Projet CVN

SUBJECT :

Large Scale Image Restoration

Hugo Thevenet

Ibrahim Enouali

Filip Razek

Müge Gökçen

CentraleSupélec

université PARIS-SACLAY

Contents

1 Abstract **4**

2 Introduction **5**

3 General resolution strategy **6**

3.1 Optimization Problem Statement 6

3.2 Majorization Minimization (MM) Principle 7

3.3 Subspace acceleration : the 3MG scheme 8

3.4 Combining subspace strategy with a block approach : BP3MG scheme 8

3.5 BD3MG algorithm 9

4 Main code **10**

4.1 Presentation 10

4.2 Results 10

5 Computation example with BD3MG and BP3MG **11**

5.1 Presentation 11

5.2 Output logs 12

5.3 Results 12

6 Speedup **13**

6.1 How the algorithm is supposed to work 13

6.2 How it works 14

6.3 Conclusion 17

7 Some results **17**

7.1 Flybrain 17

7.2 ImageSmall 17

7.3 Aneurysm 19

8 Remarks **20**

9 Conclusion **20**

10 Appendix : Supercomputer **22**

10.1 Introduction 22

10.2 Connecting 22

10.3 Batch 23

11 Appendix : File management **25**

1 Abstract

Image restoration is a way of repairing the inventive image by eliminating noise and image blur. Image blur is difficult to avoid in many cases, such as shooting to eliminate motion blur caused by camera stillness, imaging of measuring devices to eliminate the result of replicating the image pattern, etc. The aim of image restoration is to divine the innovative image from the surveillance image stripped of preservation blur and noise, as far as possible. In the field of mathematical optimization, one classical way to deal with image restoration issues consists in solving a so-call inverse problem by means of a cost function for which we try to reconstruct the "source" (the ideal noiseless image) from the accessible data (the noisy one).
In this paper we compare the performance of three optimization algorithms using the Majorization-Minimization (MM) principle so as to deal with differentiable but non-necessary convex image processing problems ; the classic MM Memory Gradient (3MG), the Block Parallel Majorize-Minimize Memory Gradient (BP3MG) and the Block Distributed MM Memory Gradient version (BD3MG). Our goal is here to promote the use of the MM parallel methods, especially BD3MG scheme, in large scale optimization context mainly using numerical illustrations.

2 Introduction

In general, image restoration is the process of making a degraded real image interpretable enough given a particular physical context using mathematical methods. Its fields of application are very numerous, and especially plenty of microscopic and medical images are restored in this way in order to guarantee better diagnosis and treatments. These reconstruction strategies are appreciated as some of them ensure some theoretical guarantees, and thus a certain fidelity on the data returned to the professionals.
Nowadays, the increasing performance of the data acquisition process promotes the manipulation of very-high qualities images and especially those able to represent 3D objects. Downstream and considering the reconstruction process, this implies the manipulation of high-dimensions data. In a clinical context, it is not uncommon to work with volumes comprising hundreds of slices with a size of 1024×1024, finally given an order of magnitude of 10^9 for the total number of pixels considered. Dealing with this kind of dataset thus requires the use of specific methods from the field of large-scale optimization. One common way to proceed is to adopt parallel optimization techniques to overcome memory limitation issues. Meanwhile, they can make use of the intrinsic acceleration that recent multicore computing architectures provide them. [2].
For the large-scale optimization problem, there are some algorithms to be discussed in the next section. These are Majorization Minimization Memory Gradient (3MG), Block Parallel Majorize-Minimize Memory Gradient (BP3MG), and Block Distributed Majorize-Minimize Memory Gradient (BD3MG) algorithms.
The works on the subject of Majorization-Minimization (MM) methods show us that a faster convergence rate and decreased need for parameter tuning can be provided with block alternating algorithms by considering the cost function structure.
In this project, the goal is designing and conducting experiments of the block distributed 3M algorithms, and there are a couple of examples for 3D image restoration. We aim to determine and develop the scalability and the versatility of the optimization algorithms mentioned in the articles. We try to implement the algorithms with the Python programming language. For this purpose, parallel computing resource which is Mesocenter from University of Paris Saclay are used. The problem is presented mathematically in the next section.

3 General resolution strategy

3.1 Optimization Problem Statement

In this section we will focus on understanding our optimization problem behind the image restoration. Firstly, we consider a 3D microscopic volume $\in \mathbb{R}^N$ of size $N = N_X \times N_Y \times N_Z$ given a degraded observation $y \in \mathbb{R}^N$. We here consider a linear model for which the degradation from x to y is associated to depth-variant blur operator $H \in \mathbb{R}^{N \times N}$. Considering $b \in \mathbb{R}^N$ the uncertainty on the model, the inverse problem we consider is the following

$$y = Hx + b \tag{1}$$

In order to solve (1), the most instinctive way to proceed is to find x as the minimizer of quantity $||y - Hx||$. However, such a strategy almost always remains irrelevant to the extent the solution of this latter problem is not necessary unique. Without any additional information on the image to reconstruct, there is thus no guarantee for an algorithm to give the "good" solution. In order to direct the research towards the desired one, we also make additional assumptions about the initial image, such as minimal and maximal values for each pixel, or smooth regions and sharp edges in the image. In the field of unconstrained optimization, this amounts to adding a so-call penalisation term. Therefore, the cost function to minimize is of the form of

$$f : x \in \mathbb{R}^N \mapsto \frac{1}{2}\|y - Hx\|^2 + R(x). \tag{2}$$

where R is a prior function that helps to smoothness the intensity (locally the intensity doesn't change rapidly) defined in our situation as

$$R : x \in \mathbb{R}^N \mapsto \eta d^2_{[x_{min}, x_{max}]^N}(x) + \lambda \sum_{i=1}^{N} \sqrt{([V^X x]_n)^2 + ([V^Y x]_n)^2 + \delta^2} + \kappa \|V^Z x\|^2. \tag{3}$$

The regularization parameters $\lambda, \eta, \kappa, \delta > 0$ will be chosen manually to maximize the Signal to Noise Ratio (SNR). $V^X, V^Y, V^Z \in R^{N \times N}$ represent a gradient operators along every direction. In an image processing context, they are frequently used to promote images with relatively high contrasts (typically the binary ones). x_{min}, x_{max} the bounds of the intensity and $d^2_{[x_{min}, x_{max}]^N}$ the square of Euclidean distance to $[x_{min}, x_{max}]^N$.

3.2 Majorization Minimization (MM) Principle

Our general approach to solve problem (2) as a unconstrained and differentiable problem, is those based on so call *Majorization Minimization* (MM) principle. The latter relies on the existence of a function $h : \mathbb{R}^N \times \mathbb{R}^N \mapsto \mathbb{R}$ associated to f which verifies the two following properties.

$$\forall x, y \in \mathbb{R}^N \begin{cases} f(y) = h(y, y) \\ f(x) \leqslant h(x, y) \end{cases} \tag{4}$$

For a given $y \in \mathbb{R}^N$, we thus impose the graph of $h(., y)$ being over of those of f with a tangent property at point y. Starting at $x_0 \in \mathbb{R}^N$, then, at any $n \in \mathbb{N}$, we recursively find x_{n+1} as

$$x_{n+1} \in \underset{x \in \mathbb{R}^N}{\arg\min}\, h(x, x_n). \tag{5}$$

To the extent that h verifies 4, the sequence $(f(x_n))_{n\in\mathbb{N}}$ is always decreasing. More generally, the interest of the MM scheme notably relies on the fact it promotes a certain robustness through some theoritcal guarantees in either convex non-convex settings [3, 4]. Figure 1 gives an graphic explanation of the process.

