1 Matlab correction

1.1 Damage modeling

The damaged image is generated using the following method, illustrated in Fig. 1, for a motion perturbation. Notice the 'circular' option in the convolution, which acts as if the psf would be periodic (this is the case in the Fourier space).

```
% generates a noisy image, with motion blur
f = checkerboard(8);
psf = fspecial('motion',7,45);
gb = imfilter(f,psf,'circular');
noise = imnoise(zeros(size(f)),'gaussian',0,0.001);
g = gb + noise;
```

The noise can also be generated by the following command:

```
sigma = .1;
noise = randn(size(f)) * sigma;
```

If the psf used is gaussian, the command will be:

```
psf = fspecial('gaussian', size(f), 1);
```

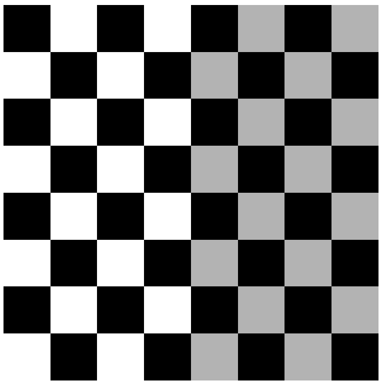
1.2 Inverse filtering with no noise

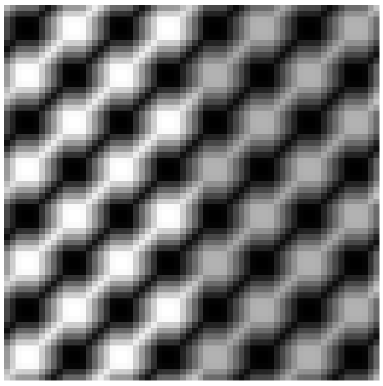
Informations

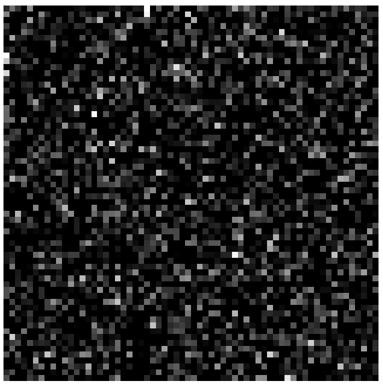
Be careful to use psf2otf instead of fft2. This is basically the same purpose, but the latter lacks centering when performing the array padding.

The inverse filter of the damaged image is performed using the following matlab command. The result is illustrated in Fig. 2.

```
% Fourier Transform of the psf
H = psf2otf(psf, size(f));
G = fft2(g);

% pseudo inverse: ensures there is no division by 0
alpha = 0.01;
```


(a) Original image.


(b) Movement blur.


(c) Gaussian noise.

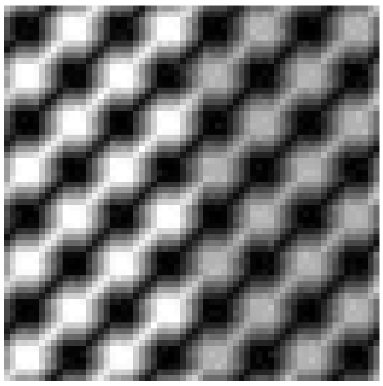

(d) Noisy image.

Figure 1: Simulation of damaged images.

```
F = G./(H + alpha);
fr= ifft2(F);
```

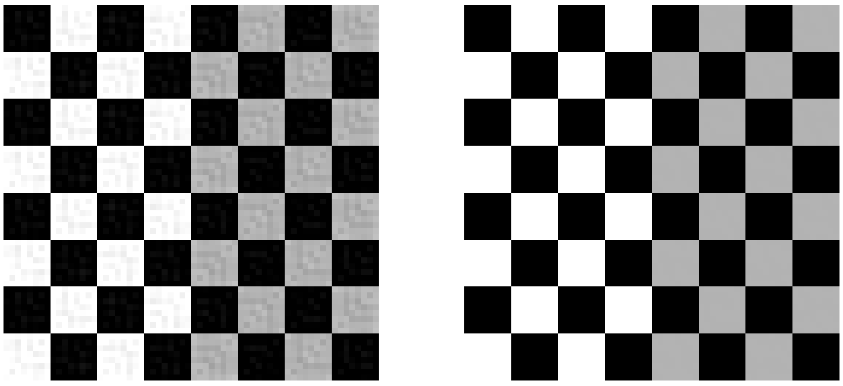

(a) Pseudo inverse filter, $\alpha = 0.1$. (b) Wiener filter.

