Reference manual
Fiji plugin VSNR Version 2

Pierre Weiss

August 22, 2013

Contents

1 Disclaimer

We are academic researchers working mostly on mathematical issues. We developed this plugin since many researchers asked us to do so. Unfortunately, we have very limited knowledge of Java programming and Fiji plugins. We thus apologize for the numerous problems you will encounter with this plugin and ask for indulgence. In case you find important bugs, you can contact me at pierre.armand.weiss@gmail.com. If I find the problem is worth it, I will try to correct the bug as fast as my agenda will permit.

2 Acknowledgments

Even though I put my name on the front page of this document (since I wrote it), many people contributed to this work.
The plugin interface was developed mostly by Léo Mouly (an intern student of Toulouse University in computer science).
The DenoiseH1 function was developed by Omar Dounia (an intern student of Toulouse University in mathematical engineering).
The main methodological ideas were developed by Jérôme Fehrenbach and myself.
Many people at ITAV-USR3505 contributed to improve the plugin with their feedback including (but not limited to) Marie-Laure Boizeau, Kathleen Perrin, Valérie Lobjois and Corinne Lorenzo.

3 Introduction

This booklet is written to explain the main features of the 2nd version of the VSNR Fiji plugin. This plugin is an implementation of an idea presented in an academic journal paper [1]. In case you successfully apply this plugin for your academic research, please cite this work. This plugin also implements the (more theoretical) ideas detailed in [2].

3.1 What is this plugin all about?

This plugin allows to denoise images degraded with stationary noise. Stationary noise can be seen as a generalization of the standard white noise. Typical applications of this plugin are:

- Standard white noise denoising using a total variation and l^2 fidelity term minimization. Figure 1 is an example of this application. Even though total variation denoising is not the state of the art (regarding SNR improvement), it may be very valuable for further tasks such as image segmentation.
- Destriping (the problem that motivated us to develop these ideas). Figures 2, 3 and 4 are examples of this application.

- Deconvolution (even though most users won't be able to use this feature).
- Cartoon + texture decomposition which might be useful to compress images, analyse textures or simplify segmentation like tasks.

In this booklet we will mostly describe the first two applications. The plugin works both for 2D and 3D images, we did not implement multi-spectral features. You can find more informations, a Matlab implementation and examples of this algorithm on the following webpage: `http://www.math.univ-toulouse.fr/~weiss/PagePublications.html`.

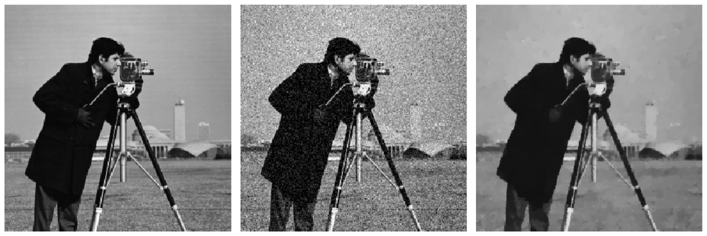


Figure 1: Left: Original image - Middle: noisy image (Gaussian noise, 11.3dB) - Right: restored image using this plugin (20.6dB).

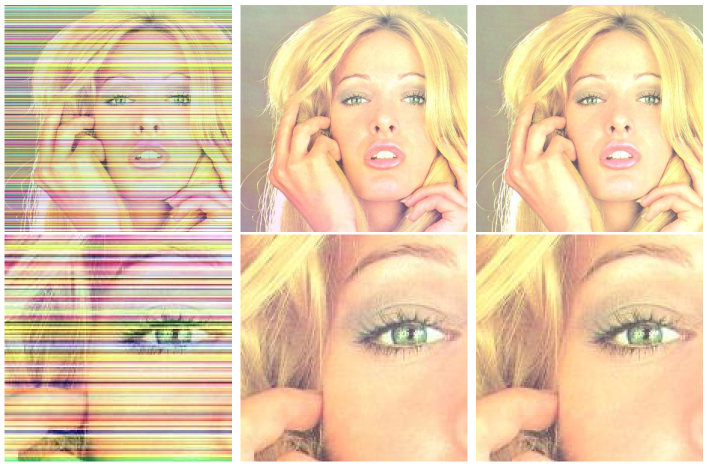


Figure 2: Top: full size images - Bottom: zoom on a small part. From left to right: Noisy image (16,5dB), denoised using the plugin (PSNR=32,3dB), original image.

3.2 What is VNSR version 1?

The first version of the plugin is not available on the web. It may be obtained upon demand.