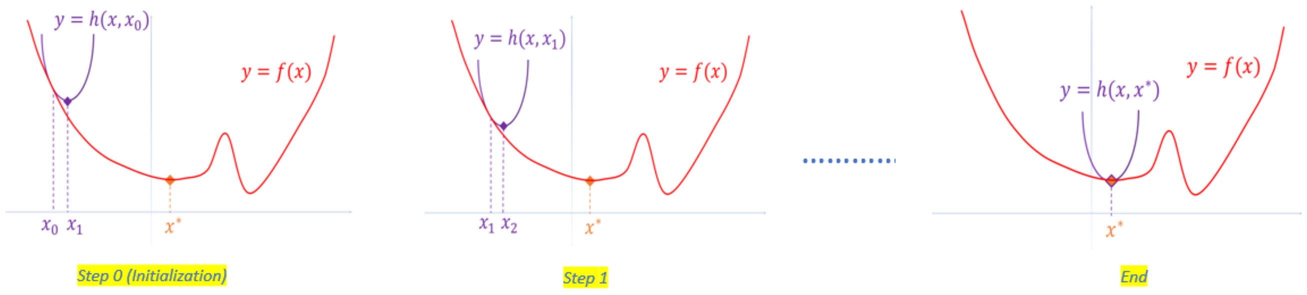


Figure 1: Illustration of the MM principle for a non-necessary convex f.

Most of the time we try to build h in a way of having very easy curvature property. A particularly privileged choice is notably those of having h with a quadratic structure. Such a construction is possible in view of function f considered here (see (2)). It follows that h verifying (4) is the form of

$$h : (x, y) \in \mathbb{R}^N \times \mathbb{R}^N = f(y) + \nabla f(y)^T (x - y) + \frac{1}{2}(x - y)^T A(y)(x - y), \tag{6}$$

with $A : y \in \mathbb{R}^N \mapsto A(y) \in \mathbb{R}^{N\times N}$ is an application returning a symmetric positive matrix. The strategy to build h as an interesting quadratic majorant function when f is the form of (2) is due to the previous works of [5, 1]. For the situation at stake, the matrix A is the following

$$A : y \in \mathbb{R}^N \mapsto H^\top H + \eta I_N + \lambda L^\top \operatorname*{Diag}_{1 \leqslant n \leqslant N} \left\{ \left[([V^X x]_n)^2 + ([V^Y x]_n)^2 + \delta^2 \right]^{-1/2} \right\} L + 2\kappa (V^Z)^\top V^Z. \tag{7}$$

where $L = \left[(V^X)^\top, (V^Y)^\top \right]^\top \in \mathbb{R}^{2N \times N}$ and $\mathrm{Diag}_{1 \leqslant i \leqslant N}\{.\}$ denotes the diagonal matrix of $\mathbb{R}^{N \times N}$ whose diagonal components are those inside the brackets.

3.3 Subspace acceleration : the 3MG scheme

The classic MM scheme (5) requires to solve a optmization sub-problem at every iteration. For the quadratic case and at a given $n \in \mathbb{N}$, such a procedure needs to inverse the matrix $A(x_n)$ which requires too much time and memory when N the dimension of the problem is relatively large . One common strategy to overcome this obstacle is based on a subspace acceleration method [3, 4] which consists in searching the new iterate x_{n+1} only along a small number of directions $d_1, ...d_{M_n}$ ($M_n << N$) instead of the whole space $\mathbb{R}^N$. If we denote $D_n = [d_1, ...d_{M_n}]$ the new sub-problem

$$x_{n+1} \in \operatorname*{argmin}_{x \in D_n} h(x, x_n) \tag{8}$$

is to inverse matrix $D_n^\top A(x_n) D_n \in \mathbb{R}^{M_n \times M_n}$ and is thus far less demanding in term of computation time. For the *Majorization Minimization Memory Gradient* (3MG) approach, we only adopt two directions ; the gradient of the current iterate as a way to keep a descent condition on the process and the difference of the two last past iterates $x_n - x_{n-1}$ so as to have $D_n = [\nabla F(x_n), x_n - x_{n-1}]$.

3.4 Combining subspace strategy with a block approach : BP3MG scheme

We now turn ourselves to an improvement over the 3MG scheme. Although accelerating the MM process using the subspace procedure is highly recommended, it is in general not sufficient to deal with very high dimension problem. Instead of focusing on the update of the entire vector x_n at every iteration $n \in \mathbb{N}$, another solution based on what it is done in parallel calculus relies on an alternative update by group of coordinates fixed in advance. Due to the new notations it induces, the resulting algorithm names BD3MG is described with more details in [2]

To reduce time of finding the minimizer, *Block Parallel Majorize-Minimize Memory Gradient* thus could be a way to fix this problem. BP3MG is a block coordinated algorithm by which we use a number of workers(cores) that, at each iteration, minimize a small portion of our image and share the result with each others. This is being done until we meet the criterion condition.

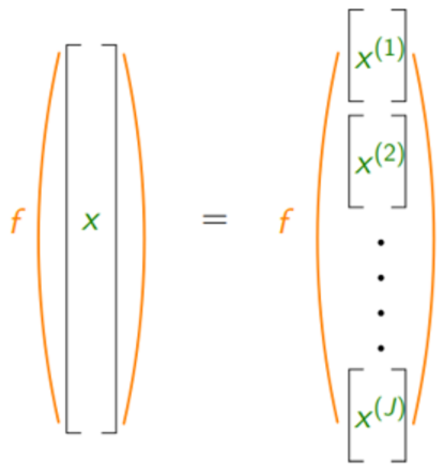


Figure 2: General approach of BP3MG scheme. $[\![1, N]\!]$ is upstream divided into J block of coordinates. At each iteration $n \in \mathbb{N}$ the 3MG procedure is only applied on one of the $x^(j)$ instead of the whole x_n. The choice of the updated block at every iteration is generally made following a cyclic or quasi-cyclic rule.

3.5 BD3MG algorithm

An alternative approach to BP3MG is the so-called *Block Distributed Majorize-Minimize Memory Gradient* (BD3MG) algorithm. Described in [7], this algorithm is based on updating smaller subspaces on different cores, **without waiting for others to have finished**. This allows faster execution than its counterpart BP3MG, but increases the convergence delay. To implement this algorithm, we need a master node and multiple workers, all executing on distinct cores. While a stopping criterion is not met, the master chooses a block subspace, sends it to a waiting worker. The worker proceed to a 3MG procedure and then returns the update to the master node. The master node then updates the global solution, and selects another block of coordinates to send back to the worker. The proof of the convergence of this algorithm can be achieved with a few properties on f, on the cyclical nature of block selection, as well as on the positive definite matrices [7], but is beyond the scope of this study.

4 Main code

4.1 Presentation

Matthieu Chavidal's GitHub contains all the necessary block functions, and a test script. We therefore need to create a main script from these functions. His test script starts from an image, then blurs it and noises it and restore it. This method permits to compare the three images : the original one, the degraded one and the restored one.
Although this is interesting to understand how the algorithm works, this is not what we needed. We wanted a script which starts from any image and restores it directly.
We coded this script and we will present the architecture in this section.
The code is constructed as :

1. Imports (some useful librairies and some functions from the files).
2. Loading the data : the image and the kernel h.
3. Prepare the data : calculating convolutions and adjoints which will be the inputs with Github's functions.
4. Set the parameters.
5. Call the restoration function.
6. Export the outputs.