Figure 2: Motion blur, 45 degrees, with no noise.

1.3 Wiener filter

1.3.1 Simple case: no noise

```
% Fourier Transform of the psf
H = psf2otf(psf, size(f));
G = fft2(g);

% Wiener filter when no noise
% Eliminates zeros values in H
H(H==0) = Inf;
F = G./H;
fw = ifft2(F);
```

The result is illustrated in Fig.2

1.3.2 Noisy images

To evaluate the ratio R, the following commands are performed:

```
SpecPuissNoise=abs(fft2(noise)).^2;
PuissMoyNoise=sum(SpecPuissNoise(:))/numel(noise);
SpecPuissImageOrig=abs(fft2(f)).^2;
```

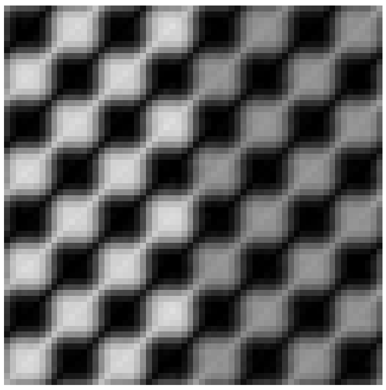


Figure 3: Wiener filter with approximated ratio R.

```
PuissMoyImageOrig=sum(SpecPuissImageOrig(:))/numel(f);
ratio=PuissMoyNoise/PuissMoyImageOrig;
```

And then, the Wiener filter is applied either as a matlab function or as your own function. The result of this second version is presented in Fig. 3.

```
% matlab function
fr2=deconvwnr(g,psf,ratio);
```

```
% create a damaged image with some noise
f=checkerboard(8);
psf = fspecial('gaussian', size(f), 2);
% noise
sigma=.01;
N = randn(size(f)) * sigma;
g = imfilter(f, psf, 'circular') + N;
g = max(0,g);
g = min(1,g);

% performs the restoration
H = psf2otf(psf, size(f)); % Fourier Transform of the psf
Hw = conj(H)./(abs(H).^2+PuissMoyNoise/PuissMoyImageOrig);
fr = ifft2( Hw .* fft2(g));
```

1.4 Iterative filters for noisy images

1.4.1 Van Cittert iterative filter

The VanCittert algorithm is maybe the simplest iterative method. It is really sensitive to noise, as illustrated in Fig. 4.

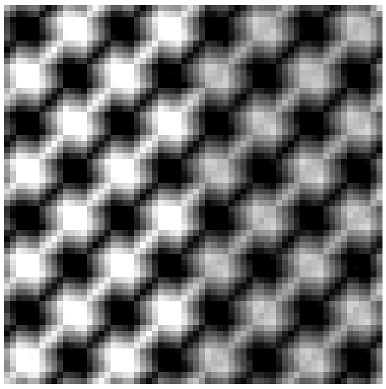


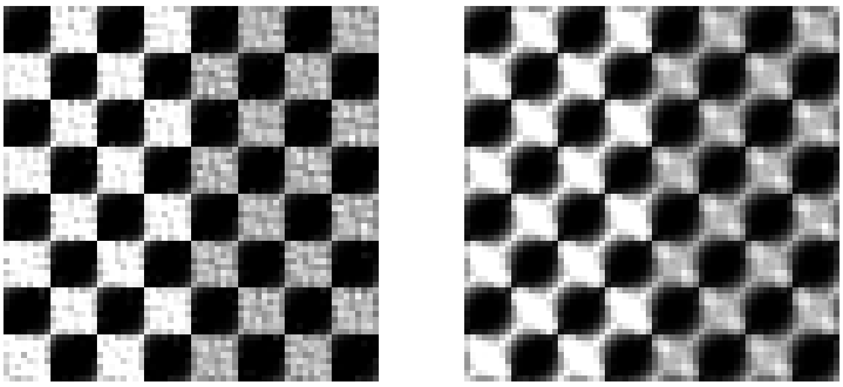
Figure 4: VanCittert iterative restoration algorithm with 10 iterations, applied on an image damaged with motion blur and gaussian white noise. This algorithm is really sensitive to noise.

```
function [ fr ] = vca( g, psf, n_iter)
% Van Cittert iteration algorithm for deconvolution
%
% g: degraded image
% psf: point spread function
% n_iter: number of iterations

% Fourier Transform of the psf
H = psf2otf(psf, size(g));

% initialization
fr=g;

beta = .1; % Jansson parameter
for iter = 1:n_iter
    fr = fr + beta*(g -ifft2(H .* fft2(fr)));
end

end
```