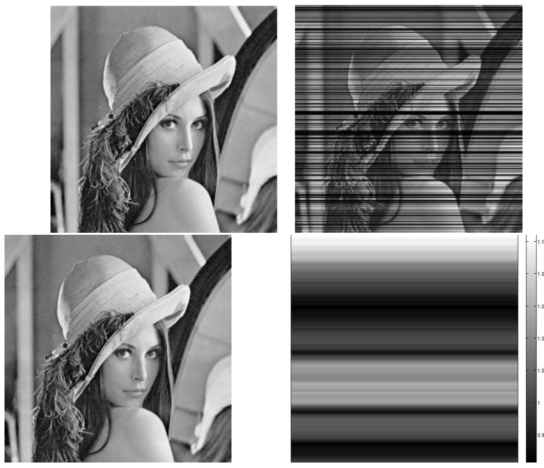


Figure 3: An example involving a multiplicative noise model. From left to right. Original image - Noisy image. It is obtained by multiplying each line of the original image by a uniform random variable in $[0.1, 1]$. SNR=10.6dB - Denoised using the plugin on the logarithm of the noisy image. SNR=29.1dB - Ratio between the original image and the denoised image. The multiplicative factor is retrieved accurately.

The core of the program was written in C code which implies that you may need to recompile it on your own computer to produce a .so or a .dll if you want to use it. Overall, we strongly recommend the use of the second version for most users since the installation is very simple and we added feature that strongly ease the plugin use.
Moreover the FFT implementation we use in the second version is the JTransform, which is a multi-theaded version developed by Piotr Wendykier `https://sites.google.com/site/piotrwendykier/software/jtransforms`. Since it is parallelized, it might be much faster on multi-core computers.
For advanced users, the first version allows to do fun stuff such as TV-l^1 (e.g. segmentation, pattern recognition) or TV-l^∞ (e.g. dequantization).
The main features of the different versions are summarized in Table 1.

	Version 1	Version 2
Denoising with an arbitrary number of filters using total variation	✓	✓
Possibility to set $p \neq 2$	✓	✗
$2D$ and $3D$ implementations	✗	✓
Simplified parameter selection	✗	✓
Possibility to use H^1 denoising	✗	✓
More robust optimization	✗	✓
Possibility to handle Huber regularization	✓	✗
More user friendly interface	✗	✓

Table 1: Differences between VSNR versions

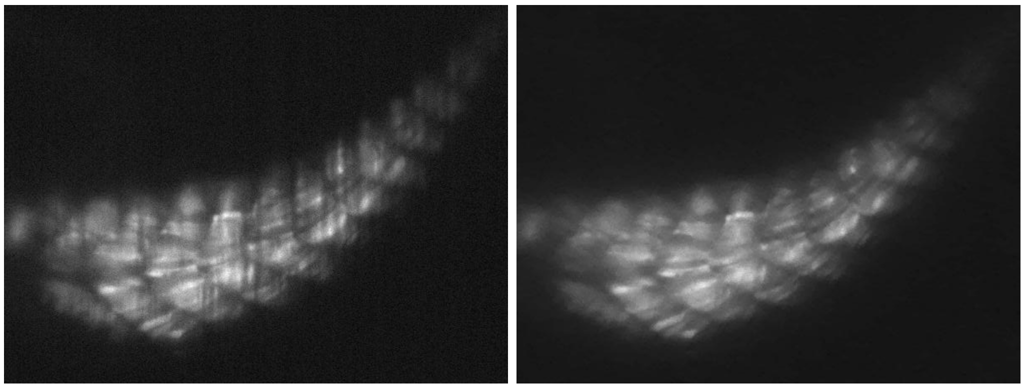


Figure 4: Left: SPIM image of a zebrafish embryo Tg.SMYH1:GFP Slow myosin Chain I specific fibers. Right: denoised image using VSNR. (Image credit: Julie Batut).

4 Installation

The implementation is very easy: just copy paste the file *VSNR2_plugin.jar* in your plugin directory. The plugin will appear in Plugins → Process → VSNR_V2.

5 How to use the plugin?

We tried to make the plugin accessible to a broad audience. However, if you have no idea of the theory behind the plugin you might fail to use it correctly. We thus recommend people to read this small guide before trying the plugin.

5.1 General principle

The general pipeline to denoise your image is the following:

a. Open your image.

b. Select the 2D or 3D modes.

c. Set one or more filters using the Dirac or Gabor buttons.