4.2 Results

We had some good results with this code. In fact, the code could have many applications and could be very useful, as example in biology. Emilie Chouzenoux sent to us a 3D matrix which corresponds to a striated skeletal muscle unsliced and the kernel which blurred it. Because it is in 3D, to see it, we just had to slice it and view it as a picture. Here is a slice of the middle of the volume (left figure 3).

We can hardly see anything in this picture except some random pixels on the right. It seems very noisy and chaotic, but it is not. In fact, this is just a very degraded picture, not a chaotic one. We restored the 3D volume with our script and here is the result for the same slice:

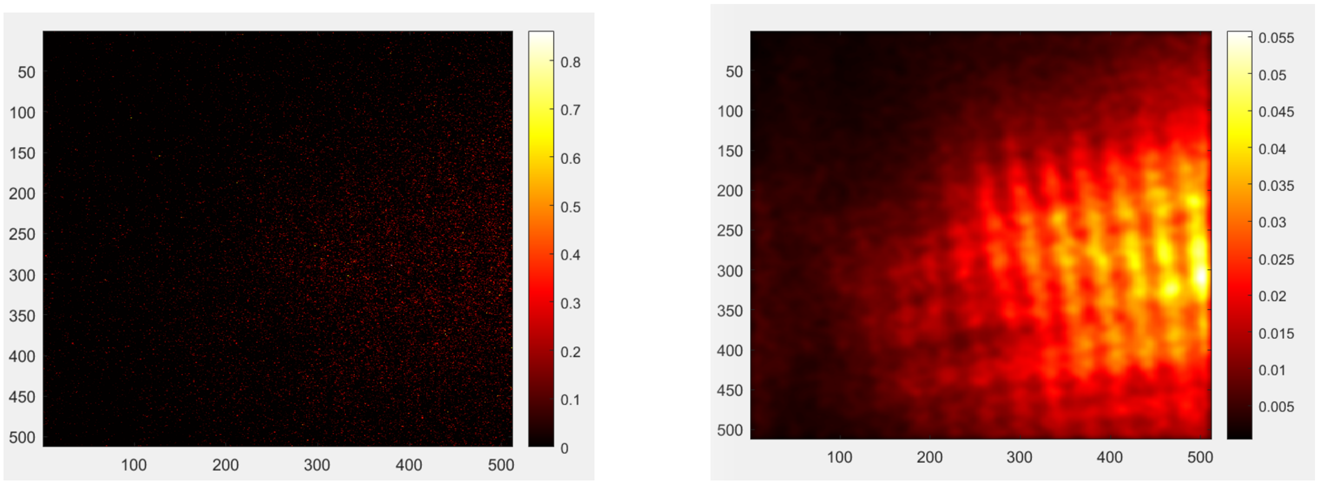


Figure 3: One example of reconstruction (right) starting from a blurred image (left) Output image Algorithm : BD3MG. Parameters : 40 cores, Nz=20, max_iterations=10.

The results are very noticeable. While we could hardly see anything on the first image, we can clearly see a shape on the second one. Note that 10 iterations is very low, but it takes only about 10s to run it with 40 cores and the result is quite impressive. We will see this in 3D and other results in part 9.

5 Computation example with BD3MG and BP3MG

In this part, we will restore a volume of data with BP3MG and BD3MG and compare the results and the outputs.

5.1 Presentation

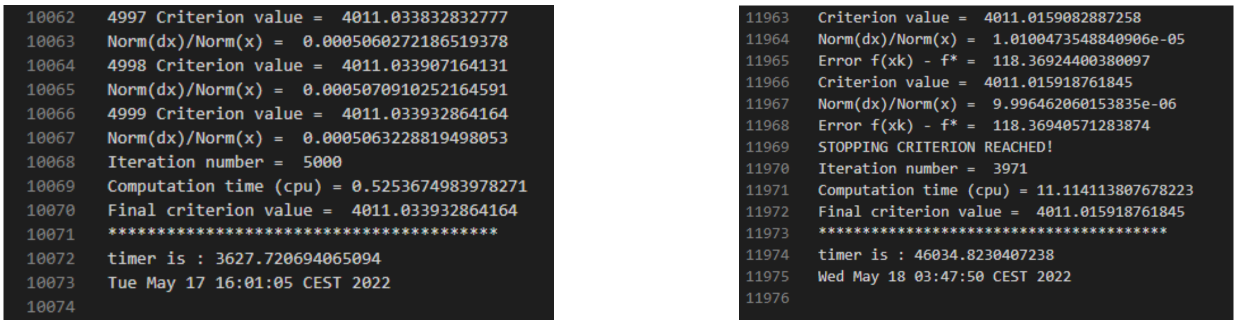
The data used is the same as before. The input matrix has a size of 512x512x100. H matrix has a size of 21x21x21.

We used our main code of BD3MG and BP3MG for the computation. We set the maximal number of iterations at 5000 and a maximal time of 24hours for both algorithm. Then we started the computation.

5.2 Output logs

Output file is very interesting to compare how algorithms work.

First, let's see the last few lines of the ouputs files.



```
4997 Criterion value =  4011.033832832777
Norm(dx)/Norm(x) =  0.0005060272186519378
4998 Criterion value =  4011.033907164131
Norm(dx)/Norm(x) =  0.0005070910252164591
4999 Criterion value =  4011.033932864164
Norm(dx)/Norm(x) =  0.0005063228819498053
Iteration number =  5000
Computation time (cpu) = 0.5253674983978271
Final criterion value =  4011.033932864164
****************************************
timer is : 3627.720694065094
Tue May 17 16:01:05 CEST 2022

```

```
Criterion value =  4011.0159082887258
Norm(dx)/Norm(x) =  1.0100473548840906e-05
Error f(xk) - f* =  118.36924400380097
Criterion value =  4011.015918761845
Norm(dx)/Norm(x) =  9.996462060153835e-06
Error f(xk) - f* =  118.36940571283874
STOPPING CRITERION REACHED!
Iteration number =  3971
Computation time (cpu) = 11.114113807678223
Final criterion value =  4011.015918761845
**************************************
timer is : 46034.8230407238
Wed May 18 03:47:50 CEST 2022

```

Figure 4: Output files structure : BD3MG on the left and BP3MG on the right.

As we can see, BD3MG took about one hour to make 5000 iterations. On the contrary, BP3MG took slightly less than 13 hours and stopped at 3971 iterations thanks to the stopping criterion. BD3MG is very fast compared to BP3MG.

Moreover, we can see that the final criterion value is the same for both algorithms. This means that they give the same result, which appears to be in this case the optimal result.

We cannot see it in the last few lines of the output file but while BP3MG always makes decrease the criterion, sometimes BD3MG makes it increase, due to local minimums.