1.4.2 Lucy-Richardson filtering

The Lucy-Richardson deconvolution filter is one of the most employed filter. The convolution operations can be either performed in the Fourier (frequency) domain, or in the spatial domain. For performance reasons, it is usually better to work in the Fourier domain, where the convolution is transform into a simple product. The results are presented in Fig. 5

```matlab
function fr = rla(g, psf, n_iter)
% Richardson-Lucy Algorithm for deconvolution
%
% g: image to restore. initial value of fr is g
% psf: point spread function
% n_iter: number of iterations

% Optical Transfer Function
H = psf2otf(psf, size(g));

% initial value
fr = g;

% iterations
for iter=0:n_iter
    % estimated blurred image
    yk = (ifft2(H.*fft2(fr)));

    M = ifft2(conj(H) .* fft2(g./yk));
    fr = max(0,fr.* M);
end
```



(a) Algorithm with 100 iterations. (b) Algorithm with 10 iterations.

Figure 5: Lucy-Richardson algorithm applied on the chessboard image with motion blur and gaussian white noise.

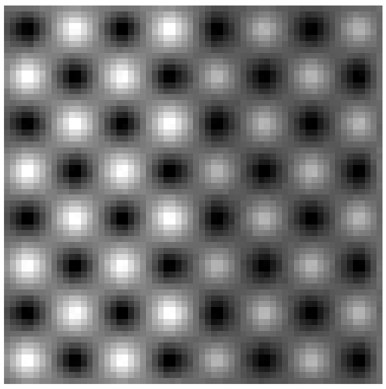
1.5 Blind deconvolution

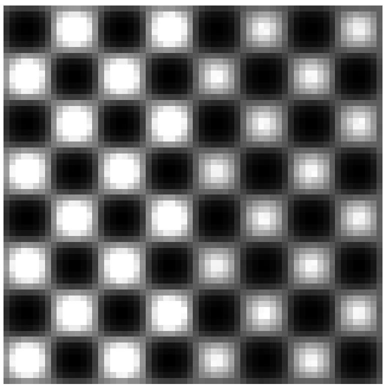
The first step is to generate a degraded image, as before.

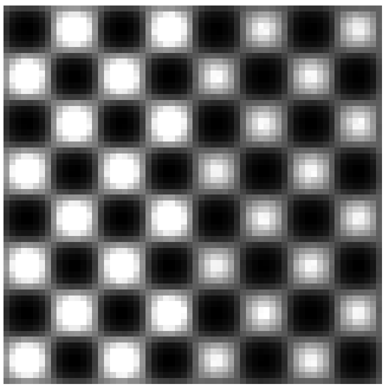
```
f=checkerboard(8);
S = 10; % size of PSF
psf=fspecial('gaussian',S,10);
sd=0.01; % std deviation of the generated noise

% add some gaussian noise with 0 mean and variance sd^2
g=imnoise(imfilter(f,psf, 'circular'),'gaussian', 0, sd^2);
```

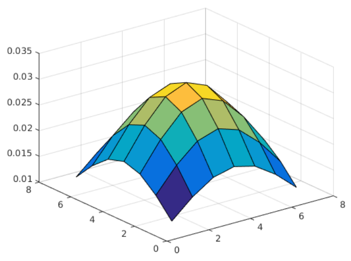
The dampar variable is a damping parameter to minimize the noise (see matlab help).

```
% damping parameter
dampar=10*sd;

% create initial psf
initpsf=ones(S);

% apply blind deconvolution for nb_iterations
nb_iterations = 5;
[fr_blind , psfe]=deconvblind(g,initpsf , nb_iterations , dampar);
```


(a) Original image.


(b) Blind deconvolution for 100 iterations.


(c) Blind deconvolution for 5 iterations.


(d) PSF used for image degradation.