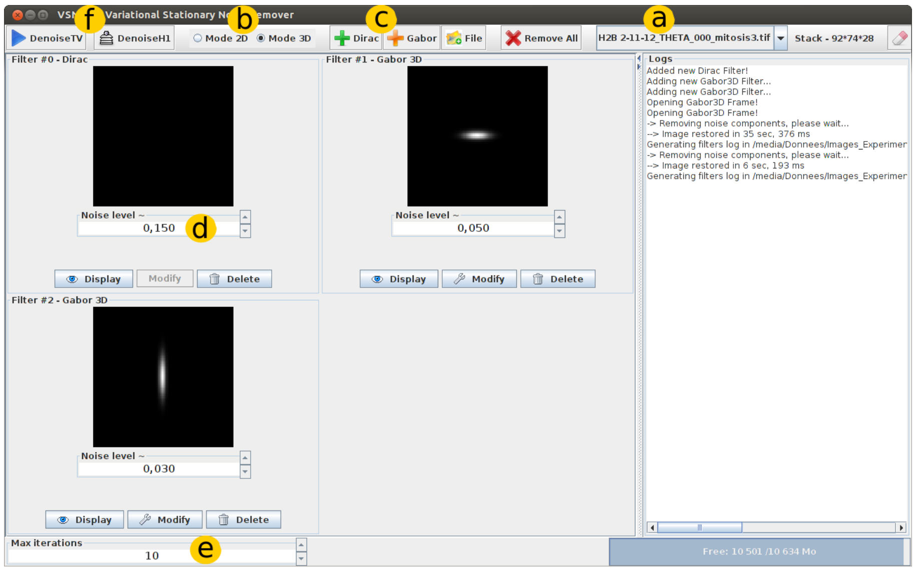
d. Set the noise level parameters.

e. Set the iteration number if you use DenoiseTV.

f. Launch DenoiseTV or DenoiseH1 to get the results.

g. The results will popup at the end of the process showing the denoising result as well as the noise component[1].

In the course of this setting, you will have to make choices about the 2D or 3D modes, the choice of a filter, the choice of the denoising parameters, the choice of DenoiseTV or DenoiseH1 as your algorithm. We detail the main ideas to make these choices below. Figure 5 show the plugin main interface together with the location of each step described above.



H2B 2-11-12_THETA_000_mitosis3.tif

Stack - 92*74*28

Filter #0 - Dirac
Noise level ~ 0,150

Filter #1 - Gabor 3D
Noise level ~ 0,050

Filter #2 - Gabor 3D
Noise level ~ 0,030

Max iterations 10

Logs

```
Added new Dirac Filter!
Adding new Gabor3D Filter...
Adding new Gabor3D Filter...
Opening Gabor3D Frame!
Opening Gabor3D Frame!
-> Removing noise components, please wait...
--> Image restored in 35 sec, 376 ms
Generating filters log in /media/Donnees/Images_Experimer
-> Removing noise components, please wait...
--> Image restored in 6 sec, 193 ms
Generating filters log in /media/Donnees/Images_Experimer
```

Free: 10 501 /10 634 Mo

Figure 5: .

5.2 2D and 3D modes

The plugin might handle 2D and 3D images. If you open a 2D image, then you will only be able to use the 2D mode. If you open a 3D image, you might choose between the 2D or 3D mode. The 2D mode will simply denoise each slice of your stack separately with a 2D filter. The 3D mode will denoise your 3D stack completely using a 3D total variation regularization.
Usually the 3D mode should be preferred and should produce better results. The 2D mode might be preferred in case of memory problems (it is much lighter) or for specific problems where different slices have little or nothing in common.

[1]Sorry, this is not recommended usage in Fiji, we discovered this when the plugin was near finished.

5.3 Setting up a filter

Perhaps the most important and difficult aspect of this plugin is the choice of a filter. In the current version, you can choose between 3 solutions: Dirac, Gabor and File. They should be chosen depending on your application. The general principle is to set a filter that "looks like" your noise.
From a mathematical point of view, our algorithm assumes that the noise that should be treated can be written as:

$$b = \sum_{i=1}^{n} \lambda_i \star \psi_i.$$

where $\star$ denotes the discrete convolution operator, λ_i is the realization of a white noise (ideally Gaussian) and ψ_i is a filter. The choice of the filter in the plugin corresponds to the choice of ψ_i.
Figure 6 illustrates instances of b for different choices of filters ψ and white noises λ.

5.3.1 Dirac

The standard white noise (might be Gaussian, Poisson, impulse, etc...) should be treated using a Dirac filter. From a statisical point of view, the method is only adapted to Gaussian white noise, but the method might still produce interesting results for other kinds of white noises. Figure 7 shows different kinds of white noises to clarify this notion (for people not used to the signal processing vocabulary).