5.3 Results

Let's review graphically the results of this computation :

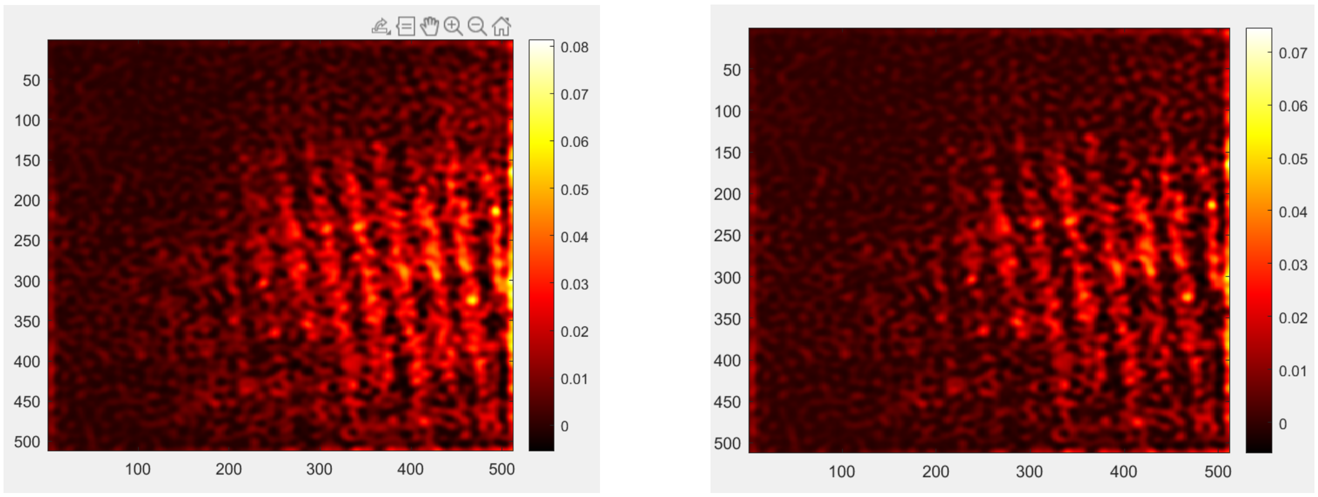


Figure 5: Muscle fibers reconstitution using BD3MG on the left and BP3MG on the right.

As you can see, results are quite the same but not exactly although the criterion was the same at the end of the execution. We will see this restoration in 3D and some others in part 8.

6 Speedup

Initially, the speedup was supposed to be a great part of this project. We will explain in this part what is wrong with that and what are our suggestions.

We have done many test on the speedup since the beginning of the project. We fastly figured many issues but the first thing to do was well understanding the problem and creating a main script. After a few months working on other things, we were back working on the speedup.

Therefore, a great part of our job has been to see how the restoration algorithm is supposed to work to understand the issues.

6.1 How the algorithm is supposed to work

In this subsection we will focus on the two main algorithms BD3MG and BP3MG.

BD3MG

In the script, we set a number of cpu and a Nz number which corresponds to the third dimension of the volume.

In the case of $Nb\ core < Nz$, the volume is sliced in $\dfrac{Nz}{Nb\ cores}$ and rounded if necessary. Thereafter, each core is allocated to its slice and the restoration begins.

According to Mathieu's paper, in the case of $Nb\ core > Nz$, the extra cores are supposed to work on others layers, with the following rule :
-CPU i works on the layer i % Nz

BP3MG

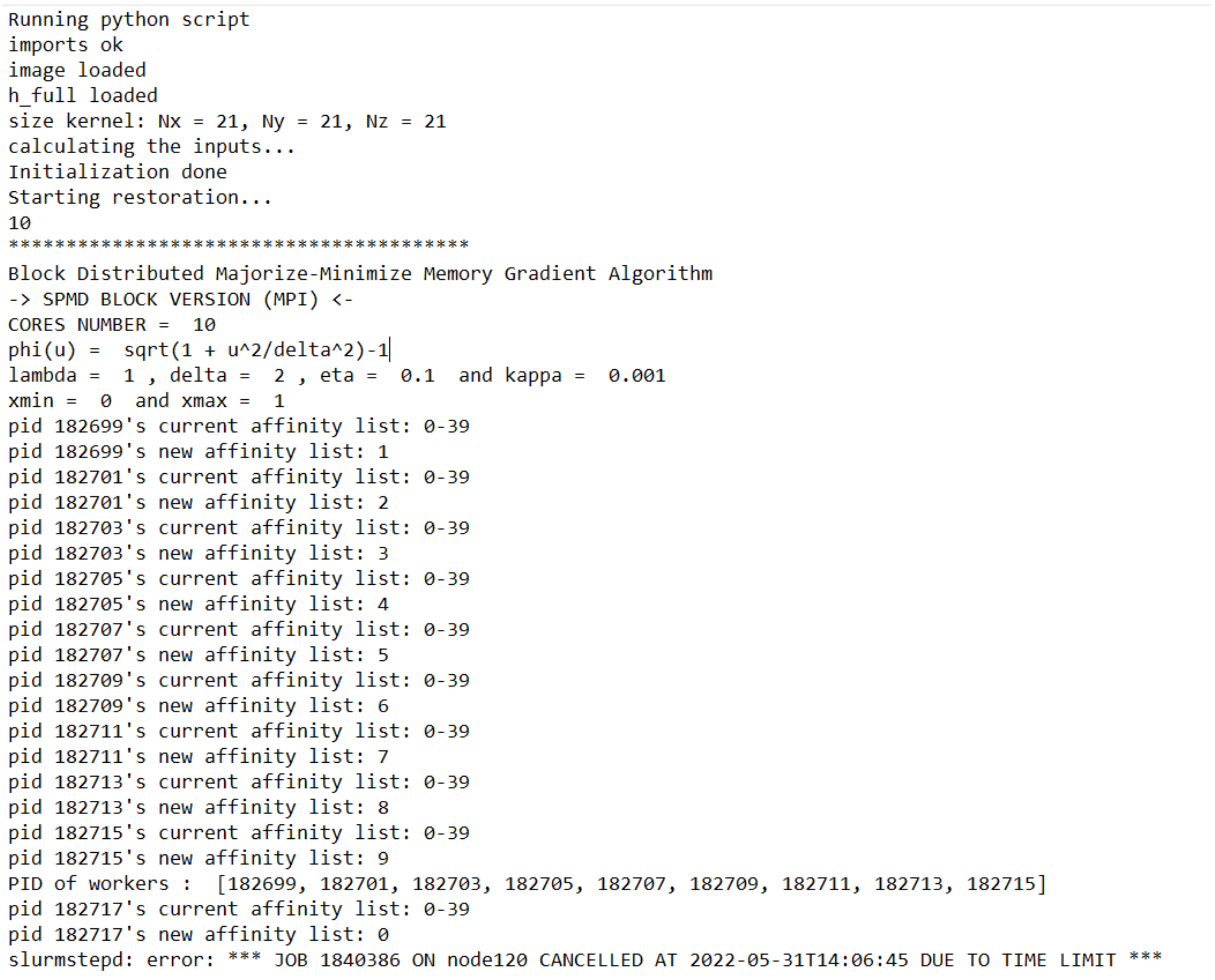
Speedup cannot be computed as usual for the BP3MG algorithm because the algorithm only works when $Nb\ cores \% Nz = 0$, ie for Nb cores=Nz, Nbcores=2Nz, etc... We will then run the script for these particular parameters.

