# Deconvolution: infrastructure and reference algorithms

Gaëtan Lehmann

August 30, 2010

[1]INRA, UMR 1198; ENVA; CNRS, FRE 2857, Biologie du Développement et Reproduction, Jouy en Josas, F-78350, France.

**Abstract**

The deconvolution, also called deblurring, tries to revert the optical distortion introduced during the aquisition of the image. It is a family of image processing which can be classed in the larger family of image restoration.

Deconvolution is a very difficult problem, and many algorithms have been proposed to solve it, with different strenght and weakness which may depend on the context where they are used. As a consequence, it is desirable to have several algorithms available when trying to restore some images. The different algorithms are often built on a similar principle, making possible to share a large part of their API in their implementation. Also, the most generic operations related to deconvolution should be reusable in order to avoid code duplication and ease the implementation of new algorithms.

In this contribution, the infrastructure for the implementation of several deconvolution algorithms is proposed. Based on this infrastructure, twelve simple deconvolution algorithms of reference are also provided.

## Contents

**1 Introduction** **2**

**2 Infrastructure** **3**

2.1 Generic classes . . . . . 3

itk::BinaryFunctorWithIndexImageFilter . . . . . 3

itk::ComplexConjugateImageFilter . . . . . 3

itk::DivideOrZeroOutImageFilter . . . . . 3

itk::MultiplyByComplexConjugateImageFilter . . . . . 4

itk::LaplacianImageFilter . . . . . 4

itk::FFTConvolveByOpticalTransferFunctionImageFilter . . . . . 4

**3 Calculators** **4**
itk::ImprovementInSignalToNoiseRatioCalculator . . . 4
itk::RelativeChangeCalculator . . . 4
itk::IDivergenceCalculator . . . 5
itk::TotalIntensityRatioCalculator . . . 5

**4 Linear deconvolution** **5**

**5 Iterative deconvolution** **5**

**6 Reference algoritms** **6**
6.1 Linear algorithms . . . 6
itk::WienerDeconvolutionImageFilter . . . 6
itk::TikhonovMillerDeconvolutionImageFilter . . . 6
itk::RegularizedLeastSquaresDeconvolutionImageFilter . . . 6
6.2 Iterative algorithms . . . 7
itk::VanCittertDeconvolutionImageFilter . . . 7
itk::JanssonVanCittertDeconvolutionImageFilter . . . 7
itk::LandweberDeconvolutionImageFilter . . . 7
itk::RichardsonLucyDeconvolutionImageFilter . . . 7
itk::MaximumEntropyRichardsonLucyDeconvolutionImageFilter . . . 7
itk::DampedRichardsonLucyDeconvolutionImageFilter . . . 8
itk::ConchelloIntensityPenalizationImageFilter . . . 8
itk::TikhonovMillerRichardsonLucyDeconvolutionImageFilter . . . 8
itk::PoissonMaximumAPosterioriDeconvolutionImageFilter . . . 8

**7 Development version** **8**

**8 Conclusion** **9**

## 1 Introduction

The deconvolution is the process used to remove the blur introduced during the image acquisition. This blur can have several origins: it can be caused by the optical properties of the equipment, or by a camera shake for example. The blur can then be caracterised by a point spread function, which may or may not be constant in the whole image.

The deconvolution is a widely used methods in several fields including astronomy, where it has been heavily developped during the optical problem of the Hubble space telescope, and fluorescence microscopy, where it is used as an alternative or in conjunction with physical methods aimed to improve the image quality, like the 2-photons or the confocal microscopy.

The image formation can be modelised as follow:

$$I = N_P(O \otimes P) + N_G \tag{1}$$

where $I$ is the observed image, $O$ is the unknown original image, $P$ is the point spread function, $\otimes$ is the convolution process, $N_G$ is an additive gaussian noise and $N_P$ is a poisson noise. $N_G$ and $N_P$ can be more or less important, and may be negleted depending of the cases.

