1 Python correction

```python
from astropy.io import fits
import matplotlib.pyplot as plt
import numpy as np
from scipy.misc import imresize
from scipy import signal
import progressbar
```

1.1 Damage modeling

There is no built-in function to generate the checkerboard image. A simple code can be used.

```python
def checkerboard():
    arr = np.zeros((8,8),dtype=int)
    arr[::2,::2] = 1
    arr[1::2,1::2] = 1
    out = imresize(arr,8*np.array(arr.shape),interp='nearest')/255;
    return out;
```

The psf can be generated as a Gaussian psf, or a motion psf. In this tutorial, the motion psf is loaded from a file that comes from the MATLAB® function `fspecial`.

```python
C = checkerboard();
psf = np.load('psf_motion.npy');
```

In order to add Gaussian noise, the following function is used:

```python
def addGaussianNoise(I, sigma=1000):
    """Adds Gaussian noise to an image
    I: original image
    sigma: signal/noise ratio
    """
    I2 = I.copy();
    m=np.min(I);
    M=np.max(I);
    N = (M-m)/sigma*np.random.randn(I.shape[0], I.shape[1]);
    I2 = I2 + N;
    I2[I2>M] = M;
    I2[I2<m]=m;
    return I2, I-I2;
```

The image is blurred and noise is then added.

```python
Cb = signal.convolve2d(C, psf, boundary='wrap', mode='same');
Cbn, noise = addGaussianNoise(Cb);

plt.imshow(Cbn, cmap='gray')
plt.title('Motion blur + Gaussian noise on checkerboard')
plt.show();
```

1.1.1 Optical Transfer Function: OTF

The OTF is the centered Fourier Transform of the PSF (Point Spread Function). The following function is used to get the OTF from the PSF:

```python
def psf2otf(psf, s):
    """
    Get OTF (Optical Transfer Function) from PSF (Point Spread Function)
    OTF is basically the Fourier Transform of the PSF, centered
    psf: PSF
    s: shape of the result, zero-padding is used to center is Fourier
        ↪ Transform
    """
    sh = psf.shape;
    sh = np.array(sh);
    s = np.array(s);
    pad = s - sh;
    h = np.pad(psf, ((0,pad[0]), (0, pad[1])), mode='constant');
    shift = (int(pad[0]/2+1), int(pad[1]/2+1));
    h_centered = np.roll(h, tuple(shift), axis=(0,1));
    h_centered = np.fft.fftshift(h_centered);
    H = np.fft.fft2(h_centered, s);
    H = np.real(H);
    return H;
```

1.2 Simple case: no noise

For the direct reconstruction, just ensure that there is no division by 0. The different results are illustrated in Fig.1.

```python
H = psf2otf(psf, C.shape);
G = np.fft.fft2(Cb);
alpha =  0.001;
F = G / (H+alpha);
fr = np.real(np.fft.ifft2(F));
plt.imshow(fr);
plt.title('Noiseless (only motion blur) direct reconstruction')
plt.imsave("cb_reconstruction.png", fr, cmap='gray');
plt.show()
```

In the presence of noise, the reconstructed image is not perfect.

```python
H = psf2otf(psf, C.shape);
G = np.fft.fft2(Cbn);
alpha =  0.001;
F = G / (H+alpha);
fr = np.real(np.fft.ifft2(F));
```

The automatic evaluation of the noise parameter is done in the Wiener filter by:

```python
SpecPuissNoise=np.abs(np.fft.fft2(noise))**2;
PuissMoyNoise=np.mean(SpecPuissNoise);
SpecPuissImageOrig=np.abs(np.fft.fft2(C))**2;
PuissMoyImageOrig=np.mean(SpecPuissImageOrig);
ratio=PuissMoyNoise/PuissMoyImageOrig;

Hw = np.conjugate(H)/(np.abs(H)**2+ratio);
fr = np.fft.ifft2( Hw * np.fft.fft2(Cbn));
```

1.3 Iterative filters

1.3.1 Van-Cittert iterative filter

The parameter (Jansson parameter) controls the precision of convergence, but a small value requires a high number of iterations (and thus a high computation time).

```python
def vca(g, psf, n_iter, beta=.01):
    """
    Van Cittert iterative filter
    g: noisy image
    psf: point spread function
    n_iter: number of iterations
    beta: Jansson parameter
    """
    H = psf2otf(psf, g.shape);

    # init iterations
    fr = g.copy();
    bar = progressbar.ProgressBar();
    for iter in bar(range(n_iter)):
        estimation = np.real(np.fft.ifft2(H * np.fft.fft2(fr)));
        fr= fr + beta* ( g - estimation);

    return fr;
```