6.2 How it works

BD3MG

In the case of $Nb\ core < Nz$, the algorithm never finishes. We have spent many hours on that, but we never figured out how to fix it. In fact, there is not as many logs as necessary to simply fix the issue. Here is the output log for the following parameters :

Nz=40 | cores number = 10 | max_iterations=500 | time_limit=48h | restoration of the matrix ImageSmall



```
Running python script
imports ok
image loaded
h_full loaded
size kernel: Nx = 21, Ny = 21, Nz = 21
calculating the inputs...
Initialization done
Starting restoration...
10
****************************************
Block Distributed Majorize-Minimize Memory Gradient Algorithm
-> SPMD BLOCK VERSION (MPI) <-
CORES NUMBER =  10
phi(u) =  sqrt(1 + u^2/delta^2)-1
lambda =  1 , delta =  2 , eta =  0.1  and kappa =  0.001
xmin =  0  and xmax =  1
pid 182699's current affinity list: 0-39
pid 182699's new affinity list: 1
pid 182701's current affinity list: 0-39
pid 182701's new affinity list: 2
pid 182703's current affinity list: 0-39
pid 182703's new affinity list: 3
pid 182705's current affinity list: 0-39
pid 182705's new affinity list: 4
pid 182707's current affinity list: 0-39
pid 182707's new affinity list: 5
pid 182709's current affinity list: 0-39
pid 182709's new affinity list: 6
pid 182711's current affinity list: 0-39
pid 182711's new affinity list: 7
pid 182713's current affinity list: 0-39
pid 182713's new affinity list: 8
pid 182715's current affinity list: 0-39
pid 182715's new affinity list: 9
PID of workers :  [182699, 182701, 182703, 182705, 182707, 182709, 182711, 182713, 182715]
pid 182717's current affinity list: 0-39
pid 182717's new affinity list: 0
slurmstepd: error: *** JOB 1840386 ON node120 CANCELLED AT 2022-05-31T14:06:45 DUE TO TIME LIMIT ***
```

As we can see on the log output, all the imports are made, there is not problem, pids are allocated. Between the penultimate line and the last one, the restoration is supposed to be made. When it is done, all the iterations values are displayed in one time before displaying the

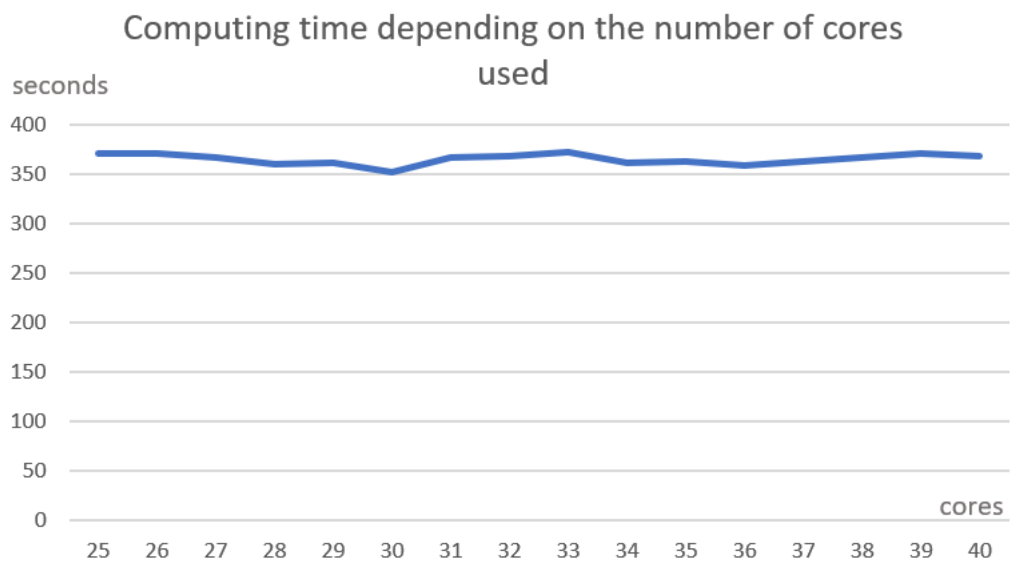
result. Therefore, fact that there is no iteration result allows us to say that the computation never finishes.
How many iterations are done ? We don't know because all the results are displayed at the end of the computation. It could be the first iteration who crashes, or the second, or the last etc... As we can see on the last line, the job is cancelled due to time limit, after 48h of computation. We don't think that this is simply due to a too long and complex computation because a restoration with 500 iterations with Nz = Nb cores takes roughly 10 minutes. This is also not a memory issue because if there is one, the job is automatically cancelled and it would be displayed on the output log.

In the case of $Nb\ core > Nz$, the circularity is simply not implemented in the algorithm, or at least in the version we had. After investigating on the code, the first Nz cores are allocated with the following rule :
-Core i allocated to z=i for i<Nz -Core i is not allocated for i>Nz

At least, the algorithm restores well the volume. But there is no speedup, since the extra cores are not used.
For example, let's take as parameters :

Nz=20 | cores number = from 25 to 40 | max_iterations=500 | time_limit=4h | restoration of the matrix ImageSmall

For this experiment, we set a timer which will display the computation time on the log output. Here is the timer result :



As we can see, the calculation time is constant. Our explanation could confirm these results.

BP3MG For the BP3MG algorithm, we tried to run the script for some specific parameters. First computation

Nz=20 | cores number = 20 | max_iterations=500 | time_limit=4h | restoration of the matrix ImageSmall

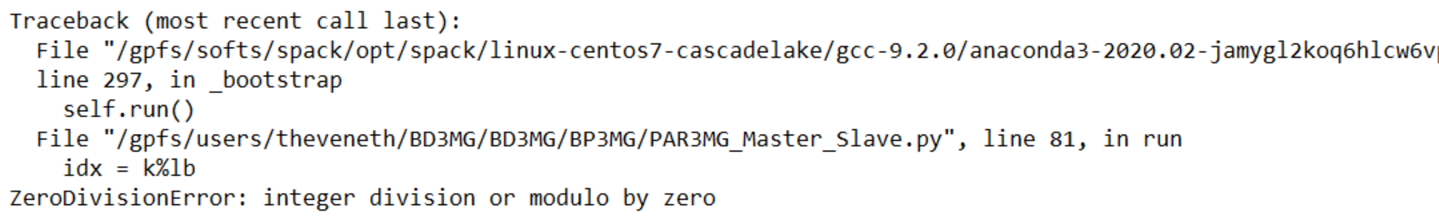
This computation works well, there is no need to see the output. We used Nz=cores number every time we had to use the BP3MG algorithm.

Second computation :

Nz=20 | cores number = 40 | max_iterations=500 | time_limit=4h | restoration of the matrix ImageSmall

This computation crashes. It is the case for every k such as $Core\_number = k * Nz$. Altough it was supposed to work. Let's review the error on the output log:

There is an error on the core of the algorithm. We tried to fix that, but it is not as simple as that because many files are involved in this error. This is the same error as if we don't respect the rule $Core\_number = k * Nz$ and try to compute something.
Therefore, to use the BP3MG in the project, we always used the rule $Core\_number = Nz$.



```
Traceback (most recent call last):
  File "/gpfs/softs/spack/opt/spack/linux-centos7-cascadelake/gcc-9.2.0/anaconda3-2020.02-jamygl2koq6hlcw6v
  line 297, in _bootstrap
    self.run()
  File "/gpfs/users/theveneth/BD3MG/BD3MG/BP3MG/PAR3MG_Master_Slave.py", line 81, in run
    idx = k%lb
ZeroDivisionError: integer division or modulo by zero
```

6.3 Conclusion

We struggled so much with this part. It was supposed to be well implemented and we were supposed to do experiments with the speedup. This was not the case and we were not able to debug it. This is because we did not have time to learn everything in parallel computation and object-oriented programming. We think debugging the code is not suitable for 2nd year students and this debugging has to be made by someone who is specialized in parallel computing and MPI.

7 Some results

In this part we will present some result we obtained will the algorithms for several input matrix. All input matrix are 3D, because we are restoring volumes. All output matrix are therefore 3D and we will show here slices of these volumes.

7.1 Flybrain

Here is the upper part of a fly brain.
Note that this is also the test restoration in the Github, so easily reproductible.

7.2 ImageSmall

Here is a small portion of muscle, obtained in 3D and restored.
Parameters are : 40 cores, 5000 iterations.