5.3.2 Gabor

The Gabor filter is a function used extensively in signal processing. Some examples are given in Figure 8. The Gabor function is defined by 2 or 3 axes length (σ_x, σ_y) (depending whether, it is a 2D or 3D function), an orientation θ, a frequency λ (the Gabor function might oscillate more or less) and a Phase which may be used to shift the frequency axis ϕ.
The mathematical formula (not standard) we use in the plugin is the following:

$$g(x, y) = \exp\left(-\frac{x'^2}{2\sigma_x^2} - \frac{y'^2}{2\sigma_y^2}\right) \cos\left(\lambda \frac{x'}{\sigma_x} + \phi\right).$$

where $x' = x\cos(\theta) + y\sin(\theta)$ and $y' = -x\sin(\theta) + y\cos(\theta)$.
By setting $\lambda = 0$ and $\phi = 0$, the above function is nothing but an anisotropic Gaussian. The parameters σ_x and σ_y denote the Gaussian extent in the x and y directions. If you want to create a very elongated shape in the x direction for instance, it suffices to set $\sigma_x \gg \sigma_y$ and $\theta = 0$.
The parameter λ basically indicates the number of oscillations in the x direction. Setting $\lambda = 0$ will produce no oscillations, while setting $\lambda = 10$ will produce about 10 oscillations.

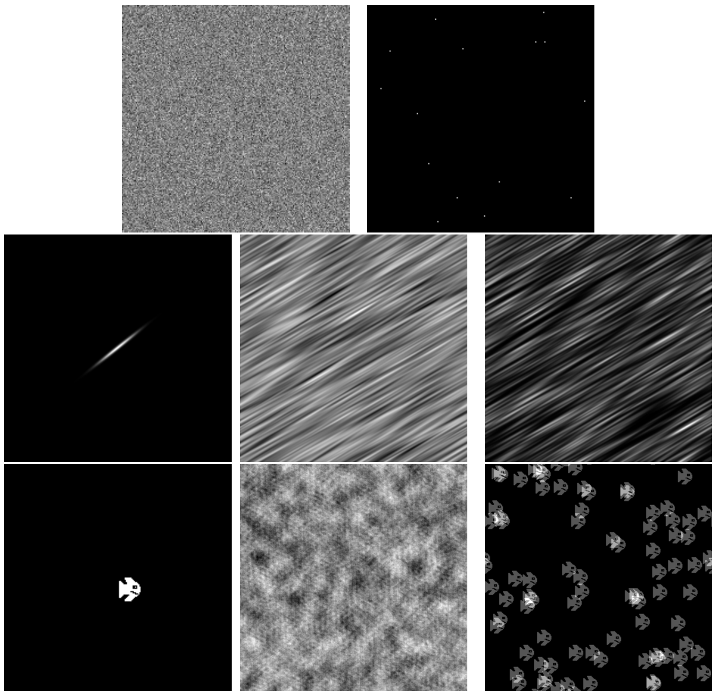


Figure 6: Different kinds of stationary noises. Top: white Gaussian noise and white impulse noise. Middle: anisotropic Gaussian filter ψ_1, stationary noise obtained by convolution of ψ_1 with the white Gaussian noise, stationary noise obtained by convolution of ψ_1 with the white impulse noise. Bottom: fish pattern filter ψ_2, stationary noise obtained by convolution of ψ_2 with the white Gaussian noise, stationary noise obtained by convolution of ψ_2 with the white impulse noise.

In all destripting applications we encountered, the phase and frequency were useless parameters that we set to 0. We thus decided to not implement them in 3D [2]. You can still use them in 2D.
Important note: in the destriping experiments we performed, we observed that it was usually beneficial to set Gabor filters with smaller axes than what they looked like. It seems it makes the algorithm more robust to e.g. the multiplicative nature of the noise.

5.3.3 File

We did not insist on implementing this feature correctly. It might work for 2D images, but it will definitely not for 3D images. The Gabor or Dirac functions provided with the plugin might not suit your application. In that case, you

[2]If your noise has a dominant frequency, it might be useful to use these parameters though.

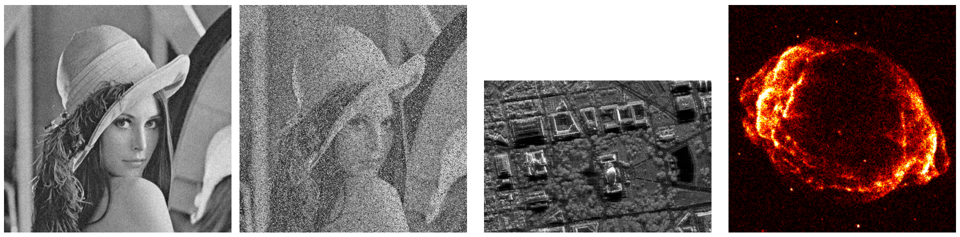


Figure 7: Different kinds of white noise. From left to right: Gaussian noise, impulse noise, multiplicative in SAR, Poisson noise in astronomy.