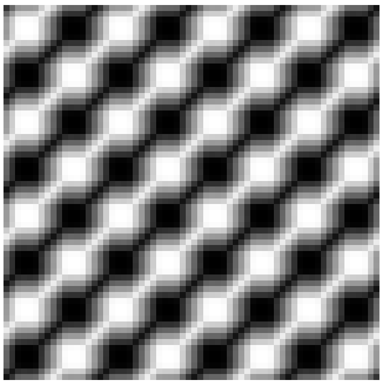
1.3.2 Richardson-Lucy iterative filter

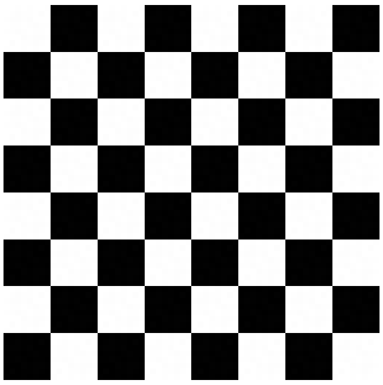
This algorithm is probably the most famous one in the domain. Remember that the symmetry in the space domain becomes the complex conjugate in the spectral domaine (after Fourier transform).

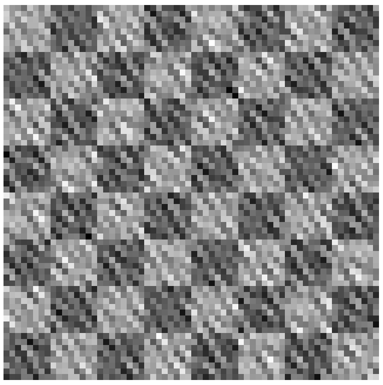
```python
def rla(g, psf, n_iter):
    """
    Richardson Lucy algorithm
    g: noisy image
    psf: point spread function
    n_iter: number of iterations
    """
    H = psf2otf(psf, g.shape);
    # init iterations
    fr = g.copy();

    bar = progressbar.ProgressBar()
    for iter in bar(range(n_iter)):
        estimation = np.fft.ifft2(H * np.fft.fft2(fr));
        M = np.real(np.fft.ifft2(np.conjugate(H) * np.fft.fft2(g /
            ↪ estimation)));
        fr= fr * M;
        fr[fr<0] = 0;

    return fr;
```

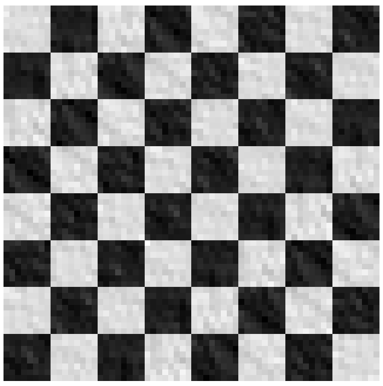
1.4 Astronomy images

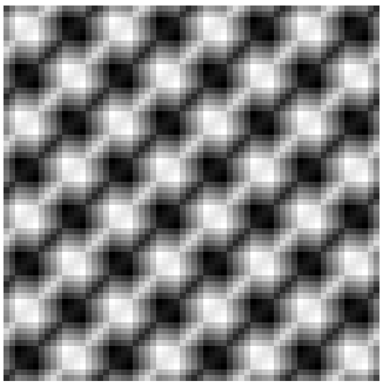
These real images come from the Hubble Space Telescope (HST, see credits), shown in Fig.2. The results for the Richardson-Lucy, Van-Cittert and Landweber algorithms are shown in Fig.3. The direct deconvolution does not give a correct result.

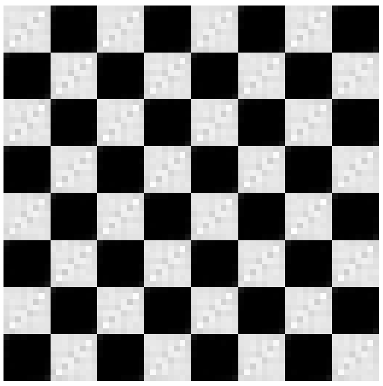

(a) Damaged checkerboard image, with motion blur and Gaussian noise.

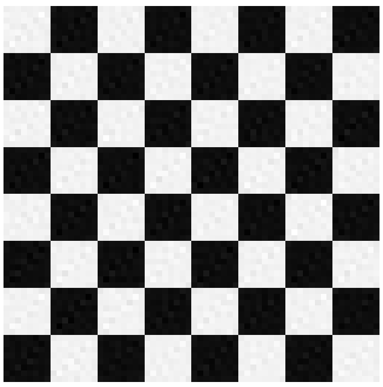

(b) Reconstruction of the checkerboard without noise.


(c) Reconstruction of the checkerboard with noise.


(d) Wiener deconvolution.