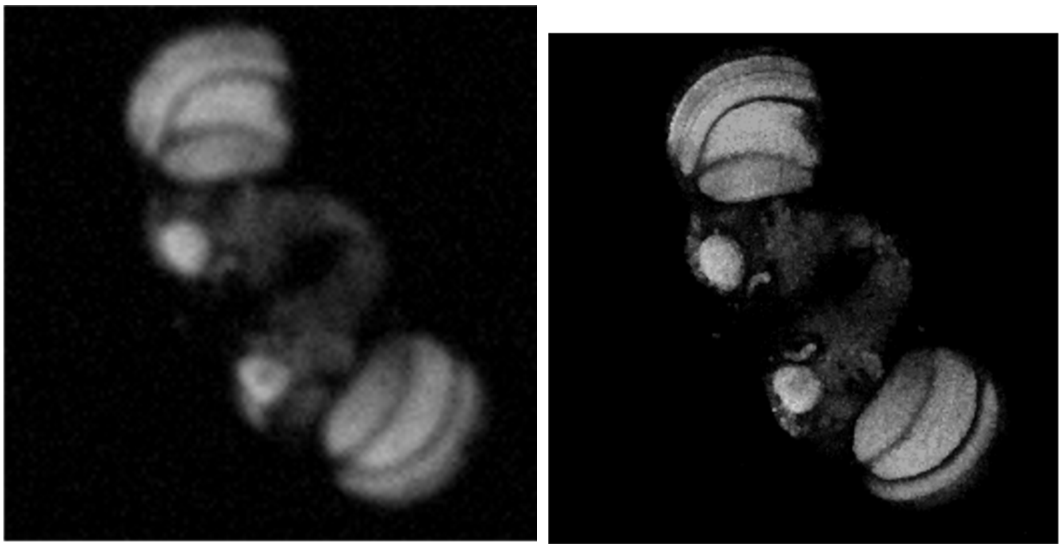


Figure 6: On the right : noised and blurred slice of the volume. On the left : a slice of the restored volume

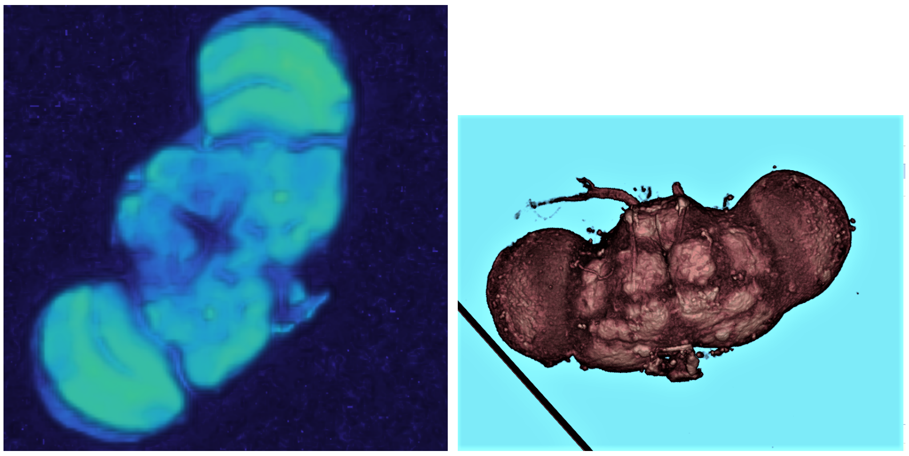


Figure 7: 3D view from above of the restored matrix on the left, and the same matrix with no noise, no blur on the right.

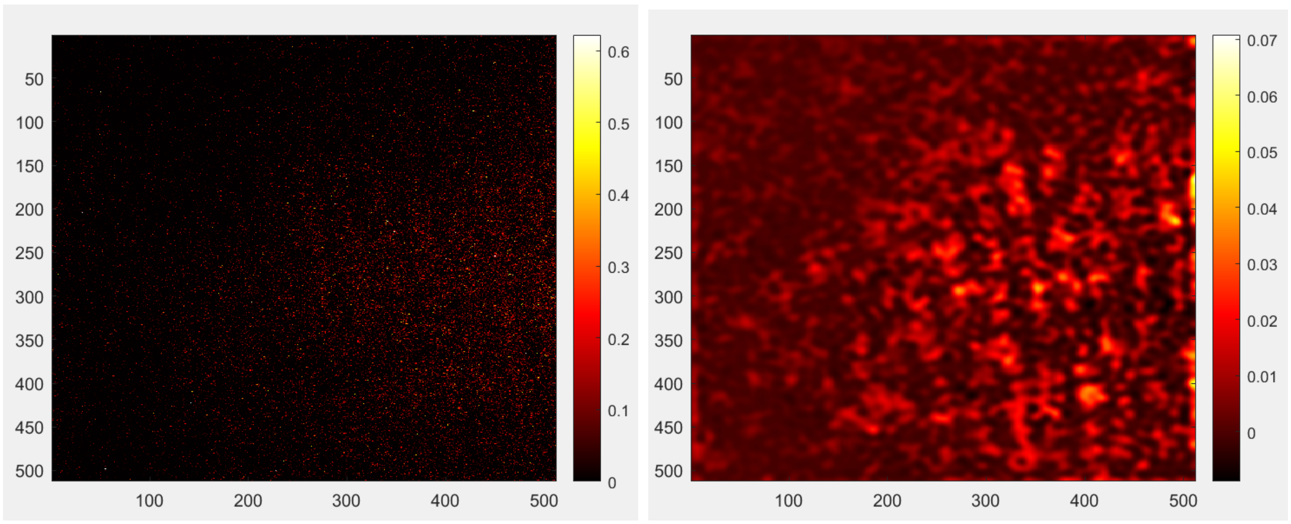


Figure 8: On the right : noised and blurred slice of the volume. On the left : a slice of the restored volume. Data from [6].

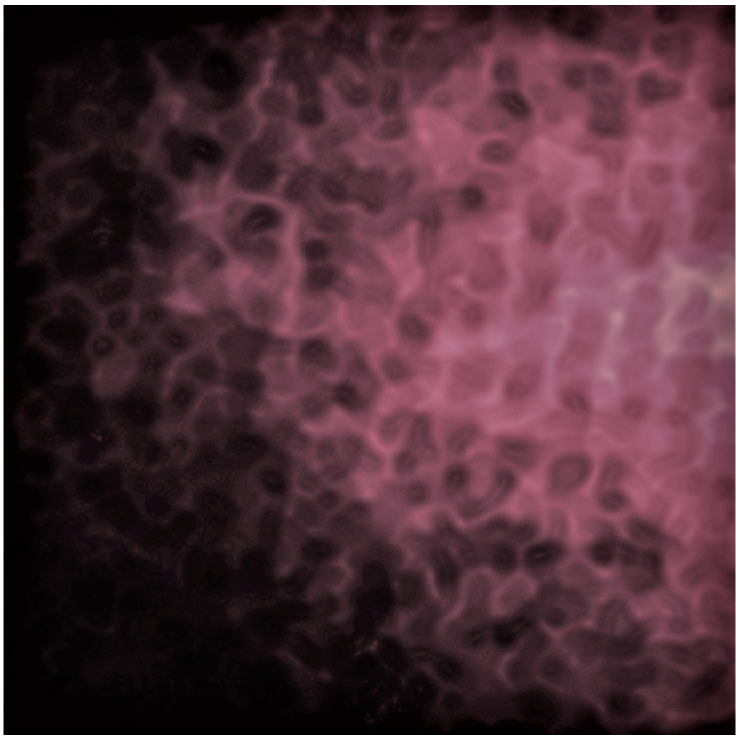


Figure 9: 3D view from above of the same restored matrix.

