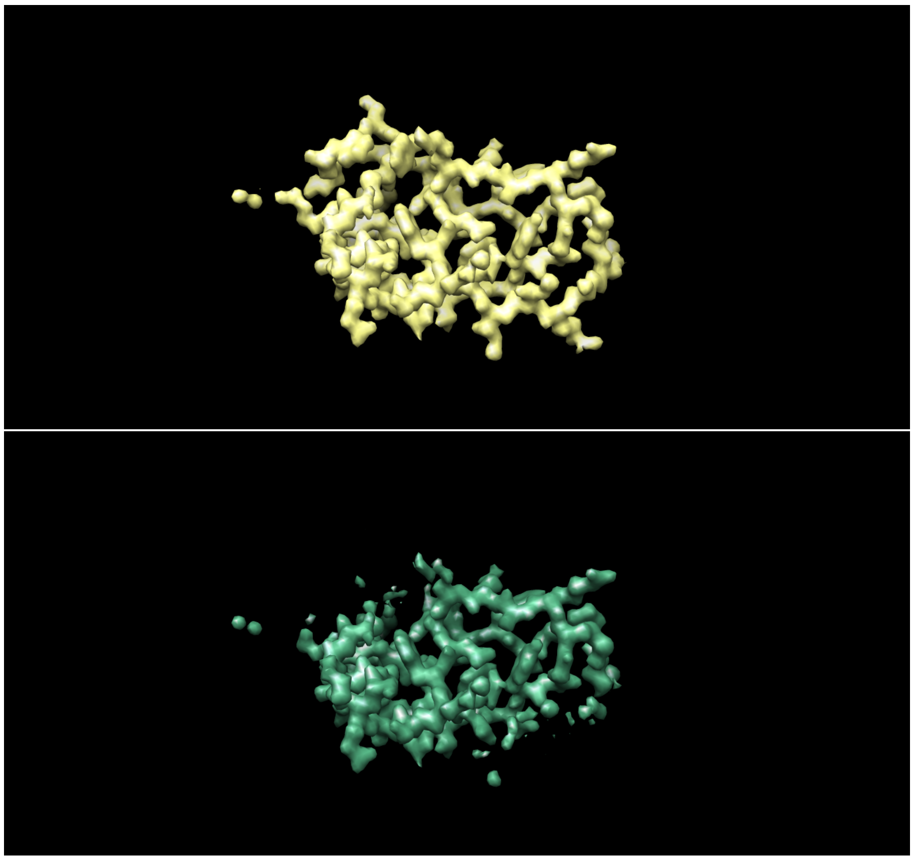

Figure 5.22: a.Hansandrey original model b.Hansandrey reconstruction xy-slices

As you could observe, there are great differences between both models. The half section of the model is the most well approximated zone, as the extremes, because they are practically empty/black images. The zones between the center and the ends of the figure are not well reconstructed[44] as you can see. These effects are consistent with the poor performance of PSNR obtained in those zones.

After that we realized that it was better to obtain xz slices instead of xy slices, like we did in last case. Indeed if we have a missing cone in the z-axis direction and we observe it in the y-axis direction and it is immediate to see that the worst case is to have half of data missing. That happens when we achieve just the middle of the cube. We can see a triangle of missing data[45], in the previous

[44]In Appendix D are attached some slices of the original model and reconstruction, to have a better comparison between them.

[45]Black zone of the figures

case the worst situation had around 80% of missing data:

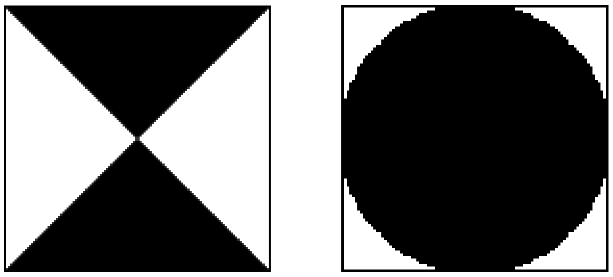

Figure 5.23: Hansandrey worst case: a.xz slices b.xy slices

The reconstruction that we achieved with xz slices had a great PSNR value which is 46.21 dB. In Figure 5.20 we can observe the real Hansandrey model and the reconstruction of it. It is quite hard to detect any differences between them. We can say that the results for this recovery are quite promising because the models are practically identical.

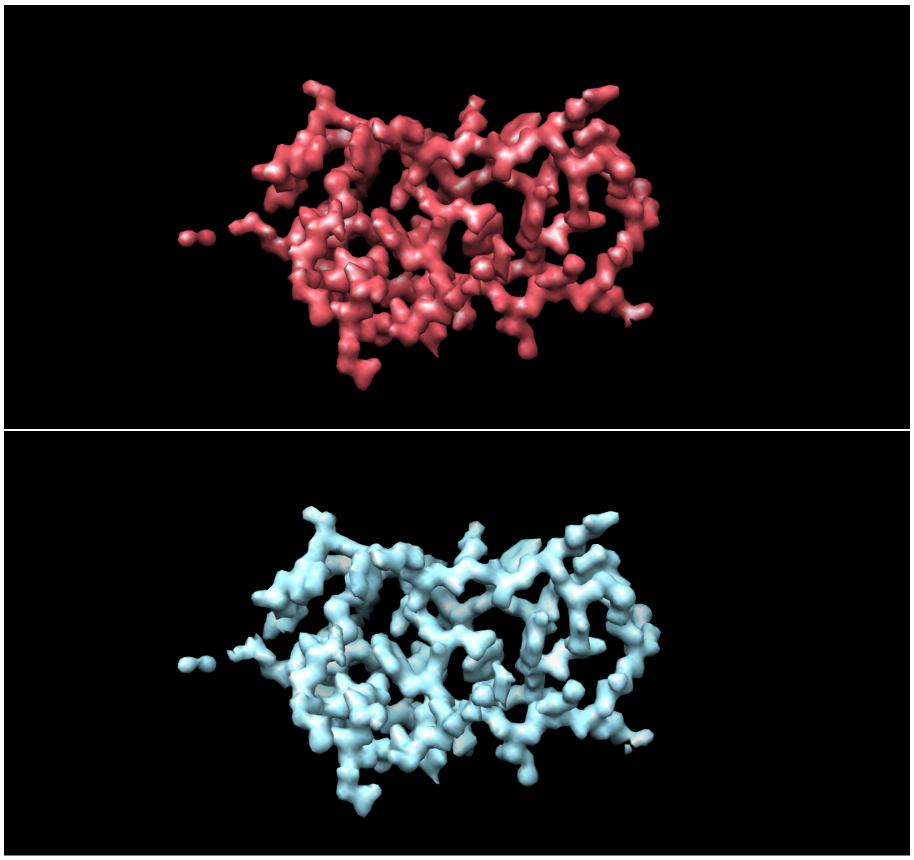

Figure 5.24: a.Hansandrey original model b.Hansandrey reconstruction xz-slices

5.3.2 First model

We apply the algorithm with the same configuration as we used in Hansandrey case. We have to remind that we do not have any idea on how the final model should be. We based our reconstruction proposal on the results of the tests related with Hansandrey crystallography. We think that if the method worked in that model it should also work in this case. The result is shown below:

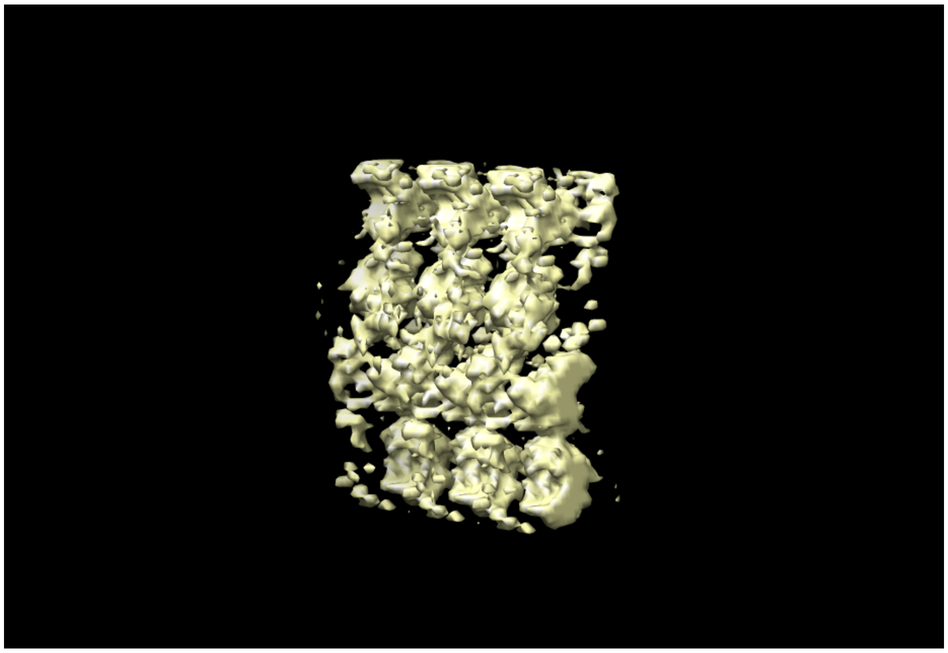

Figure 5.25: First model proposal

If we compare the original model and our proposal we can not see major changes. The model has changed but it is not a drastic change. We can observe in the 3D model, not in this snapshot of the model, that periodicity in x and y axis remains intact but it is possible to see some changes in the connections between the big structures of the model and in the form of these blocks.