Several approaches have been considered over the time to solve that difficult problem, leading to two groups of algorithms:

- linear algorithms: the computation is made in a fixed number of passes. They are generally very fast and very useful in previsualization.
- iterative algorithms: the computation of the unblurred image is refined iteration after iteration. While generally slower than the linear algorithms, they also generally produce more accurate results, by allowing the integration of constraints during the computation, like the non negativity of the light intensity.

In this contribution, the point spread function is assumed to be known and constant in the whole image. While this is not true in all the cases, this approach allows to use the highly efficient FFT based convolution to implement the deconvolution algorithms. Other cases may be developped in another contribution. The point spread function can be measured or simulated, but none of those methods are provided here, and may be developped in another contribution.

## 2 Infrastructure

There are several common steps in the deconvolution algorithms, being linear or iterative. The most obvious is the transform to the frequency domain to make the computations. This code is already available in another contribution by the same author. Some others steps, mostly related to the iterative deconvolution, are also usable outside of the context of the deconvolution – they have been grouped here in generic filters and calculators.

### 2.1 Generic classes

#### itk::BinaryFunctorWithIndexImageFilter

This filter is very similar to `itk::BinaryFunctorImageFilter`, but also is also passing the index position of the current index to the functor, allowing to apply a transform dependent on the position in the image.

#### itk::ComplexConjugateImageFilter

This filter simply compute the complex conjugate of a complex image pixel-wise.

#### itk::DivideOrZeroOutImageFilter

This filter divides an image by another, and replaces any too small value (defined by the user) by zero. It is very useful to prevent very large value when the result is used as the denominator in a division.

### itk::MultiplyByComplexConjugateImageFilter

This filter combines, in a single step, the complex conjugate transform and the multiplication by a complex number from another image. Its principal interest is to reduce the number of filters involved in a pipeline, but it may also provide some performance enhancements (not measured).

### itk::LaplacianImageFilter

This filter has been modified so that the laplacian kernel can be normalized to one, in order to produced reproductible changes independently of the image spacing. It has also been enhanced to run the expected number of threads.

### itk::FFTConvolveByOpticalTransferFunctionImageFilter

Performs the convolution of an image in the spatial domain by a kernel in the frequency domain. It is implemented with the very simple pipeline FFT-frequency domain multiplication-inverse FFT and is used in many iterative algorithms to reduce the code complexity. An option is available to compute the convolution by the conjugate of the kernel image.

## 3 Calculators

Several calculators are provided, to compute some values of interest on the processed images. Because of the lack of multithreading infrastructure for the calculators in ITK, none of those calculators are multithreaded.

### itk::ImprovementInSignalToNoiseRatioCalculator

Compute the signal to noise improvement based on three images: the original image, the blurred image and the restored image. This is useful to measure the efficiency of a restoration procedure, but requires to have the original image, and so is restricted to a simulated blur.

$$improvment = 10 \times log_{10} \left( \frac{\sum (I_{blurred} - I_{origin})^2}{\sum (I_{restored} - I_{origin})^2} \right) \tag{2}$$

### itk::RelativeChangeCalculator

Computes the relative change of an image after a transform, using the formulae:

$$change = \frac{\sum I_n \times I_{n-1}}{\sum 1} \tag{3}$$

it is used as a criteria to stop the iteration process in the iterative deconvolution.

itk::IDivergenceCalculator

Computes the idivergence between two images. The formulae is

$$idiv = \sum \begin{cases} log(I_n/I_{n+1}) - (I_n - I_{n+1}), & \text{if } I_n \neq 0 \text{ and } I_{n+1} \neq 0 \\ -(I_n - I_{n+1}), & \text{otherwise} \end{cases} \tag{4}$$

This parameter is widely used in case of poisson noise. Note that the 0 case is often not documented and as been chosen as it is in this definition by the author.

itk::TotalIntensityRatioCalculator

Computes the ratio of the total intensity of two images. This calculator allows to verify a desirable feature of some deconvolution algorithms: the non modification of the global intensity in the image.