7.3 Aneurysm

An aneurysm is an outward bulging, likened to a bubble or balloon, caused by a localized, abnormal, weak spot on a blood vessel wall.

Parameters : 20 cores, 100 iterations.

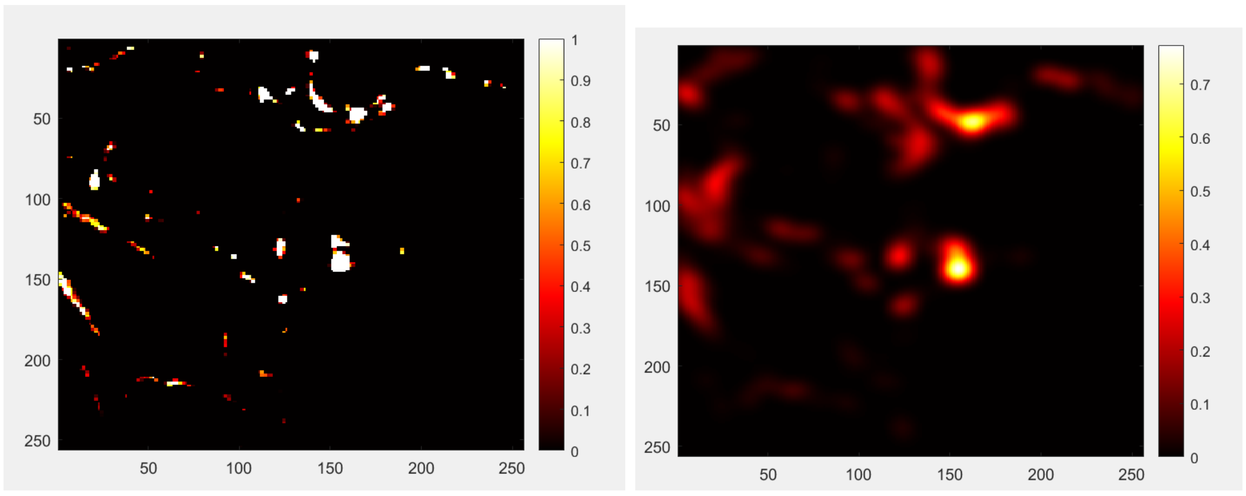


Figure 10: On the right : noised and blurred slice of the volume. On the left : a slice of the restored volume

8 Remarks

In this part we will focus on some interesting remarks that can help to continue the project.

Miscellaneous

We noticed some points during the project that are not obvious.

-Number of cores must be superior than the slice size of the volume restored. See the part about scalability for more details.

-Trying to restore a too big slice of the volume will create a memory problem on the supercomputer.

-If the volume is not entirely restored, then the number of iterations must be increased. For example, 10 iterations will only restore 9 slices of the volume.

-Be careful with the test image : the noise created induce some negative pixels which must be set to 0.

-To use the test image FlyBrain : set y[:,Nz] instead of y[:,:,Nz] in line 8 in BD3MG/BD3MG/BP3MG/Com

9 Conclusion

In this project, we studied large scale image restoration, aiming to apply optimization principles to high-dimensional spaces, and to compute restored blurred/noised images. The results for Block Distributed Majorization Minimization Memory Gradient are encouraging, allowing restoration of large images in a reasonable amount of iterations. The importance of parallelization was highlighted through our use of a supercomputer, that allowed us to run our code on multiple cores.

References

[1] G. Bouchard. Efficient bounds for the softmax function, applications to inference in hybrid models. In Presentation at the Workshop for Approximate Bayesian Inference in Continuous/Hybrid Systems at NIPS-07. Citeseer, 2007.

[2] S. Cadoni, E. Chouzenoux, J.-C. Pesquet, and C. Chaux. A block parallel majorize-minimize memory gradient algorithm. In 2016 IEEE International Conference on Image Processing

(ICIP), pages 3194–3198. IEEE, 2016.

[3] E. Chouzenoux, J. Idier, and S. Moussaoui. A majorize–minimize strategy for subspace optimization applied to image restoration. IEEE Transactions on Image Processing, 20(6):1517–1528, 2010.

[4] E. Chouzenoux, A. Jezierska, J.-C. Pesquet, and H. Talbot. A majorize-minimize subspace approach for \ell_2-\ell_0 image regularization. SIAM Journal on Imaging Sciences, 6(1):563–591, 2013.

[5] D. Geman and C. Yang. Nonlinear image recovery with half-quadratic regularization. IEEE transactions on Image Processing, 4(7):932–946, 1995.

[6] C. Lefort. Ad fast instrumental and computational pipeline for multiphoton microscopy.

[7] E. C. Mathieu Chalvidal. Block distributed 3mg algorithm and its application to 3d image restoration. 2018.