In the next figure, we compare both structures (the first model and our proposal), between them we have calculated that there is an 8% of differences, that is a PSNR around $21.89dB$. We can say that we can appreciate some differences between them but it is not a very big change of the structure:

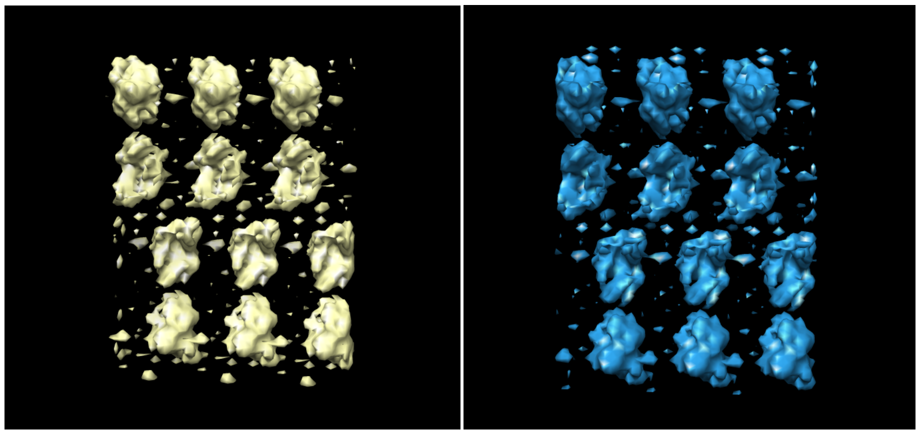

Figure 5.26: Comparison between first model and our proposal

We can observe that the big units of the structure have increased their size and some small units have disappeared in the reconstructed model, other small units have increased their size and some others have decreased their size. Another observation is that the big blocks in the reconstruction are narrower and longer than in the first model.

5.3.3 Viral DNA gatekeeper

The model of this viral DNA gatekeeper was obtained by a cryo-electron microscopy, this technique allows the observation of the specimens that have not been stained or fixed in any way, showing them in their native environment, so it is completely opposite as X-ray crystallography, which generally requires placing the samples in non-physiological environments.

This structure is used as a gate for the *bacillus subtilis bacteriophage SPP1*, which "zips" the capsid after the genome is packaged and unzips it when the virus is ready to infect the host, so the main activity as we mentioned is to control the access to the host. We can observe the shape of the gatekeeper in the following snapshot:

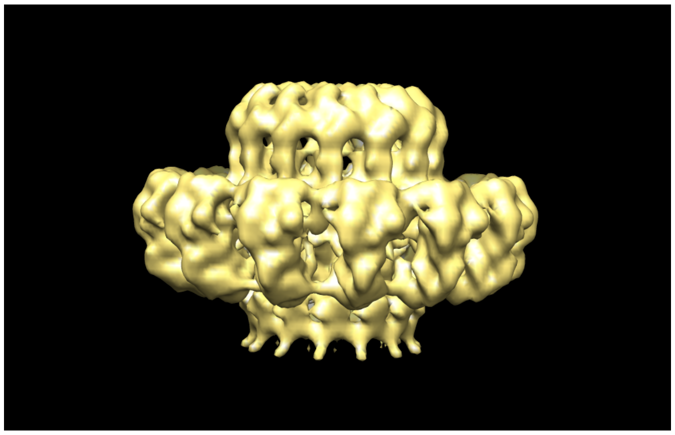

Figure 5.27: Viral DNA gatekeeper (real model)

In the last structure we extract a cone, like the cone in Figure 4.5, and in the model appears a big elongation around the center of the structure due to the great transitions in the Fourier Transform that generates these artifacts in the model in image domain:

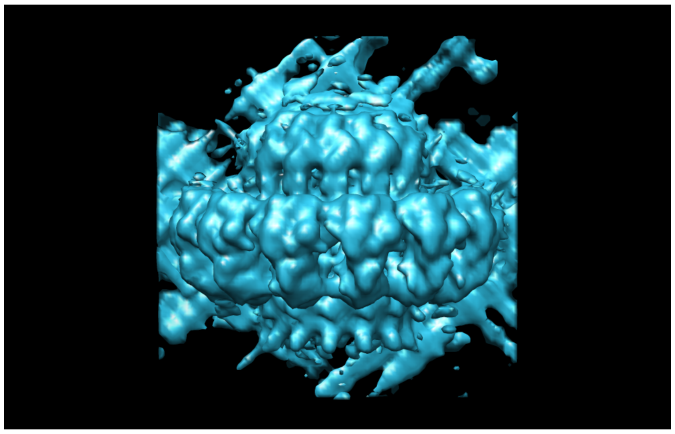

Figure 5.28: Viral DNA gatekeeper (missing cone model)

Then using the model of Figure 5.28, we apply the algorithm to fill the missing cone in the Fourier domain. After applying the algorithm we obtained a PSNR of $45.86dB$ the results of the reconstructions are presented in the next figure:

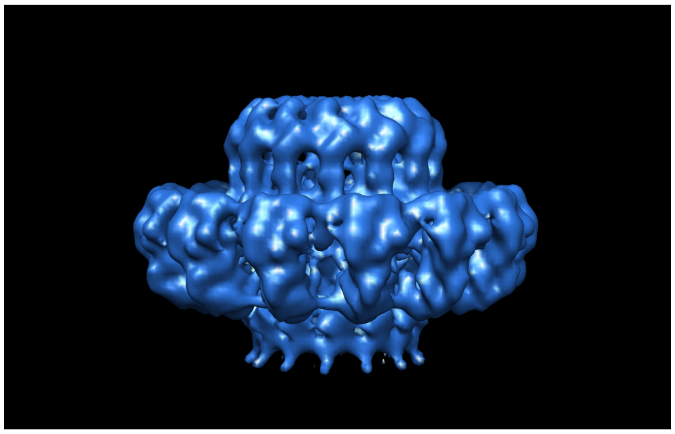

Figure 5.29: Viral DNA gatekeeper (reconstruction)

If we observe the model, we can not detect any big differences between this case and the real case, Figure 5.27. We can say that we have obtained a good reconstruction, because in terms of PSNR we have a great value and the visualization of the model proves that the reconstructed model and the real model are quite close.

6 Conclusions and future work

6.1 Conclusions

The first round of tests for this algorithm implementation works properly in test patterns like the three modifications of the Shepp-Logan Phantom[46]that we have used. In those cases we achieved exact reconstructions. This method, like we proved during the report, is also applicable to low-frequencies spectrum as figures (peppers, cameraman) to reconstruct high frequencies. Such a thing could be applied to improve the quality of the image or to apply super-resolution methods. We can say that we have been able to reproduce a method from a paper, published in [4].

As regards practical results of this work, as we advised they can not be measured because we do not have a previous model to compare with the final reconstruction. The only way to measure the quality or performance of our application has to be focused on the results of tests applied in 3D pattern models.

Focusing in our 3D artificial crystallography model[47], we can say that we are able to fill the missing cone but we have poor results[48] in the half of the first and second cone, in the case that we are taking xy slices from the model. On the other hand, if we are using xz slices instead of xy, we are dealing the missing/corrupted zones in more slices, but then we do not find slices where is almost impossible to recover anything. Ultimately, we do not have large zones of unknown data and we got to rebuild the model from the data with a missing cone in the frequency domain, obtaining a PSNR around $46dB$ (less than 1% of error). The quality of the reconstruction is not perfect but it is a good approximation. It is very hard to notice any difference between the real model and the reconstruction.

When we are reconstructing a structure with a missing wedge around z-axis in the frequency domain, like Figure 5.11, we have also succeeded in these kind of problems, obtaining similar results as the case above with a PSNR around $47dB$. Then, if we compare with one of the alternative, as might be POCS, the results are quite better. POCS results, applied in a Shepp-Logan phantom reconstruction with a missing wedge, are around 40% error rate. We have to add that we can not compare a 3-dimensional case because we have not tested POCS in that kind of scenario.

We have to add a comment about 3-dimensional reconstructions related to the processing time. We realized that is one of the major drawbacks of working with this type of structures (the processing time). To reconstruct the whole Hansandrey model, we needed around 32 hours straight and makes your job

[46]22 radon projections, 11 radon projections and 90º missing data

[47]Hans Andrey protein

[48]Results from Table 6.

harder. If you simply want to change a parameter or if you want to try another mask, etc. Every change brings a great time out. You can check in Appendix E a summary of processing times we have needed in each case.

Finally, we can say that this implementation meets all its goals we considered at first. Thanks to this method we are able to reconstruct spectra with half of the data missing, in form of a wedge, cone, etc. We can also reconstruct spectrum from multiple radon sparse projections[49] or simply reconstruct high-frequencies. We have to comment that the algorithm performance depends in great measure on the spectrum that we want to reconstruct. It is possible to apply the same missing wedge or cone in the same data and doing it in a different position and obtain totally different results. Changing the position of the wedge or cone, it is possible that we are removing an important portion of the spectrum and the algorithm has not enough prior information to reconstruct the structure.

[49]The Fourier Transform of the Radon transform with respect to the projection coordinate equals a radial FFT line (of the unknown function f), according to the Fourier Slice theorem.

6.2 Future Work

1. Like in all other algorithm implementations, one of the most common improvements is to optimize the code. (Try to minimize the execution time and computational resources, because in our case one full reconstruction takes around 32 hours).

2. Develop a version for colored models.

3. Find a way to apply the algorithm directly to 3D data, because in this project we decided to divide the problem in multiple problems in 2D. Our limitation was the filtering block because it is a two dimensional filter.