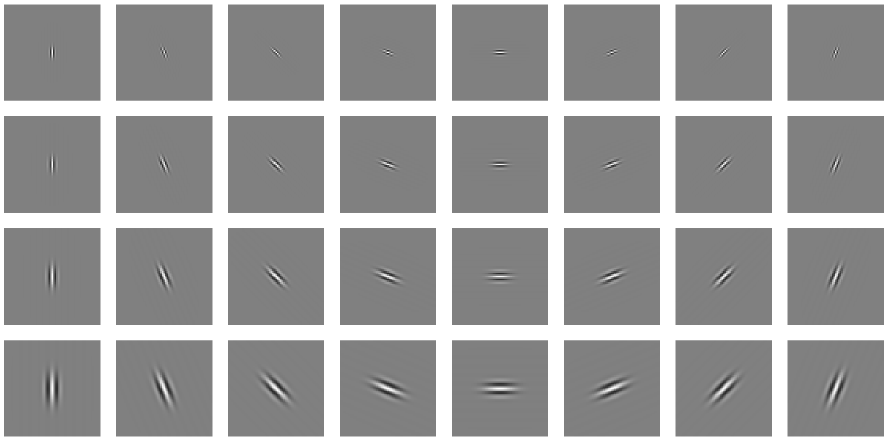


Figure 8: Different Gabor filters.

might charge the image of your choice on your hardrive using the file button.

5.4 Denoise TV

The Denoise TV button is the main feature of the plugin. It implements an algorithm similar to the one described in [1, 2]. This algorithm is iterative and solves a convex optimization problem. The plugin allows to set the iterations number. The more iterations, the closer to the solution of the optimization problem. Contrarily to some standard algorithms in image processing (e.g. Richardson-Lucy deconvolution algorithms, iterative diffusion, etc...), the solution will not change if many iterations are performed. It is important to make enough iterations to find a solution sufficiently close to what you are looking for, but making too many will not really change the solution.
The user should thus try to set a number of iterations sufficiently large for the algorithm to converge and sufficiently small for the algorithm to be fast. In most cases, setting between 30 and 100 iterations will be enough. At the end of

the iterative denoising process, a plot will popup and show an increasing curve. If this curve reaches a plateau, then the algorithm converged. If not, you should try to set more iterations. This is illustrated on Figure 9.

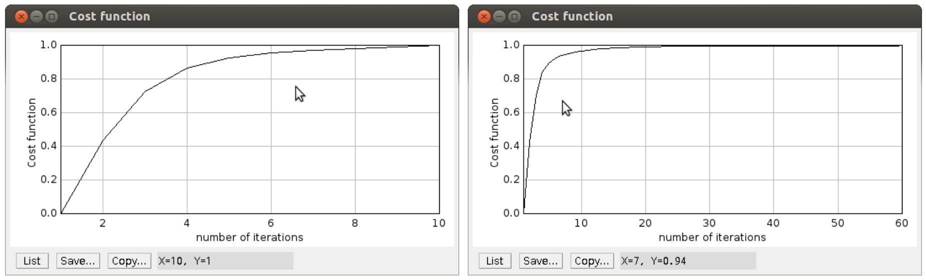


Figure 9: Assessing good or bad convergence. Left: the algorithm did not converge yet, the cost function is still increasing after 10 iterations. Right: the algorithm converged. A plateau is reached after around 30 iterations.

5.5 Denoise H1

The Denoise H1 function is an alternative to Denoise TV. The difference is that it implements an H^1 regularization (l^2-norm of the gradient) rather than a TV semi-norm regularization. The H^1 regularization is known to perform rather poorly on images, but it can be solved very rapidly by using the Fast Fourier Transform. We thus added this feature in the second version of the plugin.
The interest of this function is that it allows you to solve your denoising problems (much) faster than Denoise TV. We still implemented a simple iterative method to ease the parameter selection. One way to proceed with your images is to:

- First use the Denoise H1 method to assess the quality of the choice of a filter rapidly.
- Once you found the correct filter, use the Denoise TV algorithm to get better quality results.

5.6 How to choose the noise level parameter?

Another difficulty is to set the noise level parameter. We tried to simplify this choice by developing a mathematical theory described in [2]. The basic principle is to find a parameter η (called noise level in the plugin) such that the norm of the noise will be approximately equal to η multiplied by the norm of the noisy image.
For instance if you set the noise level to 0.1, the noise norm should be approximately equal to 10% of the image norm.
In a mathematical language, if u_0 denotes the image you want to denoise and $b_i(\eta_i)$ denotes the i-th noise component with parameter η_i, the plugin will ensure

that:

$$\|b_i(\eta_i)\|_2 \leq \eta_i \|u_0\|_2$$

and that

$$\|b_i(\eta_i)\|_2 \simeq \eta_i \|u_0\|_2.$$

Important note: the quantities will be near equal for the Denoise H1 function. There is thus no reason to set a noise level higher than 1. For the Denoise TV function, the ration between $\eta_i \|u_0\|_2$ and $\|b_i(\eta_i)\|_2$ should vary between 1 and 5. It might thus makes sense to increase the noise level above 1 in some cases. In our experiments, setting the noise level to 1 usually usually provides good enough results.