## 4 Linear deconvolution

Linear deconvolution algorithms are made in a non-iterative way, making them usually faster than the iterative deconvolution algorithms. They are mostly implemented with FFT based convolution, that's why all the implemation provided here are subclasses of `itk::FFTConvolutionImageFilterBase`. They don't have any specific needs which are not covered by this base class, and thus no special base class has been develop for this group of filters.

## 5 Iterative deconvolution

Iterative deconvolution algorithms exhibits some less usual construction than the linear ones. A base class, `itk::IterativeDeconvolutionImageFilter` has been developped to share some code bitween the different algorithm implementations and make them easier to code.

An iterative algorithm is usually quite long to run, so we want a little more observability and possibilities of interaction during the update process than with a linear algorithm.

The filter should:

- be able to start with a different image than the input image. This should allow to restart a deconvolution where the process has stopped and to use precondionned images to start the deconvolution with;
- announce when an iteration is completed;
- expose the current iteration number;
- expose the result of the last iteration to let the user visually check the deconvolution result during the process;
- be able to stop at any iteration if the user decide to do so;
- be able to run for a given number of iterations;

- be able to stop if the process has reach a stable state;
- provide the basic infrasctructure for the iteration to make the filters easier to implement;
- allow the user to modify the image at each iteration to add a smoothing step for example.

Those features have been implemented in `itk::IterativeDeconvolutionImageFilter`, as a subclass of `itk::FFTConvolutionImageFilterBase`.

## 6 Reference algoritms

Some reference algorithms implementations are provided. In the next equations, $I_\omega$ is the image $I$ in the frequency domain.

### 6.1 Linear algorithms

#### itk::WienerDeconvolutionImageFilter

An additive gaussian noise is assumed with the Wiener deconvolution.

$$\hat{I_\omega} = \frac{I_\omega P^*_\omega}{|P_\omega|^2 + \gamma} \tag{5}$$

where $\gamma$ is a user defined parameter. $\gamma$ depends on the amount of noise in the image and is usually in the range 0.001 to 0.1.

#### itk::TikhonovMillerDeconvolutionImageFilter

An additive gaussian noise is assumed with the linear Tikhonov-Miller deconvolution.

$$\hat{I_\omega} = \frac{I_\omega P^*_\omega}{|P_\omega|^2 + \gamma |R_\omega|^2} \tag{6}$$

where $R$ is a regularization operator – usually a laplacian – and $\gamma$ is a user defined parameter. $\gamma$ depends on the amount of noise in the image and is usually in the range 0.001 to 0.1. If $R$ is the identity transform, the direct Tikhonov-Miller deconvolution is equivalent as the Wiener deconvolution.

#### itk::RegularizedLeastSquaresDeconvolutionImageFilter

$$\hat{I_\omega} = \begin{cases} \frac{I_\omega P^*_\omega}{|P_\omega|^2}, & \text{if } |P_\omega|^2 > \alpha \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

## 6.2 Iterative algorithms

#### itk::VanCittertDeconvolutionImageFilter

$$\hat{I}_n = \hat{I}_{n-1} + \alpha \left( I - P \otimes \hat{I}_{n-1} \right) \tag{8}$$

An optional non negativity constraint can be applied at each iteration.

#### itk::JanssonVanCittertDeconvolutionImageFilter

$$\hat{I}_n = \hat{I}_{n-1} + \alpha \left( 1 - \frac{2}{B-A} |\hat{I}_{n-1} - \frac{A+B}{2}| \right) \left( I - P \otimes \hat{I}_{n-1} \right) \tag{9}$$

where $A$ is the minimum value in the image an $B$ the greatest value in the image. In our implementation, it is restricted to $A = 0$, which leads to

$$\hat{I}_n = \hat{I}_{n-1} + \alpha \left( 1 - \frac{2|\hat{I}_{n-1} - B/2|}{B} \right) \left( I - P \otimes \hat{I}_{n-1} \right) \tag{10}$$

An optional non negativity constraint can be applied at each iteration.

This filter is implemented by subclassing `itk::VanCittertDeconvolutionImageFilter` and slightly modifying its internal pipeline.