4. One of the most excitement works that could emerge from this Master Thesis, is the idea to develop a new technique to acquire the data from the EM achieving a large reduction in radiation dose applied to the specimen. Indeed, like we commented in the last subsection, it is enough with half or even less of the spectrum to obtain a model or a picture with a great quality. Therefore, what could be a great advance is finding a way to fill only the necessary frequencies of the spectrum, to let us apply the algorithm and obtain a final image/model that meets our needs. If it is possible to carry this task out, a significant dose reduction can be achieved.

5. Propose new reconstructions for Philip's model[50], changing parameters of the filter (size of fragments, d-distance threshold, size of the block, etc.) or changing common parameters like deviation rule for the excitation noise, noise amplitude, combining reconstruction information of xy-slices and xz-slices, etc.

6. Apply method in other kind of images/models like CT images, MRI, astronomy, geophysical exploration or other type of electron microscope models.

7. Pass MatLab code to low-level languages like C/C++, Java, etc.

[50] "philip.mrc"

7 Bibliography

References

[1] Donoho, D.L., "Compressed sensing", IEEE Trans. Inf. Theory, vol. 52, no. 4, pp. 1289-1306, April 2006.

[2] Tsaig, Y., and D.L. Donoho, "Extensions of compressed sensing", Signal Process., vol. 86, no. 3, pp. 549-571, March 2006.

[3] E. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," IEEE Trans. Info. Th., vol. 52(2), pp. 489– 509, February 2006.

[4] K. Egiazarian, A. Foi, and V. Katkovnik, "Compressed sensing image reconstruction via recursive spatially adaptive filtering," in IEEE Intl. Conf. Image Proc., 2007.

[5] Dabov, K., A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3D transform-domain collaborative filtering", IEEE Trans. Image Process., 2007 (in press)

[6] Chen G-H, Tang J. and Leng S. "Prior image constrained compressed sensing (PICCS): a method to accurately reconstruct dynamic CT images from highly undersampled projection data sets" Med. Phys. 2008

[7] Emil Y Sidky and Xiaochuan Pan "Image reconstruction in circular cone-beam computed tomography by constrained, total-variation minimization" Phys. Med. Biol. 2008

[8] Donoho, D.L., and M. Elad, "Maximal sparsity representation via l1 minimization", Proc. Nat. Aca. Sci., vol. 100, pp. 2197- 2202, 2003.

[9] MRC specification: http://ami.scripps.edu/software/mrctools/mrc_specification.php

[10] Igor Carron Compressed Sensing blog http://sites.google.com/site/igorcarron2/cs

[11] Nuit-Blanche Compressed Sensing blog http://nuit-blanche.blogspot.com/search/label/CS

[12] H. Peng and H. Stark, J. Opt. Soc. Am. A 6 (1989), p. 844.

[13] M.Ibrahim Sezan, An overview of convex projections theory and its application to image recovery problems, Ultramicroscopy, Volume 40, Issue 1, January 1992, Pages 55-67, ISSN 0304-3991, DOI: 10.1016/0304-3991(92)90234-B.

[14] M. I. Sezan and H. Stark, "Tomographic Image Reconstruction from Incomplete View Data by Convex Projections and Direct Fourier Inversion," IEEE Transactions on Medical Imaging, vol. MI-3, no. 2, pp. 91–98, June 1984.

[15] Delaney, A.H.; Bresler, Y.; , "Globally convergent edge-preserving regularized reconstruction: an application to limited-angle tomography," Image Processing, IEEE Transactions on , vol.7, no.2, pp.204-221, Feb 1998

[16] TV minimization demo: http://www.ricam.oeaw.ac.at/people/page/fornasier/

A Appendix A: Matlab code

In this appendix is attached the MatLab code of the main functions of this project. Also there is a brief explanation about each function.

▷ **MasterThesis.m:** this code reproduces the idea published in [4]. It is the main function of our method, because it carries out the iterative algorithm. It needs some output parameters and other external functions that helps to calculate the masks to use or to filter the data, etc.

```
function [ima_fft, ima_fin, PSNR, clock_ini, clock_end] = MT(amplitude, step_size, iterations, image_name, mask_profile, path_fft
%
%
%%%%%%Reconstruction function for multiple missing data cases%%%%%
%
%This function is based in the method published in paper:
%
%"COMPRESSED SENSING IMAGE RECONSTRUCTION VIA RECURSIVE SPATIALLY ADAPTIVE FILTERING"
%Authors: Karen Egiazarian, Alessandro Foi, and Vladimir Katkovnik
%
%This algorithm is one of the options to solve limited-angle problems, like
%limited-angle CT or X-ray, etc.
%
%It was reproduced by Marc Vilà Oliva
%
%
%   INPUT Parameters
%
%   amplitude    -> indicates the amplitude of the exciting noise
%   step_size    -> controls the speed of the algorithm
%   iterations   -> number of iterations
%   image_name   -> is the name of the path where the images are going to be
%   saved
%   mask_profile -> 'normal' 90º degrees mask, 'adaptive' mask which
%   suits with the FFT of the image or 'alternative' is a mixed of the previous ones
%   path_fft     -> path where is located the FFT of the image which we want
%   to reconstruct
%   size         -> size of the input image
%   y            -> input image (generaly we introduce the elongated phantom 128x128
%   pixels)
%   original_ima -> original image, to compute the diference the predicted
%   and the original one
%   after_fft    -> is a flag that if it is active '1' indicates that the
%   first 20 and the last 20 rows are empty
%   sigma_excite -> sigma of the exciting noise - it's saved in the toolbox
%   path
%
%   OUTPUT Parameters
%
%   ima_fft      -> FFT of the predicted image
%   ima_fin      -> predicted image
%   PSNR         ->    value of the PSNR, every 200 iterations, between the original and predicted image
%   clock_ini    ->   saves the moment when the algorithm starts
%   clock_end    ->   saves the moment when the algorithm ends


if (exist('amplitude') ~= 1)
    amplitude        = 25; %% default sigma of the excited noise
end

if (exist('y') ~= 1)
    y         = imread('/home/arvs/Documents/Master_thesis/toolbox_sparsity/phantom128.png'); %% default original image
end

if (exist('step_size') ~= 1)
```

```
    step_size         = 1; %% default sigma of the excited noise
end

if (exist('iterations') ~= 1)
    iterations         = 5000; %% default iterations of the algorithm
end

if (exist('image_name') ~= 1)
    image_name         = '/noise_random/noise_random_it'; %% default iterations of the algorithm
end

if (exist('mask_profile') ~= 1)
    mask_profile         = 'normal'; %% default iterations of the algorithm
end

if (exist('path_fft') ~= 1)
    path_fft         = ' '; %% FT of the image which we one to fix (it's used for the mask creation)
end

if (exist('size') ~= 1)
    size         = 128; %% default size in pixels of the image
end

if (exist('original_ima') ~= 1)
    original_ima         = imread('/home/arvs/Documents/Master_thesis/phantom128.png'); %% default original image
end

if (exist('sigma_excite') ~= 1)
    load /home/arvs/Documents/Master_thesis/toolbox_sparsity/sigma_excite.mat %% default sigma of the exciting noise
end

if (exist('after_fft') ~= 1)
    after_fft = 0;
end
clock_ini = clock;

N=iterations; % Number of iterations

%%%Creation of the masks.
%%%We generate masks to obtain different parts of
%%%the image, y1 (known data) or y2 (unknown data).

[mask, mask_transpose] = mask_creator(mask_profile, size, path_fft);

        filename1 = sprintf('%s%d.png','mask' , 001 );
        imwrite(fftshift(mask),filename1,'png');
        filename2 = sprintf('%s%d.png','mask_transpose' , 002 );
        imwrite(fftshift(mask_transpose),filename2,'png');

y1 = y; %rgb2gray(original_ima) if it is necessary original RGB elongated image to Gray scale
y11 = im2double(y1); %We shrink the values of the image between 0 and 1
y1_t = fft2(y11);

original = original_ima; %rgb2gray(original_ima) if it is necessary original image which we want to reconstruct to gray
original = im2double(original); %We shrink the values of the image between 0 and 1

y1_trans = y1_t.*mask; %We take the known data

imwrite(fftshift(y1_trans),'/home/arvs/Escriptori/primera_.png','png');

y2_ant = uint8(zeros(size));            %First sample of the missing data - all zeros
y2 = fft2(y2_ant).*mask_transpose;      %We obtain the transform of the first sample, all zeros.


for i=1:N  %iterations -> loop

    m = clock;
```

```
if i == 1
    sigma_Filtering = 2*amplitude*sigma_excite(1); %At first iteration we introduce more noise
else
    sigma_Filtering = amplitude*sigma_excite(i-1); %Sigma that it's going to be used by the filter block Filtering
end

sigma_n = sigma_excite(i);  %The standard desviation of the excitation noise has to be reduced while iterations increase.

if i==N
    sigma_Filtering = 0;
end

sec = ceil(m(6));           %We generate a different seed each time we enter in the loop

randn('seed', sec*i);         %We generate the seed of the gaussian noise

ima             = y2 + y1_trans;    %y1 + y2_k

if i==1
    figure(444444);imshow(abs(ifft2(1.1*ima))) %We visualize our first image
    figure(1);imshow(fftshift(abs(ima))) %We check if our first image in fourier domain is the expected
end

z   = ifft2(ima);       %T-1{y1 + y2_k}

[NA, filtered]        = Filtering(1, z, sigma_Filtering); %O[T-1{y1 + y2_k}]

trans           = fft2(filtered);                   %T{O[T-1{y1 + y2_k}]}
y2_pred         = mask_transpose.*trans;            %(1-S).*trans;

noise =  sigma_n*randn(size); % noise with mean '0' and standard deviation sigma_n

if i==N
    noise = zeros(size);
end

error           = step_size*(y2 - y2_pred + mask_transpose.*fftshift(noise)); %step_size*(y2 - y2_pred + (1-S).*noise);

y_k             = y2 - error;

if mod(i,500) == 0  %We represent the evolution of the algorithm each 500 iterations

    ima_fft = y1_trans + y2;
    ima_fin = abs(ifft2(ima_fft));

    if after_fft == 1; %If after_fft is true then we add 0 to the first and last 20 rows
        ima_fin(1:20,:)=0;
        ima_fin(81:100,:)=0;
    end

    if mod(i,5000) == 0
        figure(i+10); imagesc(fftshift(abs(log(ima_fft))));
        filename = sprintf('%s%d.png','/home/arvs/Escriptori/fft_recons_fin_' , i );
        imwrite(fftshift(abs(ima_fft)),filename,'png');
    end

    PSNR(double(i)/500) = 10*log10(1/mean((original(:)-ima_fin(:)).^2)) %We calculate the PSNR between the original image and

    pic_name = image_name;
    filename = sprintf('%s%d.png',pic_name , i );

    [NA2, filtered2]        = BM3D_variant(1, ima_fin, 0);

    filtered2 = min(1,max(0,filtered2));  % mantain values between 0 and 1 after last estimation

    imwrite(filtered2,filename,'png');

    if PSNR(double(i)/500) >= 60 %if PSNR is better than 60dB we end the algorithm
```

```
        break;
        end
    end

    if i == 1
        ima_fft = y1_trans + y2;
        figure(i);imshow(fftshift(abs(ima)))

        ima_fin = abs(ifft2(ima_fft));
        figure(i+10); imagesc(abs(log(fftshift(ima_fft))));

        PSNR(double(i)) = 10*log10(1/mean((original(:)-ima_fin(:)).^2)) % We calculate the PSNR

        pic_name = image_name;
        filename = sprintf('%s%d.png',pic_name , i );
        imwrite(ima_fin,filename,'png');

        if PSNR(i) >= 60 %if PSNR is better than 60dB we end the algorithm
        break;
        end
    end


    y2     = y_k;       %We save the last iteration to use in the next iteration

end

clock_end = clock; %We register the ending time to calculate the total process time



[NA2, filtered2] = BM3D_variant(1, ima_fin, 0); %Last estimation of the image

filtered2 = min(1,max(0,filtered2));  %Mantain values between 0 and 1 after last estimation

imwrite(filtered2,filename,'png');

ima_fft = y1_trans + y2;       %S.*y1_trans + (1-S).*y2;
figure(10000);imagesc(abs(fftshift(log(ima_fft)))); %We visualize last spectrum of the estimation
ima_fin = abs(ifft2(ima_fft));
figure(481516);imagesc(ima_fin); %We visualize last estimation

return;
```