5.7 Are the results and parameters saved somewhere?

Yes they are. At the end of a denoising process, all the informations you used to denoise your images are saved as well as the resulting images. You can find the location of the save file in the log of the main interface. The location is usually the repertory VSNR. It is created in the repertory of your image. The informations are saved in a file that is called "Name_of_your_image_vsnrinfo.txt". If more than one processing is made on the same image, the above file will contain the informations for all the processings you performed.
The following informations are given:

- Date of processing.
- Hour of processing.
- Number of iterations.
- Number of filters.
- Parameters of the filters.

Moreover, the resulting images are also saved in the vsnr repertory.

6 Practical examples

In this section we provide a few examples of how to use the plugin in practice. If you wish to make these experiments on your own, you can download the images used as examples here: `http://www.math.univ-toulouse.fr/~weiss/PageCodes.html`.

6.1 Example 1: 2D denoising (one filter).

The goal of this paragraph is to denoise the image displayed in Figure 10. The noise consists of (additive) horizontal stripes.
The stripes cover the whole image range, so it is natural to set up a very elongated Gabor filter. To do so you can perform the following actions:

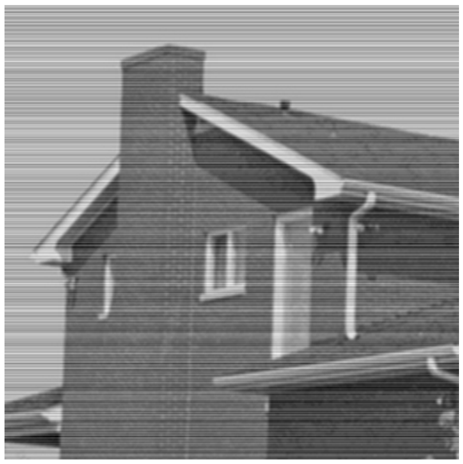


Figure 10: Denoising example 1.

- Open the stripy house image and select it in the plugin.
- Click on Gabor filter and set sigma_x=1000, sigma_y=0.1 (very elongated in the x direction), angle=0, phase=0, lambda=0. You will see that the filter looks like a straight line. Finish by clicking on add filter.
- In the main interface, for the denoise H1 method, you can set noise level = 0.3 (there is about 30% noise in the image). Then you can try denoising the image using the button Denoise H1.
- In order to use denoise TV, you should set the noise level to be greater than 0.3 (see the paragraph about parameter selection). In this example, you can use a value 2 and set the number of iterations to 50.

Important note: the results are saved in 16 bits whatever the format of the original image (actually the algorithm runs with double precision). You might thus see a black images as a result in some cases. Just adjust the contrast and the image will appear.
The results of this procedure are displayed in Figure 11. On my computer, the computing time for DenoiseH1 is less than 1 second, while the computing time for DenoiseTV is around 13 seconds. Notice however that the Denoise TV result is better than the Denoise H1 result. Notice also that the method is able to recover very thin details such as the bricks on the house.

6.2 Example 2: 2D denoising (multiple filters).

The goal of this paragraph is to denoise the image displayed in Figure 12. The noise consists of (additive) horizontal stripes and Gaussian noise. Since there are 2 noise components, 2 filters can be used.

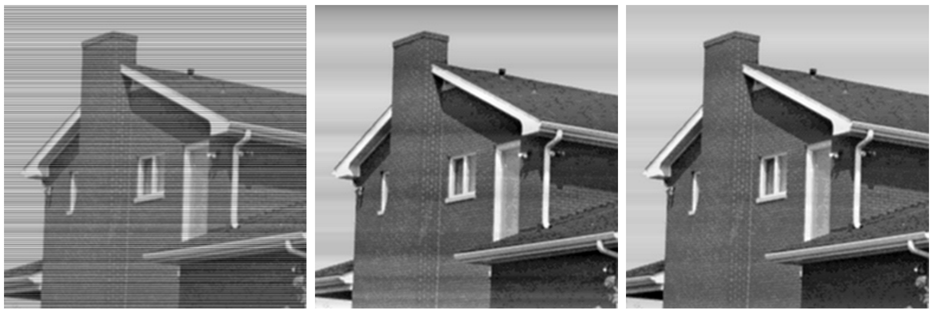


Figure 11: Left: noisy image. Middle: restoration using the H1 method. Right: restoration using the TV method.

