



INTRODUCTION AU TRAITEMENT DES IMAGES

Rapport TP02 IMA201

Réalisé par:

Adnane EL BOUHALI

Encadré par:

GOUSSEAU Yann, LECLAIRE Arthur, LESNÉ Gwilherm, ROUX
Michel

Etablissement :

Télécom Paris

Filière :

Image

Année universitaire:

2023 - 2024

Table des matières

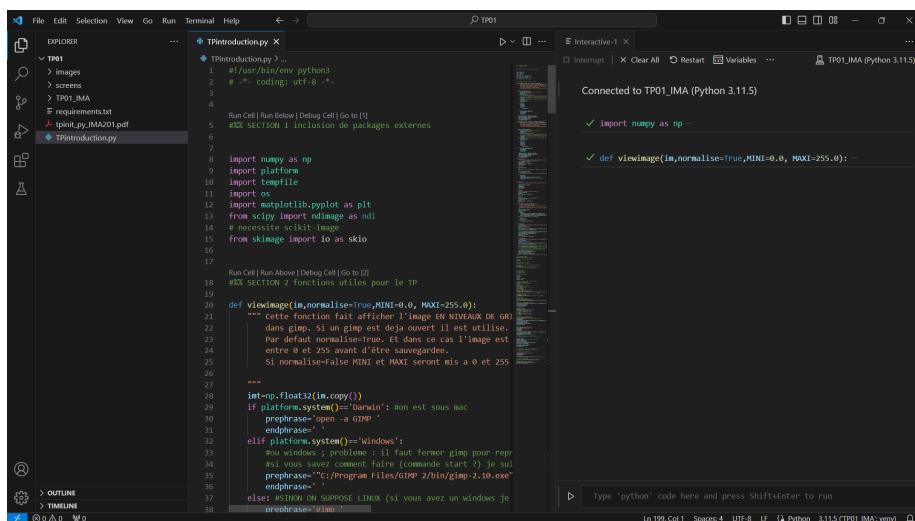
INTRODUCTION AU TRAITEMENT DES IMAGES.....	0
1 Préliminaires.....	2
2 Transformation géométrique.....	2
3 Filtrage linéaire et médian.....	4
4 Restauration.....	8
5 Applications.....	10
5.1 Comparaison filtrage linéaire et médian.....	10
5.2 Calcul théorique du paramètre de restauration.....	10

1 Préliminaires

Tout d'abord, nous préparons l'environnement de développement en installant les librairies nécessaires.

```
PS C:\Users\adnan\Desktop\Documents\telecom Paris\Etudes\IMA\IMA201\TPs\TP01> pip freeze > requirements.txt  
PS C:\Users\adnan\Desktop\Documents\telecom Paris\Etudes\IMA\IMA201\TPs\TP01> pip install -r requirements.txt
```

Une fois cette étape terminée, nous pouvons passer à l'exécution des deux premières sections, ce qui nous permettra de commencer le TP.



The screenshot shows a Jupyter Notebook interface with two panes. The left pane, titled 'TP01', displays the code for 'TPintroduction.py'. The right pane, titled 'Interactive-1', shows the Python interpreter output. The code in the notebook includes imports for numpy, platform, and skimage, along with a function definition for 'viewimage'. The interpreter output shows the function being defined and then executed.

```
TP01  
TPintroduction.py  
1 #!/usr/bin/env python3  
2 # -*- coding: utf-8 -*  
3  
4 Run Cell | Run Below | Debug Cell | Go to [1]  
5 #%% SECTION 1 Inclusion de packages externes  
6  
7 import numpy as np  
8 import platform  
9 import tempfile  
10 import os  
11 import matplotlib.pyplot as plt  
12 from scipy import image as ndi  
13 import skimage  
14 import skimage.io as skio  
15  
16  
17 Run Cell | Run Above | Debug Cell | Go to [2]  
18 #%% SECTION 2 Fonctions utiles pour le TP  
19  
20 def viewimage(im,normalise=True,MINI=0.0, MAXI=255.0):  
21     """ Cette fonction permet de visualiser une image dans un environnement de dev.  
    Par défaut si un gimp est déjà ouvert il est utilisé.  
    Par défaut normalise=True. Et dans ce cas l'image est  
    entre 0 et 255 avant d'être sauvegardée.  
    Si normalise=False MINI et MAXI seront mis à 0 et 255  
    """  
22  
23     im=np.float32(im.copy())  
24     if platform.system()=='darwin': mon est sous mac  
25         prephrase='open -a GIMP'  
26         endphrase=''  
27     elif platform.system()=='windows':  
28         #si tu as windows : il faut former pimpl pour rapp  
         #si vous savez comment faire (commande start ?) je suis  
         prephrase='"/Program Files/GIMP 2/bin/gimp-2.10.exe"  
         endphrase=''  
29     else: #SI NON ON SUPPOSE LINUX (si vous avez un windows je  
30         prephrase='xdg-open'
```

Connected to TP01_IMA (Python 3.11.5)
import numpy as np...
def viewimage(im,normalise=True,MINI=0.0, MAXI=255.0):

2 Transformation géométrique

Utilisons la fonction rotation avec les options clip = True (à gauche) et False (à droite).



Maintenant, regardons la différence entre la méthode à plus proche voisin (à gauche) et la méthode bilinéaire (à droite).