▷ **mask_creator.m:** creates the masks for each case, wedge, cone, 11 or 22 projections and low frequency.

```
function [mask, mask_op] = mask_creator(profile, size, ima)

%Function that creates the masks necessaries to proceed with the
%reconstruction algorithm:
%OUTPUT Parameters
%
%    mask    -> takes the known data
%    mask_op -> takes the unknown data
%
%INPUT Parameters
%
%    mask_profile -> 'normal' 90º degrees mask or 'adaptive' to the fft image
%    ima      -> path of the fft transform from the image which we want to reconstruct

if strcmp(profile, 'normal') == 1

    on = ones (0.5*size);
    zer = zeros(0.5*size);
    media= [zer,on];
    media1= [on, zer];
    mask= [media; media1];           %mask 000000...111111
    mask_op = [media1;media]; %mask 111111...000000

end

if strcmp(profile, 'adaptive') == 1

    rad = imread(ima);
    rad = rad;%rgb2gray(rad);
    rad = im2double(uint8(rad));

    low = min(rad(:))
    fil_col = numel(rad)
    mask = zeros(sqrt(fil_col));

    for i=1:(fil_col)

        if abs(rad(i))> low % normally it is 1/255,
        cause in some pictures the lowest value is 1.

                mask(i) = 1;
                %figure(100);imagesc(mask)

        end
    end

    allones = ones(sqrt(fil_col));
    mask_op = fftshift(allones - mask);
    mask = fftshift(mask);

end

if strcmp(profile, 'alternative') == 1
on = ones (0.5*size);
    zer = zeros(0.5*size);
    media= [zer,on];
    media1= [on, zer];
    mask_sub= [media; media1];     %mask 000000...111111
    mask_op_sub = [media1;media]; %mask 111111...000000

    rad = imread(ima);
    rad = rgb2gray(rad);
    rad = im2double(uint8(rad));
    low = min(rad(:))
    fil_col = numel(rad)
    mask_pre = zeros(sqrt(fil_col));
    allones = ones(sqrt(fil_col));

    for i=1:(fil_col)

        if abs(rad(i))> 0.25;%2.5*low % normally it is 1/255, cause in some pictures the lowest value is 1.

                mask_pre(i) = 1;
                %figure(99);imagesc(mask)

        end

    end

    mask = fftshift(mask_pre);%.*mask_sub;
    figure(100);imagesc(fftshift(mask))
    mask_op = (allones-mask);%.*mask_op_sub;
    figure(101);imagesc(fftshift(mask_op))
end

return
```