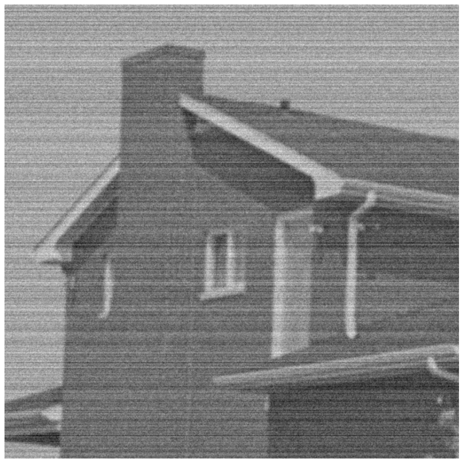


Figure 12: Denoising example 2.

Similarly to example 1, the stripes cover the whole image range, so it is natural to set up a very elongated Gabor filter. Moreover, there is some additive white noise on the image. To remove it you can add a Dirac filter. To denoise the image you can perform the following actions:

- Open the stripy house 2 image and select it in the plugin.
- Click on Gabor filter and set sigma_x=1000, sigma_y=0.1 (very elongated in the x direction), angle=0, phase=0, lambda=0. You will see that the filter looks like a straight line. Finish by clicking on add filter.
- Click on Dirac to add a Dirac filter.
- In the main interface, for the denoise H1 method, you can set noise level of the Gabor filter to 0.3 (there is about 30% noise in the image). You can set the noise level of the Dirac filter to 0.1 (there is about 10% white

noise). Then you can try denoising the image using the button Denoise H1.

- In order to use denoise TV, you can set the noise level of the Gabor filter to 2 and set the noise level of the Dirac filter to 0.2 or 0.5 for instance. Also set the number of iterations to 100 (in this example 50 iterations is too small).

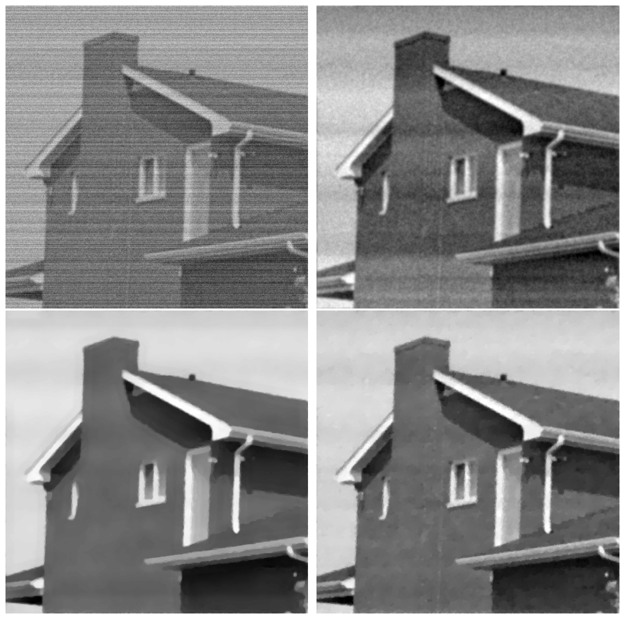


Figure 13: Top: noisy image and restored using the H1 algorithm. Bottom: restoration using the TV method with the noise level set to 0.5 (left) of 0.2 (right).

The results of this procedure are displayed in Figure 13. On my computer, the computing time for DenoiseH1 is less than 1 second, while the computing time for DenoiseTV is around 26 seconds for 100 iterations. This time, the Denoise TV result is much better than the Denoise H1 result. Depending on the processing you want to do after the denoising step, you might prefer either the bottom left (good for segmentation) or bottom right image (good for segmentation).
Important note: once again the results are saved in 16 bits whatever the format of the original image. You might thus see a black images as a result in some cases. Just adjust the contrast and the image will appear.

6.3 Example 3: 3D denoising (3D mode).

In this example we will treat a 3D image. The central slice is displayed in Figure 14. It is a synthetic image that we generated on Matlab. It could represent a 3D multi-cellular tumor spheroid that is used in our laboratory [3]. This example is used to simulated an image obtained with a SPIM (Selective Plane Illumination Microscope) which tends to produce stripes due to laserlight diffraction. In this example, the stripes are vertically elongated.

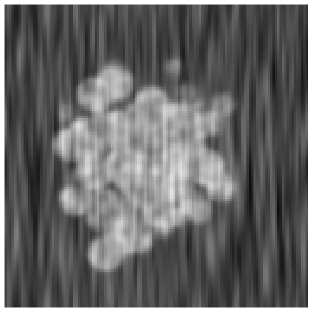


Figure 14: Denoising example 3. This image is a slice of a 3D synthetic image of size 128x128x128. It could represent a spheroid used extensively in our laboratory [3].