(e) Van-Cittert iterative deconvolution with 100 iterations, and Jansson parameter at 0.01.


(f) Richardon-Lucy iterative deconvolution with 1000 iterations.


(g) Landweber iterative deconvolution with 1000 iterations and Jansson parameter at 1.

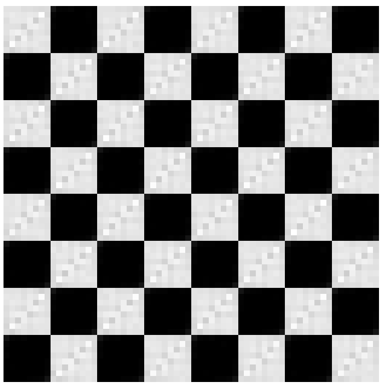
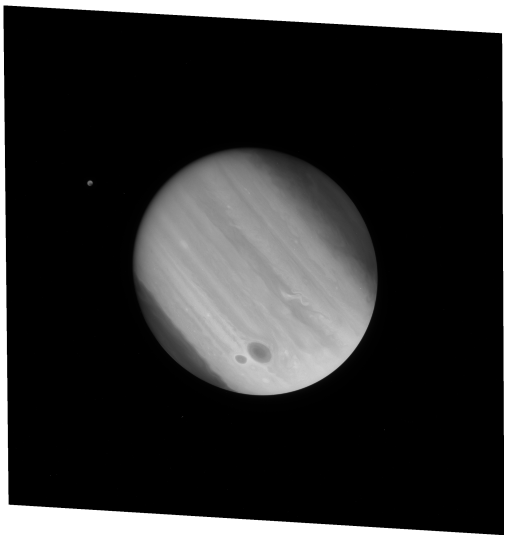
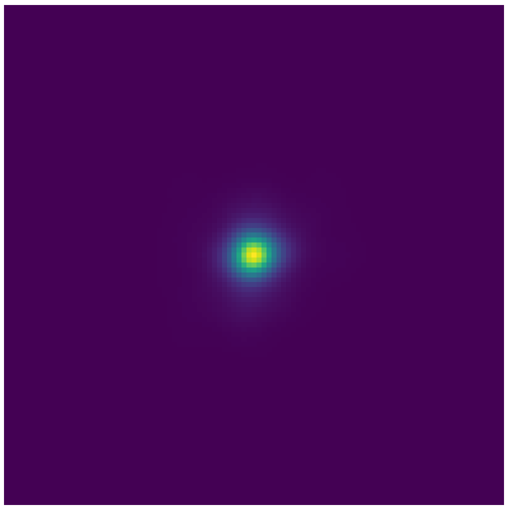
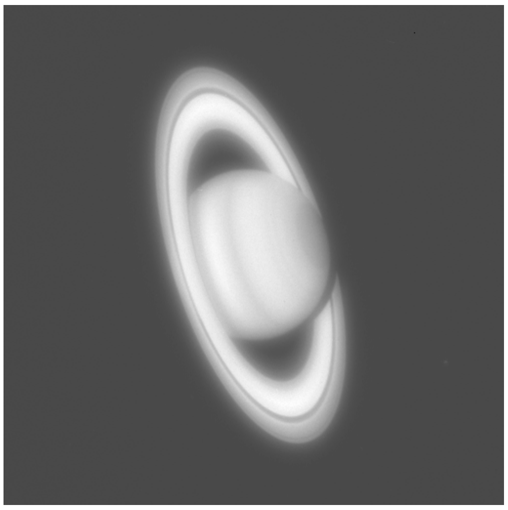

(h) Poisson Maximum A Posteriori iterative deconvolution with 1000 iterations.

Figure 1: Different deconvolution methods applied on the synthetic image 'checkerboard'.


(a) Jupiter.


(b) PSF for the observation of Jupiter.


(c) Saturn.