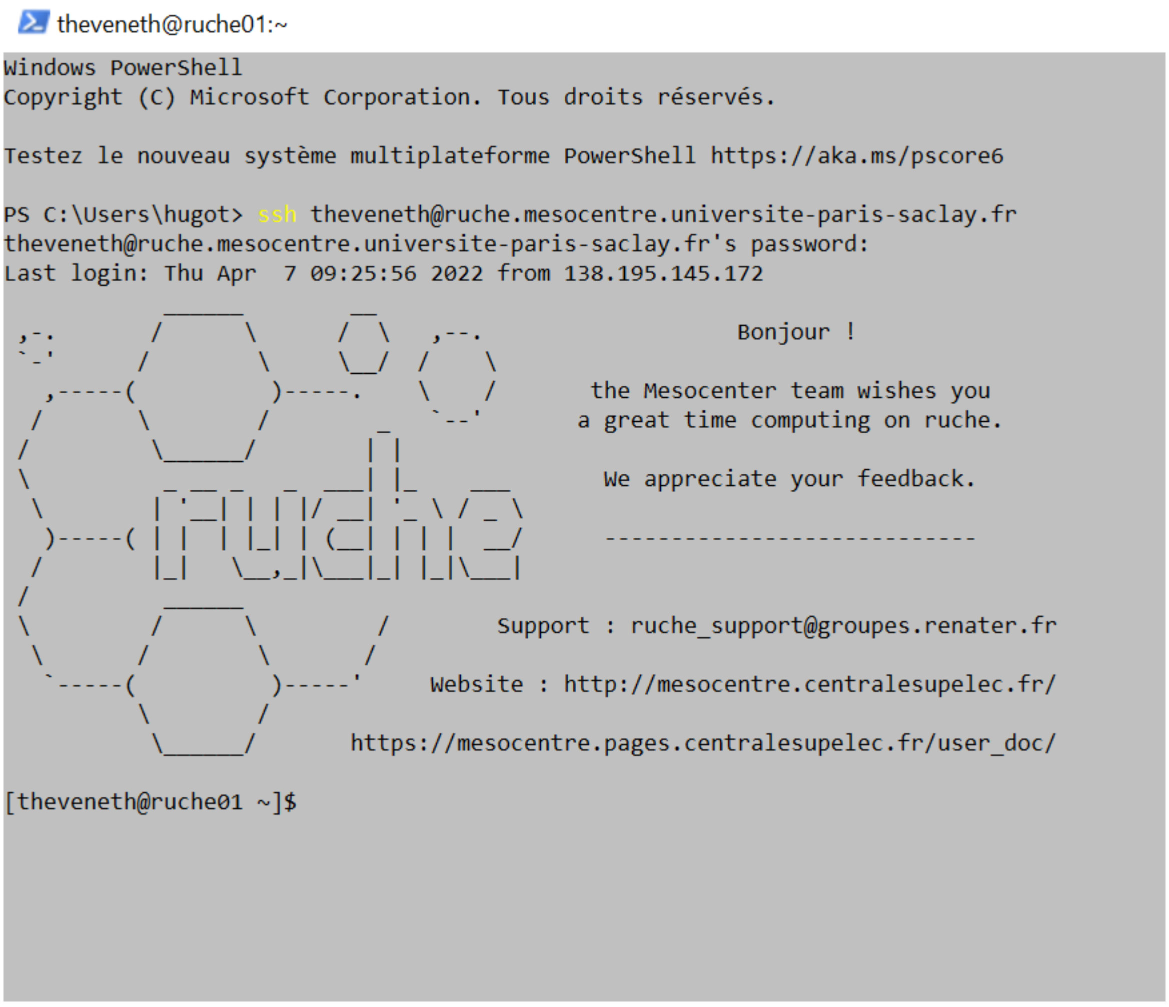
10 Appendix : Supercomputer

10.1 Introduction

For this project we used the Ruche supercomputer at the Paris-Saclay Mésocentre. Reserved for researchers and members of the university justifying the need, this supercomputer proposes to use up to 2 nodes and 40 cpu per node for each user.
The Mésocentre offers training to properly use the supercomputer.
We had never used a supercomputer before and had to do it on our own, so we now propose to explain in broad outline how the Mésocentre works and how to run a code with it.

10.2 Connecting

To connect easily, you can use any ssh client. On Windows, the most powerful installed in every computer is the PowerShell. Here are the commands to connect to Ruche. Note that you need an account to connect to it.



```
theveneth@ruche01:~
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\hugot> ssh theveneth@ruche.mesocentre.universite-paris-saclay.fr
theveneth@ruche.mesocentre.universite-paris-saclay.fr's password:
Last login: Thu Apr  7 09:25:56 2022 from 138.195.145.172
           ______       __
 ,-.      /      \     /  \   ,--.                  Bonjour !
 `-'     /        \    \__/  /    \
   ,-----(          )-----.  \    /        the Mesocenter team wishes you
  /      \        /        _  `--'       a great time computing on ruche.
 /        \______/        | |
 \          _ __ _   _  ___| |__   ___        We appreciate your feedback.
  \        | '__| | | |/ __| '_ \ / _ \
   )-----( | |  | |_| | (__| | | |  __/     ----------------------------
  /        |_|   \__,_|\___|_| |_|\___|
 /
 \          ______         /        Support : ruche_support@groupes.renater.fr
  \        /      \       /
   `-----(          )-----'    Website : http://mesocentre.centralesupelec.fr/
          \        /
           \______/      https://mesocentre.pages.centralesupelec.fr/user_doc/

[theveneth@ruche01 ~]$
```

Unfortunately, this is very difficult to code from the PowerShell because you need to write linux commands to open the files, modify them and save them.
Thus, after a few weeks with the PowerShell, we decided to use the ssh remote client from VS code. This is an very useful extension who permitted us to work in good condition. Therefore, we had a normal interface and we were using a terminal in VS code to execute the files.

10.3 Batch

With a supercomputer, you cannot you execute the files. You need to write a batch script which lists what you want to execute, with what hardware resources... Then, send the batch script to the supercomputer. Your file is placed on a queue and executed when hardware resources are available. After running it, the supercomputer creates a .out file which shows the outputs of your script.

We struggled a lot with batch files. At the beginning of the project, we did not know how to use the supercomputer and we never made any batch command. We figured out we needed to use it, so we learnt by our own the main commands.

Here is a simple batch script we used, with comments :

```
::1st section : initialize the script
#!/bin/bash
#SBATCH --time=01:00:00
::set time limit
#SBATCH --job-name=JobCVN
:: set job name
#SBATCH --nodes=2
::number of used nodes (up to 2)
#SBATCH --ntasks=80
#SBATCH --cpus-per-task=1
::set 1 task per cpu, up to 40 cpu per node.

:: Load necessary modules for the imports
module purge
module load anaconda3/2020.02/gcc-9.2.0

:: Run python script
echo "Running python script"
python3 BD3MG/v1.py ::path to the script

date
::end of the file
```

Here are some simple commands to manage your jobs with the Mésocentre :

After coding you batch file, you simply have to execute it in the terminal with the code :

```
sbatch my_file.bat
```

Your file is sent to the queue. To see it, write :

```
squeue
```

There, you can see your jobs and their ids number. To cancel a job, write in the terminal :

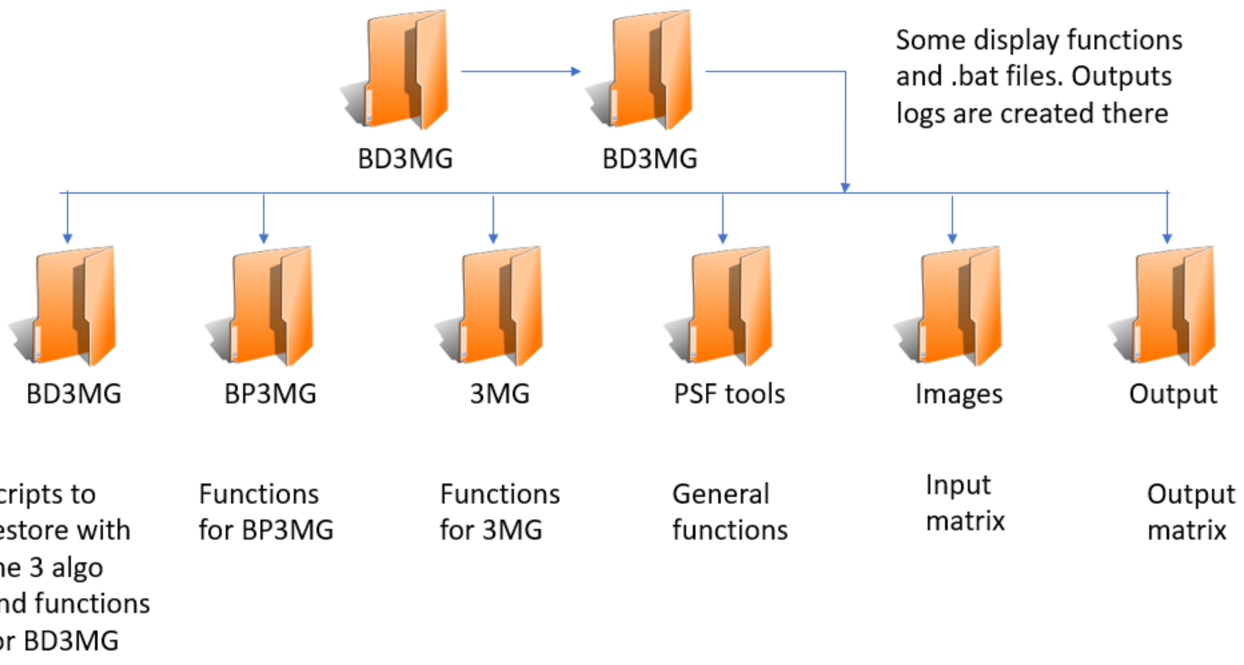
```
scancel jobid
```

To see the .out file, write in the terminal :

```
vim slurm-jobid.out
```

Now you can successfully run your code on Ruche !

11 Appendix : File management


Some display functions and .bat files. Outputs logs are created there
BD3MG
BD3MG
BD3MG
BP3MG
3MG
PSF tools
Images
Output
Scripts to restore with the 3 algo and functions for BD3MG
Functions for BP3MG
Functions for 3MG
General functions
Input matrix
Output matrix