▷ **BM3D_variant.m:** this functions is originally taken from [4]. It has been modified by us to optimize processing time and performance of the algorithm. For example we eliminated the Wiener filtering that it was present in the original code and in the 3rd dimension transform we selected Haar wavelet transform instead of Biortogonal1.5 wavelet transform.

```
function [PSNR, y_est] = Filtering(y, z, sigma, profile, print_to_screen)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  This function BM3D_variant is a modification of the BM3D that is an algorithm for attenuation of
%  additive white Gaussian noise from grayscale images. This algorithm reproduces
%  the results from the article:
%
%  [1] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image Denoising
%      by Sparse 3D Transform-Domain Collaborative Filtering,"
%      IEEE Transactions on Image Processing, vol. 16, no. 8, August, 2007.
%      preprint at http://www.cs.tut.fi/~foi/GCF-BM3D.
%
%
%
%
%  INPUT Parameters
%
%     y                 -> is a matrix (MxN) noise-free image (needed for computing PSNR),
%                          replace with the scalar 1 if not available.
%     z                 -> is a matrix (MxN) noisy image (intensities in range [0,1] or [0,255])
%     sigma             -> Std. dev. of the noise (corresponding to intensities
%                          in range [0,255] even if the range of z is [0,1])
%     profile           -> 'np' --> Normal Profile
%                          'lc' --> Fast Profile
%     print_to_screen   -> 0 --> do not print output information (and do
%                                not plot figures)
%                          1 --> print information and plot figures
%
%  OUTPUT Parameters
%     PSNR      -> output PSNR (dB), only if the original
%                  image is available, otherwise PSNR = 0
%     y_est     -> is a matrix (MxN) final estimation (in the range [0,1])
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Copyright (c) 2006-2010 Tampere University of Technology.
% All rights reserved.
% This work should only be used for nonprofit purposes.
%
% AUTHORS:
%     Kostadin Dabov, email: dabov _at_ cs.tut.fi
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% In case, a noisy image z is not provided, then use the filename
%%%%  below to read an original image (might contain path also). Later,
%%%%  artificial AWGN noise is added and this noisy image is processed
%%%%  by the filtering block.
%%%%
image_name = [
%     'montage.png'
     'Cameraman256.png'
%     'boat.png'
%     'Lena512.png'
%     'house.png'
%     'barbara.png'
%     'peppers256.png'
%     'fingerprint.png'
%     'couple.png'
%     'hill.png'
%     'man.png'
    ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  Quality/complexity trade-off profile selection
%%%%
%%%%  'np' --> Normal Profile (balanced quality)
%%%%  'lc' --> Low Complexity Profile (fast, lower quality)
%%%%
%%%%  'high' --> High Profile (high quality, not documented in [1])
%%%%
%%%%  'vn' --> This profile is automatically enabled for high noise
%%%%           when sigma > 40
%%%%
%%%%  'vn_old' --> This is the old 'vn' profile that was used in [1].
%%%%           It gives inferior results than 'vn' in most cases.
%%%%
if (exist('profile') ~= 1)
```

```
    profile         = 'np'; %% default profile
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  Specify the std. dev. of the corrupting noise
%%%%
if (exist('sigma') ~= 1),
    sigma               = 25; %% default standard deviation of the AWGN
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Following are the parameters for the Normal Profile.
%%%%

%%%% Select transforms ('dct', 'dst', 'hadamard', or anything that is listed by 'help wfilters'):
transform_2D_HT_name     = 'haar'; %% transform used for the HT filt. of size N1 x N1
%%We do not use this parameter%% transform_2D_Wiener_name = 'haar'; %% transform used for the
                                 % Wiener filt. of size N1_wiener x N1_wiener
transform_3rd_dim_name   = 'haar'; %% transform used in the 3-rd dim, the same for HT and Wiener filt.

%%%% Hard-thresholding (HT) parameters:
N1                  = 8;   %% N1 x N1 is the block size used for the hard-thresholding (HT) filtering
Nstep               = 3;   %% sliding step to process every next reference block
N2                  = 16;  %% maximum number of similar blocks (maximum size of the 3rd dimension of a 3D array)
Ns                  = 39;  %% length of the side of the search neighborhood for full-search block-matching (BM), must be odd
tau_match           = 3000;%% threshold for the block-distance (d-distance)
lambda_thr2D        = 0;   %% threshold parameter for the coarse initial denoising used in the d-distance measure
lambda_thr3D        = 2.7; %% threshold parameter for the hard-thresholding in 3D transform domain
beta                = 2.0; %% parameter of the 2D Kaiser window used in the reconstruction

%%%% Wiener filtering parameters:
N1_wiener           = 4;
Nstep_wiener        = 3;
N2_wiener           = 16;%%32;
Ns_wiener           = 39;
tau_match_wiener    = 400;
beta_wiener         = 2.0;

%%%% Block-matching parameters:
stepFS              = 1;  %% step that forces to switch to full-search BM, "1" implies always full-search
smallLN             = 'not used in np'; %% if stepFS > 1, then this specifies the size of the small local search neighb.
stepFSW             = 1;
smallLNW            = 'not used in np';
thrToIncStep        = 8;  % if the number of non-zero coefficients after HT is less than thrToIncStep,
                          % than the sliding step to the next reference block is incresed to (nm1-1)

if strcmp(profile, 'lc') == 1,

    Nstep               = 6;
    Ns                  = 25;
    Nstep_wiener        = 5;
    N2_wiener           = 16;
    Ns_wiener           = 25;

    thrToIncStep        = 3;
    smallLN             = 3;
    stepFS              = 6*Nstep;
    smallLNW            = 2;
    stepFSW             = 5*Nstep_wiener;

end

% Profile 'vn' was proposed in
%  Y. Hou, C. Zhao, D. Yang, and Y. Cheng, 'Comment on "Image Denoising by Sparse 3D Transform-Domain
%  Collaborative Filtering"', accepted for publication, IEEE Trans. on Image Processing, July, 2010.
% as a better alternative to that initially proposed in [1] (which is currently in profile 'vn_old')
if (strcmp(profile, 'vn') == 1) | (sigma > 40),

    N2                  = 32;
    Nstep               = 4;

    N1_wiener           = 11;
    Nstep_wiener        = 6;

    lambda_thr3D        = 2.8;
    thrToIncStep        = 3;
    tau_match_wiener    = 3500;
    tau_match           = 25000;

    Ns_wiener           = 39;

end

% The 'vn_old' profile corresponds to the original parameters for strong noise proposed in [1].
if (strcmp(profile, 'vn_old') == 1) & (sigma > 40),

    transform_2D_HT_name = 'dct';

    N1                  = 12;
    Nstep               = 4;
    N1_wiener           = 11;
    Nstep_wiener        = 6;

    lambda_thr3D        = 2.8;
    lambda_thr2D        = 2.0;
    thrToIncStep        = 3;
    tau_match_wiener    = 3500;
    tau_match           = 5000;

    Ns_wiener           = 39;

end

decLevel = 0;        %  dec. levels of the dyadic wavelet 2D transform for blocks
                     %  (0 means full decomposition, higher values decrease the dec. number)
thr_mask = ones(N1); %  N1xN1 mask of threshold scaling coeff. --- by default there is no scaling,
                     %  however the use of different thresholds for different wavelet decompoistion
                     %  subbands can be done with this matrix

if strcmp(profile, 'high') == 1, %% this profile is not documented in [1]

    decLevel     = 1;
    Nstep        = 2;
    Nstep_wiener = 2;
    lambda_thr3D = 2.5;
    vMask = ones(N1,1); vMask((end/4+1):end/2)= 1.01; vMask((end/2+1):end) = 1.07;
    % this allows to have different threhsolds for the finest and next-to-the-finest subbands

    thr_mask = vMask * vMask';
    beta         = 2.5;
    beta_wiener  = 1.5;

end

%%% Check whether to dump information to the screen or remain silent
dump_output_information = 1;
if (exist('print_to_screen') == 1) & (print_to_screen == 0),
    dump_output_information = 0;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Create transform matrices, etc.
%%%%
[Tfor, Tinv]   = getTransfMatrix(N1, transform_2D_HT_name, decLevel);
% get (normalized) forward and inverse transform matrices

if (strcmp(transform_3rd_dim_name, 'haar') == 1) | (strcmp(transform_3rd_dim_name(end-2:end), '1.1') == 1),
    %%% If Haar is used in the 3-rd dimension, then a fast internal transform is used, thus no need to generate transform
    %%% matrices.
    hadper_trans_single_den         = {};
    inverse_hadper_trans_single_den = {};
else
    %%% Create transform matrices. The transforms are later applied by
    %%% matrix-vector multiplication for the 1D case.
    for hpow = 0:ceil(log2(max(N2,N2_wiener))),
        h = 2^hpow;
        [Tfor3rd, Tinv3rd]   = getTransfMatrix(h, transform_3rd_dim_name, 0);
        hadper_trans_single_den{h}         = single(Tfor3rd);
        inverse_hadper_trans_single_den{h} = single(Tinv3rd');
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% 2D Kaiser windows used in the aggregation of block-wise estimates
%%%%
if beta_wiener==2 & beta==2 & N1_wiener==8 & N1==8 % hardcode the window function so
    % that the signal processing toolbox is not needed by default

    Wwin2D = [ 0.1924    0.2989    0.3846    0.4325    0.4325    0.3846    0.2989    0.1924;
        0.2989    0.4642    0.5974    0.6717    0.6717    0.5974    0.4642    0.2989;
        0.3846    0.5974    0.7688    0.8644    0.8644    0.7688    0.5974    0.3846;
        0.4325    0.6717    0.8644    0.9718    0.9718    0.8644    0.6717    0.4325;
        0.4325    0.6717    0.8644    0.9718    0.9718    0.8644    0.6717    0.4325;
        0.3846    0.5974    0.7688    0.8644    0.8644    0.7688    0.5974    0.3846;
        0.2989    0.4642    0.5974    0.6717    0.6717    0.5974    0.4642    0.2989;
        0.1924    0.2989    0.3846    0.4325    0.4325    0.3846    0.2989    0.1924];
    Wwin2D_wiener = Wwin2D;
else
    Wwin2D           = kaiser(N1, beta) * kaiser(N1, beta)'; % Kaiser window used in the aggregation of the HT part
    Wwin2D_wiener    = kaiser(N1_wiener, beta_wiener) * kaiser(N1_wiener, beta_wiener)'; % Kaiser window used
    % in the aggregation of the Wiener filt. part
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% If needed, read images, generate noise, or scale the images to the
%%%% [0,1] interval
%%%%


if (exist('y') ~= 1) | (exist('z') ~= 1)
    y        = im2double(imread(image_name));  %% read a noise-free image and put in intensity range [0,1]
    randn('seed', 0);%% generate seed
    z        = y + (sigma/255)*randn(size(y)); %% create a noisy image
else  % external images

    image_name = 'External image';

    % convert z to double precision if needed
    z = double(z);

    % convert y to double precision if needed
    y = double(y);

    % if z's range is [0, 255], then convert to [0, 1]
    if (max(z(:)) > 10), % a naive check for intensity range
        z = z / 255;
    end

    % if y's range is [0, 255], then convert to [0, 1]
    if (max(y(:)) > 10), % a naive check for intensity range
        y = y / 255;
    end
end



if (size(z,3) ~= 1) | (size(y,3) ~= 1),
    error('Filtering accepts only grayscale 2D images.');
end


% Check if the true image y is a valid one; if not, then we cannot compute PSNR, etc.
y_is_invalid_image = (length(size(z)) ~= length(size(y))) | (size(z,1) ~= size(y,1)) | (size(z,2) ~= size(y,2));
if (y_is_invalid_image),
    dump_output_information = 0;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Print image information to the screen
%%%%
if dump_output_information == 1,
    fprintf('Image: %s (%dx%d), sigma: %.1f\n', image_name, size(z,1), size(z,2), sigma);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Step 1. Produce the basic estimate by HT filtering
%%%%
tic;
y_hat = bm3d_thr(z, hadper_trans_single_den, Nstep, N1, N2, lambda_thr2D,...
    lambda_thr3D, tau_match*N1*N1/(255*255), (Ns-1)/2, (sigma/255), thrToIncStep, single(Tfor), single(Tinv)',...textcolorcomment
    inverse_hadper_trans_single_den, single(thr_mask), Wwin2D, smallLN, stepFS );
estimate_elapsed_time = toc;

if dump_output_information == 1,
    PSNR_INITIAL_ESTIMATE = 10*log10(1/mean((y(:)-double(y_hat(:))).^2));
    fprintf('BASIC ESTIMATE, PSNR: %.2f dB\n', PSNR_INITIAL_ESTIMATE);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Step 2. Produce the final estimate by Wiener filtering (using the
%%%%  hard-thresholding initial estimate)
%%%%
%%%%%
%%%%%

%%%%%%%%%%WE DO NOT USE THE WIENER FILTERING PART%%%%%%%%%%%

%%%%%
%%%%%

%tic;
%y_est = bm3d_wiener(z, y_hat, hadper_trans_single_den, Nstep_wiener, N1_wiener, N2_wiener, ...
%    'unused arg', tau_match_wiener*N1_wiener*N1_wiener/(255*255), (Ns_wiener-1)/2, (sigma/255),...
%    'unused arg', single(TforW), single(TinvW)', inverse_hadper_trans_single_den, Wwin2D_wiener,...
%    smallLNW, stepFSW, single(ones(N1_wiener)) );
%wiener_elapsed_time = toc;

y_est = y_hat;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Calculate the final estimate's PSNR, print it, and show the
%%%% denoised image next to the noisy one
%%%%
y_est = double(y_est);

PSNR = 0; %% Remains 0 if the true image y is not available
if (~y_is_invalid_image), % checks if y is a valid image
    PSNR = 10*log10(1/mean((y(:)-y_est(:)).^2)); % y is valid
end

if dump_output_information == 1,
    fprintf('FINAL ESTIMATE (total time: %.1f sec), PSNR: %.2f dB\n', ...
        wiener_elapsed_time + estimate_elapsed_time, PSNR);

    figure, imshow(z); title(sprintf('Noisy %s, PSNR: %.3f dB (sigma: %d)', ...
        image_name(1:end-4), 10*log10(1/mean((y(:)-z(:)).^2)), sigma));

    figure, imshow(y_est); title(sprintf('Denoised %s, PSNR: %.3f dB', ...
        image_name(1:end-4), PSNR));

end

return;
```