To treat the image, first create a Gabor filter with parameters Angle X,Y,Z = 0 and Sigma_x = 1, Sigma_Y=10, Sigma_Z=1. To denoise the image using the H1 method, you can set the noise level to 0.3. The result is displayed in Figure 15 left. To denoise the image using the TV method, you can set the iterations number to 50 and the noise level to 1.5 (see Figure 15 middle) or set noise level=4, iterations number=100 (see Figure 15 right). Once again, the TV method produces much better results but takes a much longer time (around 4 seconds for the Denoise H1 against 4 minutes for the TV method).

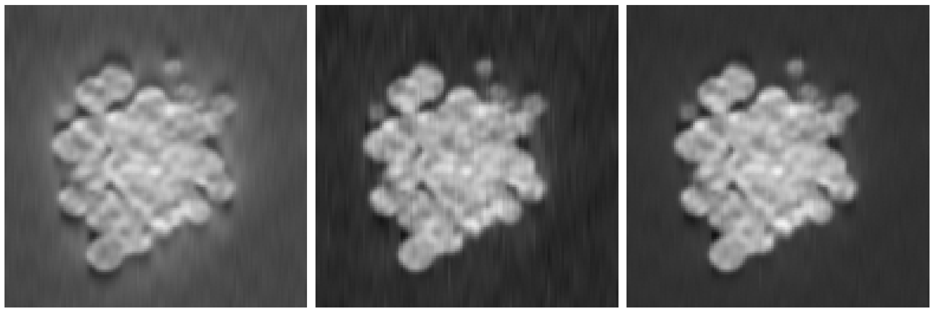


Figure 15: Left: H1 denoising in 3D mode. Middle: TV denoising in 3D mode with noise level=1.5. Right: TV denoising in 3D mode with noise level=4.

7 Known limitations of the plugin

This plugin has some known limitations. We describe some that we can think of below. In case, you find that something should **definitely** be implemented, you can contact me.

7.1 What the heck, it is too slow!

The plugin might take a lot of time to process medium size to large 3D images. Unfortunately, the iterative methods we designed are the state of the art in numerical optimization and it is little likely that new methods can reduce substancially the iterations number. Making good quality denoising methods has a price: the computing times.
If the plugin is too slow for your applications, some solutions can be considered though:

- Using a more powerful (e.g. multi-core) computer. A part of the plugin is multi-threaded and could thus be accelerated substancially on a good machine.
- Use the Matlab implementation we provide here: `http://www.math.univ-toulouse.fr/~weiss/PageCodes.html`. It uses less memory than the Java plugin and might work better.
- Ask us a GPU implementation on CUDA. It is not implemented yet, but might be done in a near future.

7.2 How can I handle color images?

We did not implement a color version of the plugin (due to a lack of time and Fiji expertise). If you wish to treat multi-spectral images, you should first split them into their individual components (e.g. R, G, B) and treat each component independently. We apologize for this limitation.

7.3 I can't handle my very large images!

You might encounter memory troubles with very large images. The reason is simple: we have to store about 20 images in double precision to run the algorithm. This might exceed your RAM, and make the plugin bug. In that case, two solutions can be considered:

- Split your image into smaller pieces and treat each piece independently.
- Use the 2D mode which is much lighter in terms of memory.

Both solutions are actually suboptimal but can produce results similar to the normal implementation.

7.4 What if I have multiplicative noise in my images?

In many situations, the noise is multiplicative and not additive. Our algorithm assumes that the noise is additive. If you use if for multiplicative noise, you could thus obtain suboptimal results. In many situations we encountered the additive noise assumption was near sufficient. If it is not the case in the experiment you perform you could try the following tweak: take the log of your image and process the logarithm. You can find an example of result of this procedure in [2].
Note: this log procedure is not correct from a statisical point of view and another algorithm should be devised in that case. If you have an application for which you think it would be great to have this feature added, you can contact us and we could think of it.

References

[1] J. Fehrenbach, P. Weiss, and C. Lorenzo. Variational algorithms to remove stationary noise. applications to microscopy imaging. 21, 2012.

[2] J. Fehrenbach and P. Weiss. Processing stationary noise: model and parameter selection in variational methods. *arXiv preprint arXiv:1307.4592*, 2013.

[3] C. Lorenzo, C. Frongia, R. Jorand, J. Fehrenbach, P. Weiss, A. Maandhui, G. Gay, B. Ducommun, and V. Lobjois. Live cell division dynamics monitoring in 3d large spheroid tumor models using light sheet microscopy. *Cell division*, 6(1):22, 2011.