#### itk::LandweberDeconvolutionImageFilter

$$\hat{I}_n = \hat{I}_{n-1} + \alpha P^T \otimes \left( I - P \otimes \hat{I}_{n-1} \right) \tag{11}$$

An optional non negativity constraint can be applied at each iteration.

This filter is implemented by subclassing `itk::VanCittertDeconvolutionImageFilter` and slightly modifying its internal pipeline.

#### itk::RichardsonLucyDeconvolutionImageFilter

$$\hat{I}_n = \hat{I}_{n-1} \left( P^T \otimes \frac{I}{P \otimes \hat{I}_{n-1}} \right) \tag{12}$$

#### itk::MaximumEntropyRichardsonLucyDeconvolutionImageFilter

$$\hat{I}_n = \hat{I}_{n-1} \left( P^T \otimes \frac{I}{P \otimes \hat{I}_{n-1}} \right) - T I_{n-1} ln(I_{n-1}) \tag{13}$$

This filter is implemented by subclassing `itk::RichardsonLucyDeconvolutionImageFilter` and slightly modifying its internal pipeline.

This filter doesn't conserve the global intensity in the image.

itk::DampedRichardsonLucyDeconvolutionImageFilter

$$\begin{aligned} U_{n-1} &= -\frac{2}{T^2}\left(I + ln(\frac{P \otimes \hat{I}_{n-1}}{I}) - P \otimes \hat{I}_{n-1} + I\right) && (14)\\ \hat{I}_n &= \hat{I}_{n-1}\left(P^T \otimes \left(1 + U_{n-1}^{N-1}(N-(N-1)U_{n-1})\frac{I - P \otimes \hat{I}_{n-1}}{P \otimes \hat{I}_{n-1}}\right)\right) && (15)\end{aligned}$$

This filter is implemented by subclassing `itk::RichardsonLucyDeconvolutionImageFilter` and slightly modifying its internal pipeline.

itk::ConchelloIntensityPenalizationImageFilter

The Conchello's intensity regularization is implemented as an external regularization filter to plug into the regular Richardson-Lucy filter.

At each iteration, it adds the following computation to avoid over intensifying the bright pixels:

$$\hat{I}_n = \frac{-1.0 + \sqrt{1 + 2\lambda \hat{I}_n}}{\lambda} \tag{16}$$

This filter doesn't conserve the global intensity in the image.

itk::TikhonovMillerRichardsonLucyDeconvolutionImageFilter

$$\hat{I}_n = \frac{\hat{I}_{n-1}}{1 - 2\lambda\Delta_{\hat{I}_{n-1}}}\left(P^T \otimes \frac{I}{P \otimes \hat{I}_{n-1}}\right) \tag{17}$$

This filter is implemented by subclassing `itk::RichardsonLucyDeconvolutionImageFilter` and slightly modifying its internal pipeline.

itk::PoissonMaximumAPosterioriDeconvolutionImageFilter

$$\hat{I}_n = \hat{I}_{n-1} e^{P^T \otimes \left(\frac{I}{P \otimes \hat{I}_{n-1}} - 1\right)} \tag{18}$$

## 7 Development version

A development version is available in a darcs repository at http://mima2.jouy.inra.fr/darcs/contrib-itk/deco

## 8 Conclusion

With this work, we are providing a well constructed infrasctructure which should highly simplify the implementation of deconvolution algorithms with ITK, and should help to keep the API very similar in all the future deconvolution filters.

We have also provided several reference algorithms implementation. While not at the state of the art, those algorithms are considered as reference in the domain, and their simplicity was a great opportunity to ensure a high reusability of the infrastructure code.

Several things are still to be developed to make ITK a real actor in the deconvolution fields:

- some state of the art algorithm implementations. Significant improvements has been made in the last decade in the deconvolution field, and the implemented algorithms, while useful as references, just can't compete with the more recent published algorithms;
- blind deconvolution;
- theorical PSF generation;
- real test cases.

## References

[1] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.