▷ **readraw.m:** is responsible for reading the raw data (*e.g: "protein.mrc" and "hansandrey.raw"*) and give a readable MatLab format.

```
function [data] = readraw(filename, type, size, endian)
%Function to read raw data
%
%INPUT Parameters
%
%filename -> path of the RAW data
%type -> type of data, float, uint, int, binary...
%size -> size of the file
%endian -> litte 'l' or big 'b' endian
%
%OUTPUT Parameters
%
%data -> vector with the RAW data

if( nargin == 3 )
    endian = 'b';
end

fp=fopen(filename, 'rb', endian);
data = fread(fp, size, type);
fclose(fp);

return
```

▷ **oneDto3Ddata.m:** reshapes the data from 1D (vector form) to 3D data (cube form).

```
function [data] = oneDto3Ddata(rawdata, dimaxis, header)
%Function to read raw data and reshape into 3D-data
%
%INPUT Parameters
%
%rawdata    -> vector of RAW data
%dimaxis    -> dimension of cube axis
%header     -> if the data has a header (yes) o not (no)
%
%OUTPUT Parameters
%
%data       -> reshaped 3D-data

k=1;
l=1;
m=1;

ds = size(rawdata);

if strcmp(header, 'yes') == 1
    ite = ds(1) - 256;
else
    ite = ds(1);
end
```

```
for i=1:ite

    if strcmp(header, 'yes') == 1
        data(l,k,m) = rawdata(i+256);
    else
        data(l,k,m) = rawdata(i);
    end

    if mod(k,dimaxis)==0;
        k=0;
        if mod(l,dimaxis)==0
            l=0;
            m=m+1;
        end
        l=l+1;
    end
    k=k+1;
end

return
```

▷ **slices.m:** takes 2D slices of a 3D structures in directions z, y or x.

```
function [] = slices(data, dimaxis, pathname, figure, dim)
%Function that with an INPUT 3D data takes the 2D-slices around the z, y or x axis and
%saves them in the pathname path
%
%INPUT parameters
%
%data       -> 3D data which we want to obtain its slices, it has to be
%normalized between 0 to 1.
%dimaxis    -> dimesion of cube axis
%pathname   -> path where you want to save the slice, automatically in /home
%drive and .png format. Example: '/home/arvs/Documents/name_of_the_picture'
%figures    -> if figure is set to '1' then we save the Matlab figures in .png format instead of
% the common .png images
%dim        -> in which dimension we want to take the slices x, y or
%z(default)

%NOTE: if you are getting images from the file 'philip.mrc' it's going to
%be necessary to multiply the data per 255, cause the values of the data
%are too low and it's impossible to appreciate the original values in the image.

if (exist('figure') ~= 1)
    figure         = 0; %% default figure input
end

if (exist('dim') ~= 1)
    dim         = 'z'; %% default figure input
end


if dim == 'z'
for i=1:dimaxis
    ima_fin = data(:,i,:)/max(data(:)); %We obtain and shrink the data between 0 to 1

    fig = imagesc(ima_fin); %in case we want to save the Matlab figure
    image_name = pathname;

    if figure == 0
        filename = sprintf('%s%d.png',image_name, i );
        imwrite(ima_fin,filename,'png')
    else
        filename = sprintf('%s%d',image_name, i );
        saveas(fig,filename,'png');
    end

end
end

if dim == 'y'
    for i=1:dimaxis
        ima_fin = data(:,i,:)/max(data(:)); %We obtain and shrink the data between 0 to 1
        ima_fin = rot90(squeeze(ima_fin),3); %we have to rotate 270 degrees due to
        %squeeze function, cause it is changing the order of the matrix.

        fig = imagesc(ima_fin); %in case we want to save the Matlab figure

        image_name = pathname;

        if figure == 0
            filename = sprintf('%s%d.png',image_name, i );
            imwrite(ima_fin,filename,'png')
        else
            filename = sprintf('%s%d',image_name, i );
            saveas(fig,filename,'png');
        end

    end
end

if dim == 'x'
    for i=1:dimaxis
        ima_fin = data(i,:,:)/max(data(:)); %We obtain and shrink the data between 0 to 1
        ima_fin = rot90(squeeze(ima_fin),3); %we have to rotate 270 degrees due to
        %squeeze function, cause it is changing the order of the matrix.

        fig = imagesc(ima_fin); %in case we want to save the Matlab figure

        image_name = pathname;

        if figure == 0
            filename = sprintf('%s%d.png',image_name, i );
            imwrite(ima_fin,filename,'png')
        else
            filename = sprintf('%s%d',image_name, i );
            saveas(fig,filename,'png');
        end

    end
end

return
```

▷ **fft_slice.m:** gives the Fourier transform of the 2D slices of a 3D structure.

```
function [] = fft_slice(data, dimaxis, pathname)
%Function that gives the FFT transform of each slice of 3D data and saves
%the FFTs in the "filename" path.
%
%INPUT parameters
%
%data       -> 3D data which we want to obtain its slices, so the data is a
%variable in matlab
%dimaxis    -> dimesion of cube axis
%pathname   -> path where you want to save the slice, automatically in the /home/arvs/Documents
%path and .png format. Example: '/(/backslash/)path/(/backslash/)name_of_the_picture'

for i=1:dimaxis
    noise =  0; % It is optional to add noise
    m = clock;
    sec = ceil(m(6));             % We generate a different seed each time we enter in the loop
    randn('seed', sec*i);         % We generate the seed of the gaussian noise

    slice_fft = fft2(data(:,:,i)+noise);  % We call the function fft2 to proceed with the fourier transform in 2D
    ima_fin=(1/max(abs(slice_fft(:))))*fftshift(abs(slice_fft));% Shift to get the f=0 in the
                               % center of the picture and shrink data between 0 to 1

    image_name = pathname;
    filename = sprintf('%s%d.png',image_name, i );
    imwrite(ima_fin,filename,'png');
end
return
```

▷ **fft_slice_fromdisk.m:** does the same as the last function but instead of taking 2D slices of a 3D structures, takes the slices from a given path.

```
function [image] = fft_slice_fromdisk(data, num_images, image_format)
%Function that with an INPUT 2D data from a path makes the fft.
%
%INPUT parameters
%
%data -> 2D data which we want to obtain the fft
%num_images -> total number of images that we want to obtain the fft
%image_format -> format of the images
%
%OUTPUT parameters
%
%image -> fft of the last image
%
for i=1:num_images

    noise = 0; % optional to introduce noise (double(1.0)/255)*randn(80);
    m = clock;
    sec = ceil(m(6));             % We generate a different seed each time we enter in the loop
    randn('seed', sec*i);         % We generate the seed of the gaussian noise

    data_new = sprintf('%s%d.%s',data, i, image_format );

    image = double(imread(data_new))/max(data_new(:));% We read the image which we want to obtain the fft
                                                      % and we shrink the values between 0 to 1
    slice_fft = fft2(image+noise); % We call the function fft2 to proceed with the fourier transform in 2D
    ima_fin=fftshift(abs(slice_fft))/max(slice_fft(:)); % Shift to get the f=0 in the
                               % center of the picture and shrink data between 0 to 1
    image_name = '/home/arvs/Escriptori/Philip_Meeting/fft_trash_';

    filename = sprintf('%s%d.%s',image_name, i, image_format );
    imwrite(ima_fin,filename,image_format); %we save the image in the specified path by filename
end
return
```

▷ **represent_contours.m:** represents 3D structures in MatLab.

```
function [] = represent_contours(data, num_lines, iso_level, first_slice, last_slice)
%Function to represent 3D structures
%
% INPUT Parameters
%
```

```
% data           -> the 3D-data that we are going to represent
% num_lines      -> number of lines per contour
% iso_level      -> level of the iso value
% first_slice    -> which slice is going to be the first
% last_slice     -> which slice is going to be the last
% reconstruct
%
figure(1);

contourslice(data,[],[],[first_slice:last_slice],num_lines);
view(3);
daspect([1 1 1]);
axis tight

figure(2);

isosurface(data, iso_level, data);
view(3);
daspect([1 1 1]);
axis tight

return
```