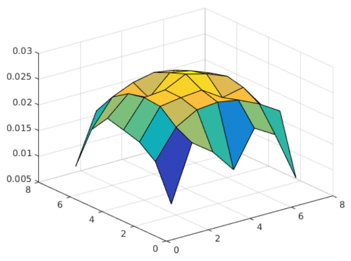
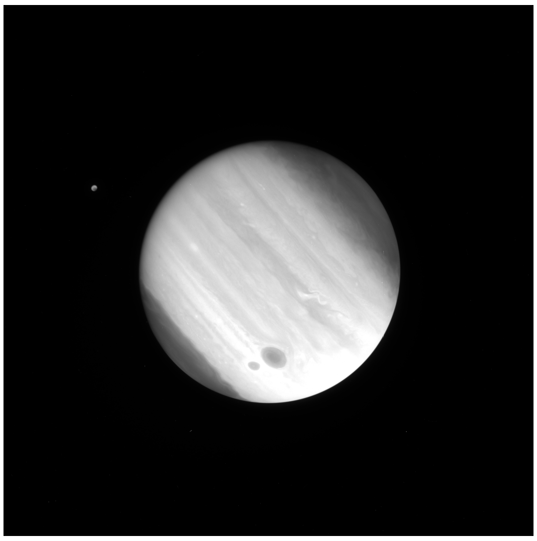

(e) PSF estimated after blind deconvolution.

Figure 6: Blind deconvolution


(a) Original image of Jupiter.

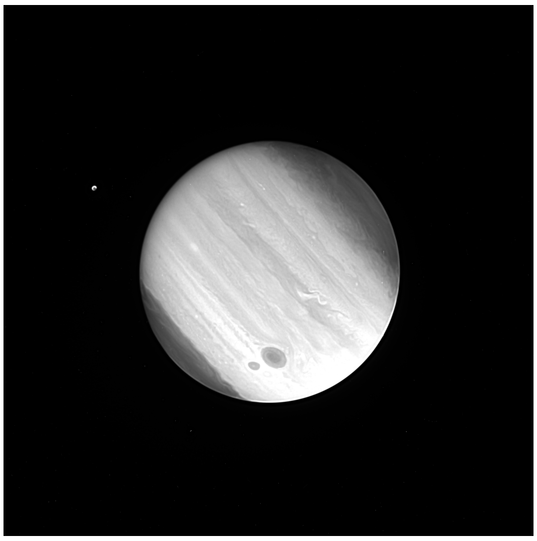
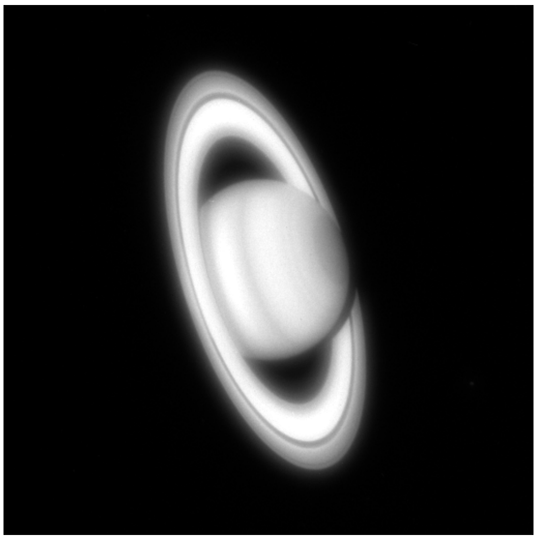
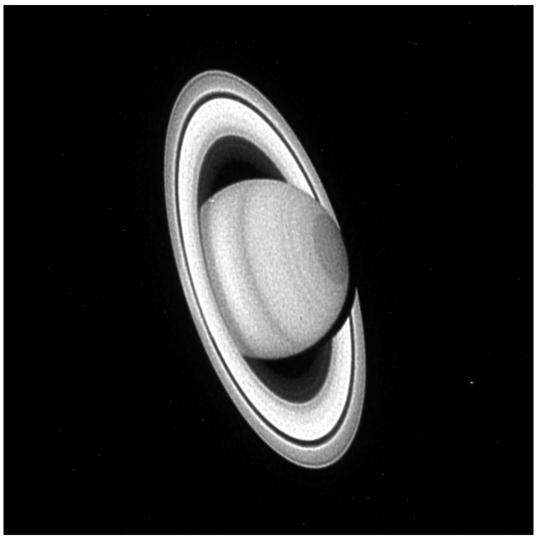

(b) Jupiter image after Lucy-Richardson algorithm with 10 iterations.

Figure 7: Jupiter, from Hubble Space Telescope, ads/Sa.HST#ic3g01qlq.



(a) Original image of Saturn.



(b) Saturn image after Lucy-Richardson algorithm with 100 iterations.

Figure 8: Saturn, from Hubble Space Telescope.