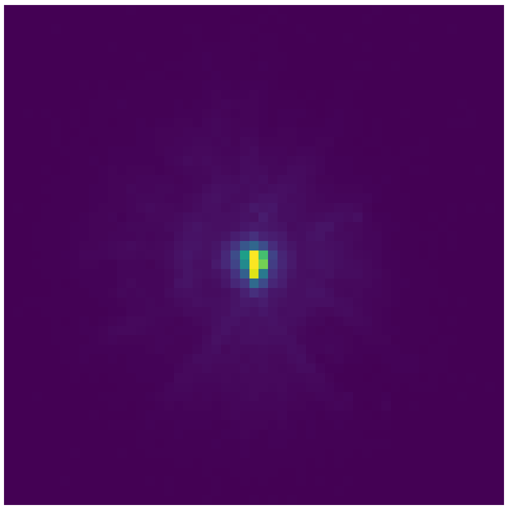
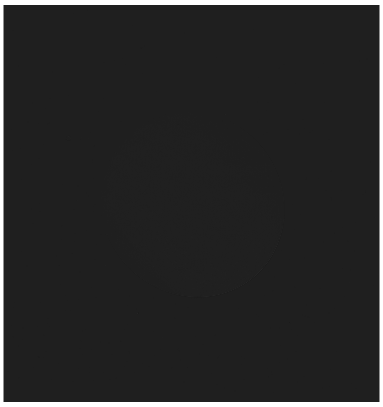
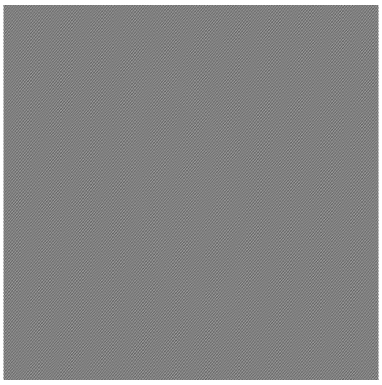
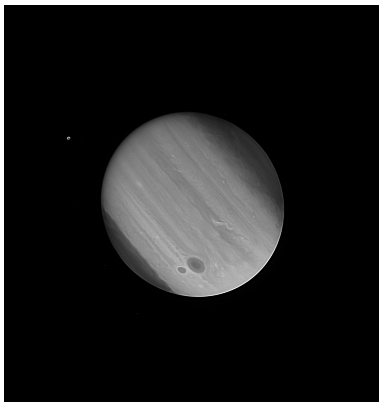
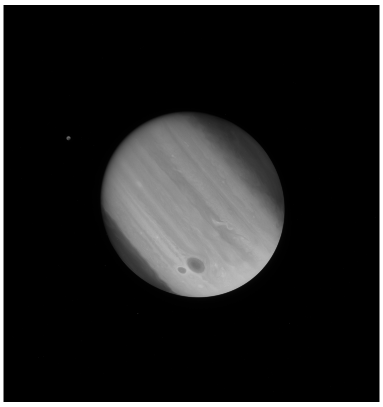

(d) PSF for the observation of Saturn.

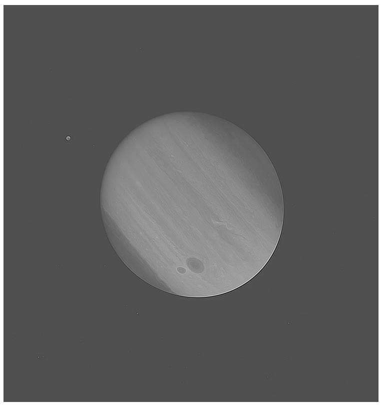
Figure 2: Original images and PSF, from the HST.

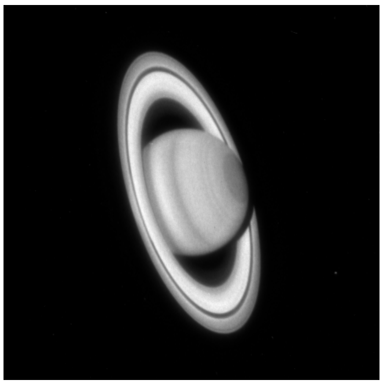

(a) Direct deconvolution.

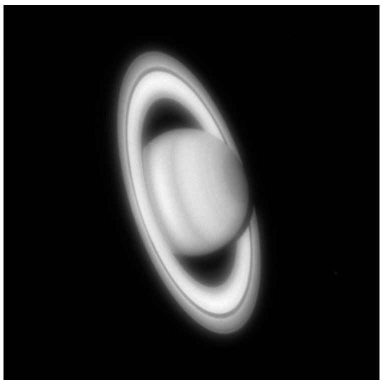

(b) Direct deconvolution.


(c) Richardon-Lucy iterative deconvolution with 10 iterations.


(d) Van-Cittert iterative deconvolution with 10 iterations.


(e) Landweber iterative deconvolution with 200 iterations and Jansson parameter at 1.


(f) Richardon-Lucy iterative deconvolution with 10 iterations.


(g) Van-Cittert iterative deconvolution with 10 iterations.

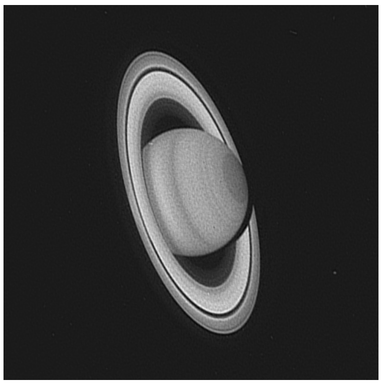

(h) Landweber iterative deconvolution with 200 iterations and Jansson parameter at 1.

Figure 3: Different deconvolution methods applied for the astronomy images.