▷ **writeMRC.m:** converts a MatLab 2D or 3D structure into a MRC file.

```
function writeMRC(data, size, path_name)
%
% Write out a 2D image or a 3D volume as an MRC map file, for example for viewing in
% Chimera.  'data' is the 3D array, size is the voxel size in angstroms.
%
% INPUT Parameters
%
% data -> is the 3D structure that we want to convert into MRC structure
% size -> is the size, in Ångströms, of the voxels in the new representation
% path_name -> where do you want to save the MRC structure


q = typecast(int32(1),'uint8');
machineLE=(q(1)==1);  % true for little-endian machine

hdr=int32(zeros(256,1));

sizes=size(data);

if numel(sizes)<3
    sizes(3)=1;
end;
if nargin >3
    sizes(3)=nim;
end;

% Get statistics

data=reshape(data,numel(data),1);  % convert it into a 1D vector
theMean=mean(data);
theSD=std(data);
theMax=max(data);
theMin=min(data);


hdr(1:3)=sizes; % number of columns, rows, sections
hdr(4)=2;  % mode: real, float values
hdr(8:10)=hdr(1:3);  % number of intervals along x,y,z
hdr(11:13)=typecast(single(single(hdr(1:3))*size),'int32');  % Cell dimensions
hdr(14:16)=typecast(single([90 90 90]),'int32');   % Angles
hdr(17:19)=(1:3)';  % Axis assignments
hdr(20:22)=typecast(single([theMin theMax theMean]'),'int32');
hdr(23)=0;  % Space group 0 (default)

if machineLE
    hdr(53)=typecast(uint8('MAP '),'int32');
    hdr(54)=typecast(uint8([68 65 0 0]),'int32');  % LE machine stamp.
else
    hdr(53)=typecast(uint8(' PAM'),'int32');  % LE machine stamp, for writing with BE machine.
    hdr(54)=typecast(uint8([0 0 65 68]),'int32');
end

hdr(55)=typecast(single(theSD),'int32');

handle=fopen(path_name,'w','ieee-le');
count1=fwrite(handle,hdr,'int32');
fclose(handle);

return;
```

▷ **test.m:** tests if the filtering block is working properly.

```
%Function that it helps to prove that the filter block "Filtering" works
%properly. To check that, we use one of the common pictures in image
%processing "cameraman.jpg" in which one we introduce gaussian noise
%that it is filtered by the filter block...and the results are saved
%in the variable filtered, original and in the PSNR factor.

function [filtered, origi, PSNR] = test(path, sigma)

if (exist('sigma') ~= 1)
    sigma         = 25; %% default sigma
end

if (exist('path') ~= 1)
    path         = 'E:\toolbox_sparsity\cameraman.jpg'; %% default path
end

m = clock;
sec = ceil(m(6)); % we generate a different seed each time we enter in the loop
randn('seed', sec); %We generate the seed of the gaussian noise

original = imread(path);
origi = double(original/256);
noisy = origi + (sigma/256)*randn(size(original));

figure(10);imshow(double(original/256)); title ('%%ORIGINAL%%');
figure(11);imshow(noisy); title ('%%NOISY%%');



[NA, filtered]= BM3D(1, noisy, 25);
PSNR = 10*log10(1/mean((origi(:)-filtered(:)).^2))
figure(12);imshow(filtered); title ('%%FILTERED%%');

return
```

B Appendix C: Standard Deviation graphics

In this appendix we show the standard deviation graphics of the excitation noise which we apply in different cases.

▷ 22 and 11 projections case:

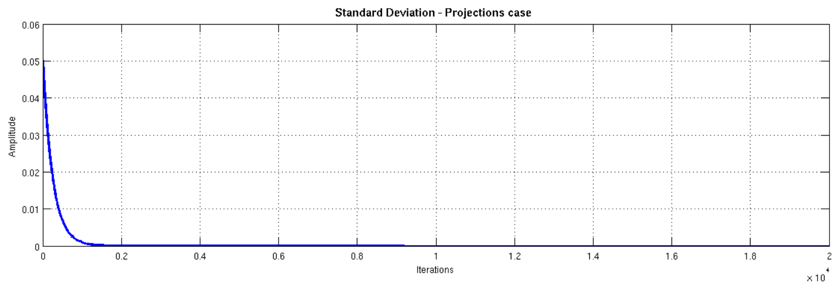

Figure B.1: Standard deviation projections case

▷ 90º missing data case:

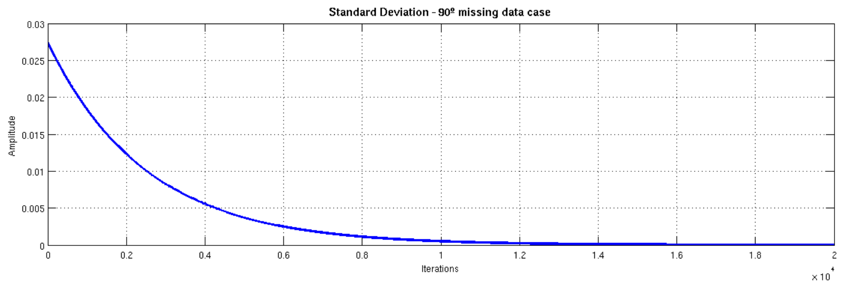

Figure B.2: Standard deviation 90º case

▷ Low frequency case:

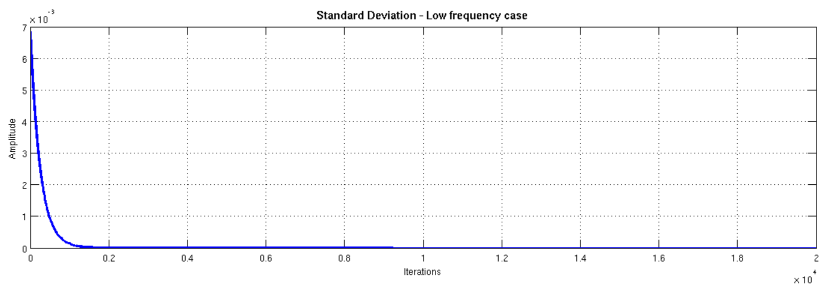

Figure B.3: Standard deviation low frequency case

▷ 3D data case:

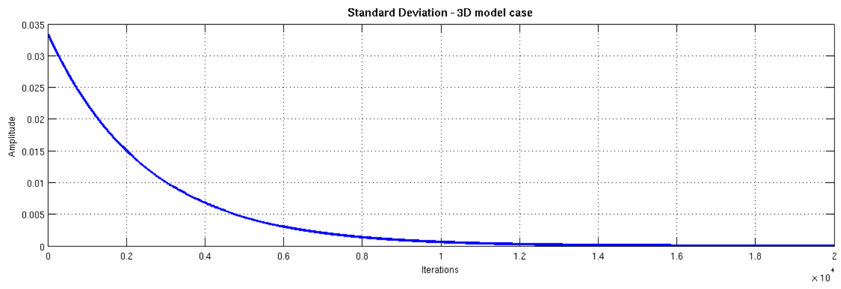

Figure B.4: Standard deviation 3D case

C Appendix C: Reconstruction of the Shepp-Logan Phantom

In this appendix is shown the reconstruction sequence of Shepp-Logan phantom of section 5.1 case c:

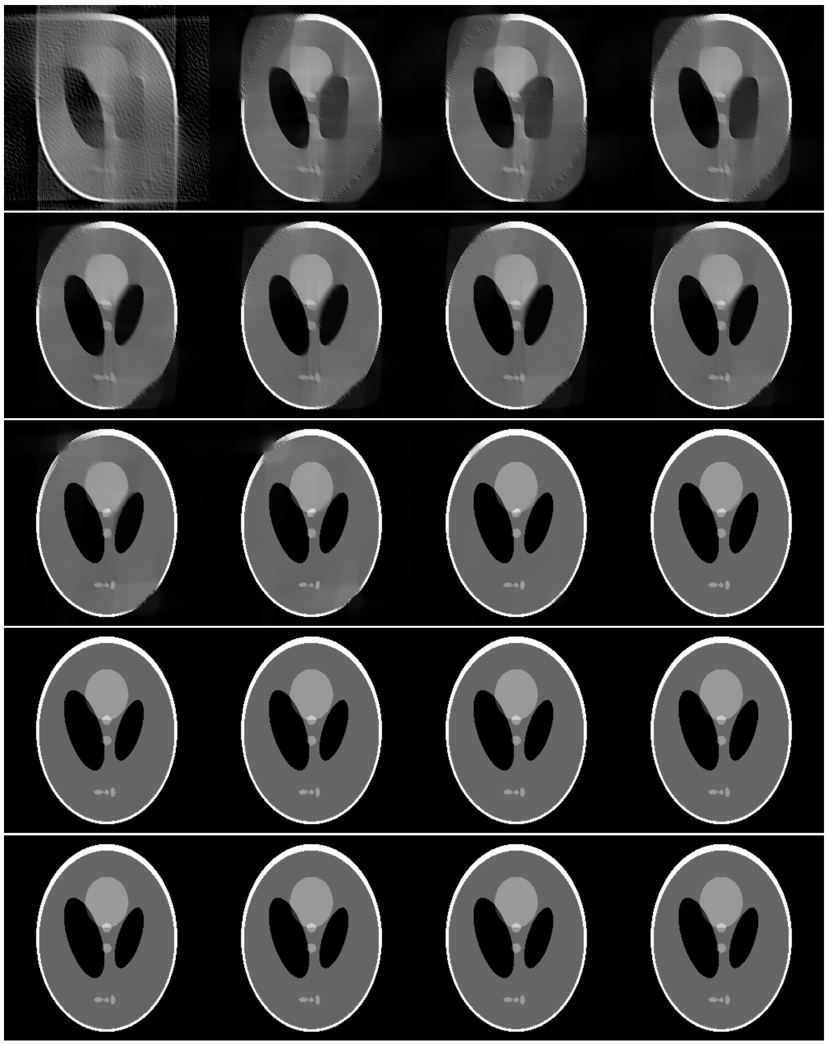

Figure C.1: Shepp-Logan reconstruction. Slices 1, 1000, 2000..., 19000.

D Appendix D: 3D-data

▷ In the following section is shown a group of **xy-slices** from *original* 3D model "hansandrey.raw":

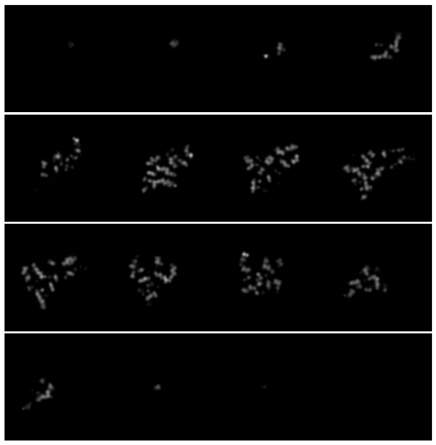

Figure D.1: Hansandrey xy-slices 15, 20...85, 90.

▷ In the following section is shown a group of **xy-slices** from *reconstructed* 3D model "hansandrey.raw":

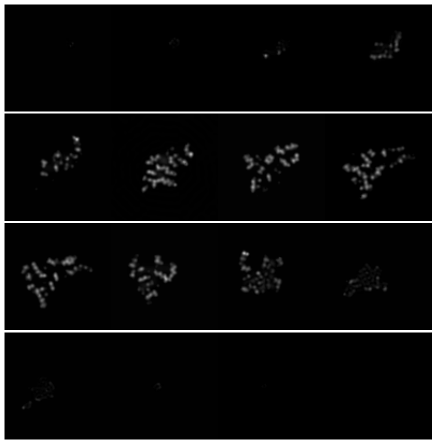

Figure D.2: Hansandrey reconstructed xy-slices 15, 20...85, 90.

▷ In the following section is shown a group of **xz-slices** from *original* 3D model "hansandrey.raw":

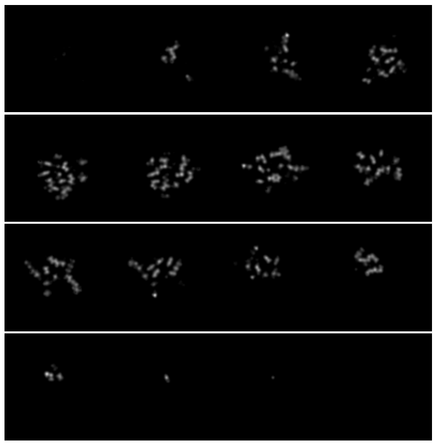

Figure D.3: *Hansandrey xz-slices 15, 20...85, 90.*

▷ In the following section is shown a group of **xz-slices** from *reconstructed* 3D model “hansandrey.raw”:

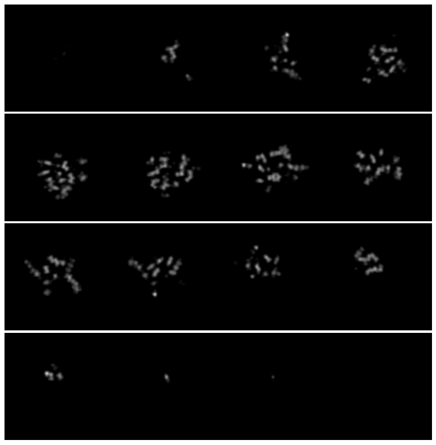

Figure D.4: *Hansandrey reconstructed xz-slices 15, 20...85, 90.*

▷ In the following section is shown a group of slices from 3D model “philip.mrc”:

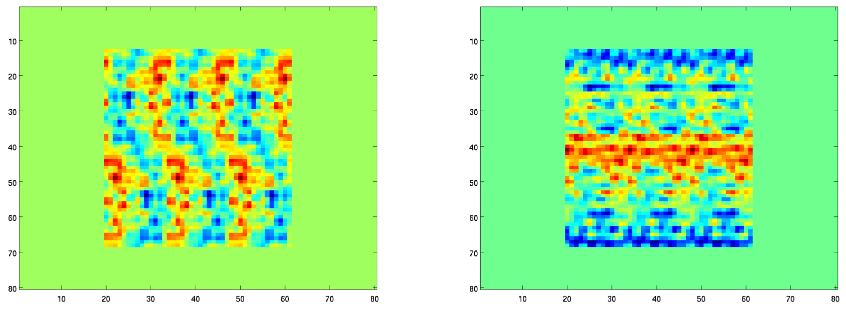

Figure D.5: *Protein slice #26 and #29*

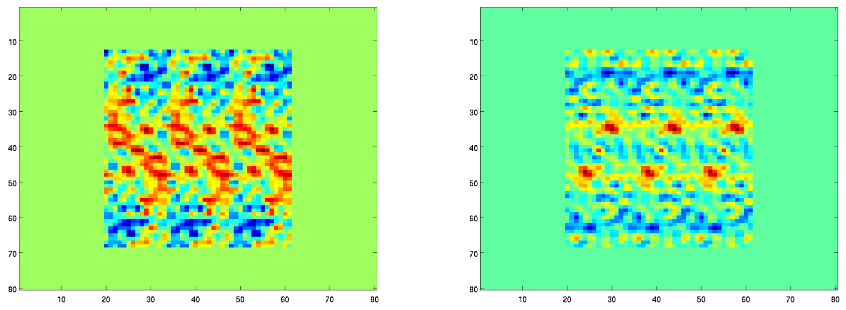

Figure D.6: *protein slice #32 and #35*

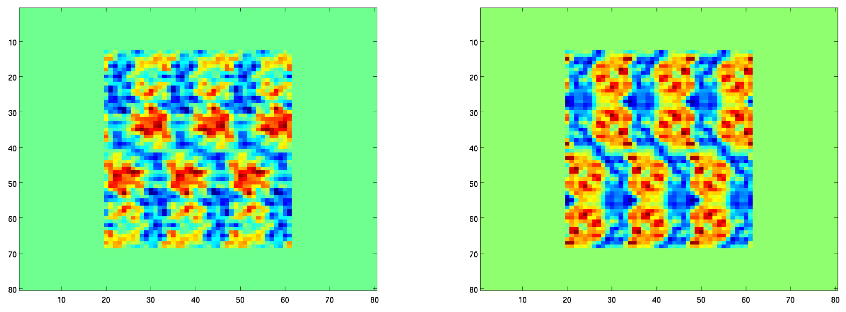

Figure D.7: *Protein slice #38 and #41*

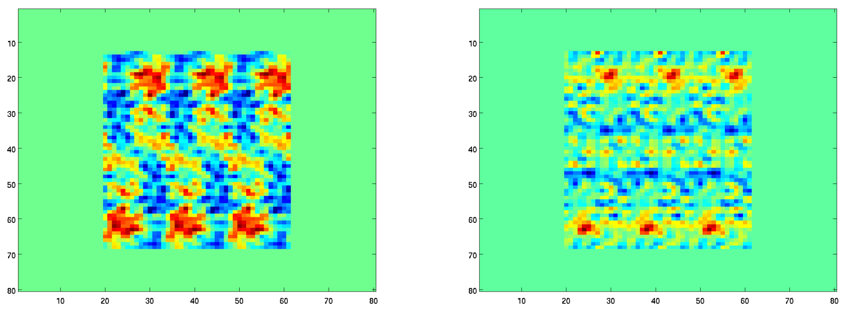

Figure D.8: Protein slice #44 and #47

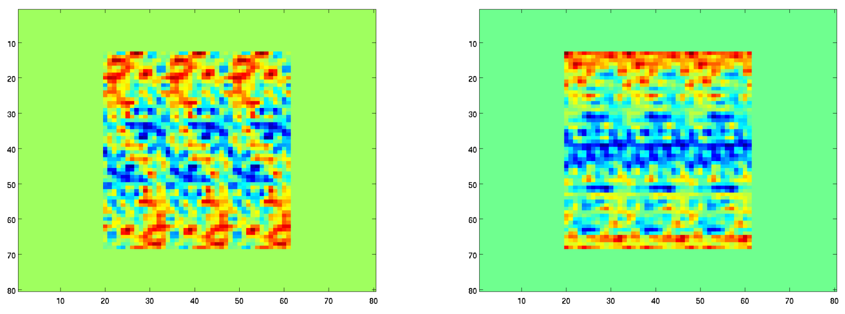

Figure D.9: Protein slice #50 and #53

E Appendix E: Processing Time

We have not talked about the processing time of the algorithm, because we did not think that it was an important issue of the project. However we want to give an approximate idea about it with the following graphic where is shown the needed processing time to reconstruct an image of “x” pixels with 20000 iterations:

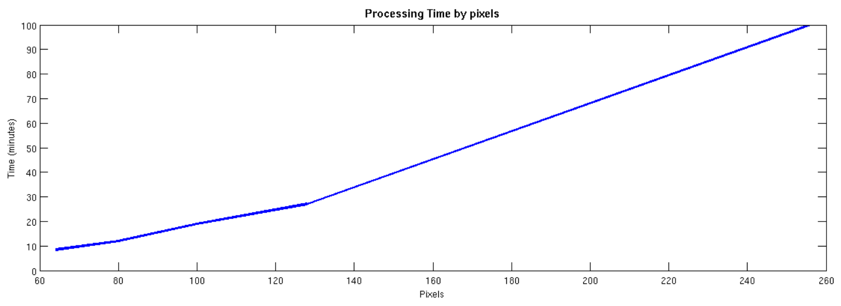

Figure E.1: Processing Time by pixels

As you can see to reconstruct an image of 256×256 we need almost 100 minutes. It is what we spend to reconstruct Shepp-Logan phantom.

For Hansandrey crystallography we spent for each slice around 19 minutes. That model consisted in 100 slices, then the total amount of time is $100 \times 19 = 1900$ minutes. This is a total of approximately 32 hours non-stop processing.

We resume all cases in this table:

Case	Processing Time
Image 64 pixels	8min 30seg
Image 80 pixels	12min
Image 100 pixels	19min
Image 128 pixels	27min
Images 256 pixels	100min
Hansandrey	31h 40min
Philip	16h

Table E.1: Processing time