La méthode bilinéaire donne généralement des résultats de meilleure qualité, on ne peut pas le voir vraiment dans cet exemple, mais on le verra quand le nombre de rotations à faire est important.

A gauche, on a la méthode du plus proche voisin, et à droite la méthode bilinéaire.

Que constatez-vous sur une image qui aurait subi huit rotations de 45 degrés (en bilinéaire et en plus proche voisin) ?



On remarque que la méthode bilinéaire est bien meilleure.

Que constatez-vous si vous appliquez la rotation avec un facteur de zoom inférieur à 1 (par exemple 1/2) ? Qu'aurait-il fallu faire pour atténuer l'effet constaté ?

On regarde la différence entre la méthode à plus proche voisin et la méthode bilinéaire en effectuant un zoom inférieur à 1.



On remarque un phénomène d'aliasing, on a des artefacts qui apparaissent (puisque'on effectue un zoom sans méthode d'interpolation). Pour atténuer ce phénomène, il faudrait filtrer l'image avec un filtre passe bas avant d'appliquer la fonction rotation (typiquement un filtre gaussien).

3 Filtrage linéaire et médian

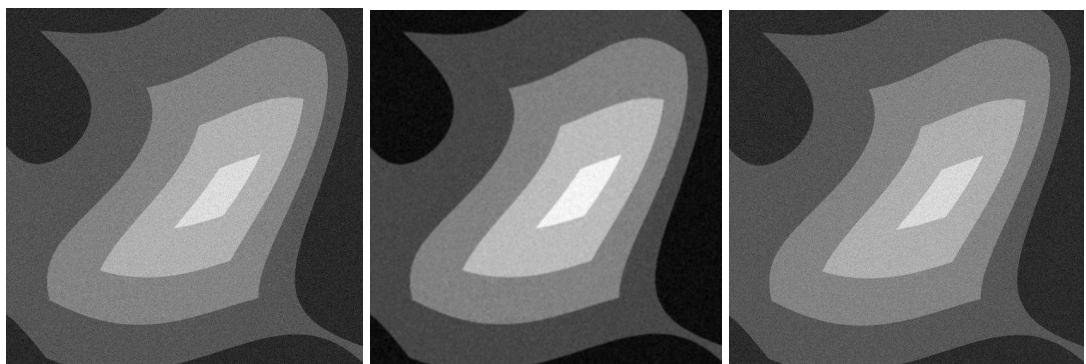
Expliquer le rapport entre la taille du noyau (size) renvoyé par get_gau_ker et le paramètre cette commande.

Le paramètre s détermine la taille du noyau gaussien généré par la fonction **get_gau_ker**. Plus il est grand, plus la taille du noyau sera grande, et la gaussienne sera plus étalée.

Effectuons un filtrage linéaire de l'image **lena.tif** bruitée (à gauche) avec un noyau gaussien (au milieu) et un noyau constant (à droite).



On fait de même avec l'image **pyramide.tif**.



Expliquez comment on peut évaluer (sur des images aussi simples) la quantité de bruit résiduel

Il est possible de quantifier le bruit résiduel en examinant la variation de la teinte grise au sein des zones uniformes après l'application du filtre linéaire gaussien grâce à la fonction **var_image**.

```
var_image(filtre_lineaire(im_br, noyau_gaussien), 0, 0, 1, 1)
✓ 0.0s
1.607826531060316
```

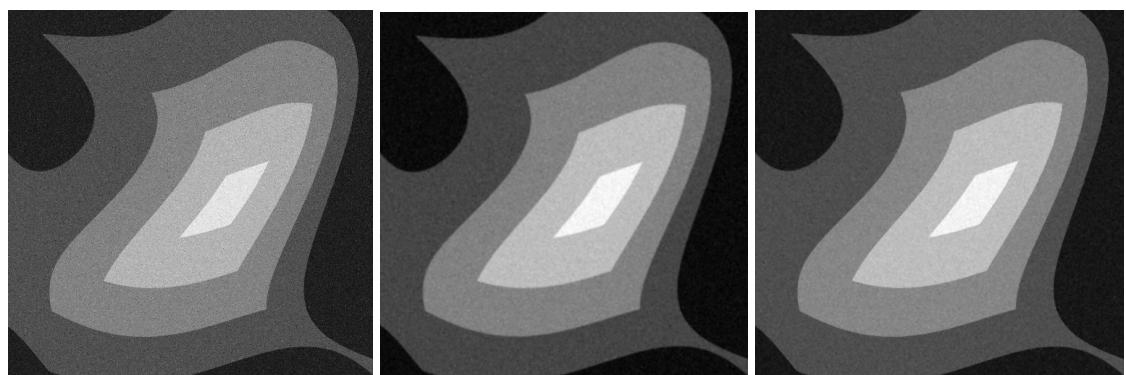
<pre>var_image(im_br, 0, 0, 1, 1)</pre>	<pre>var_image(im, 0, 0, 1, 1)</pre>
✓ 0.0s	✓ 0.0s
53.01246002525474	0.0

Appliquer un filtrage médian a une image bruitée et comparer le résultat avec un filtrage linéaire.

A gauche on retrouve l'image bruitée, au milieu on retrouve l'image filtrée par un filtre linéaire avec un noyau gaussien, et à droite l'image filtrée par un filtre médian.

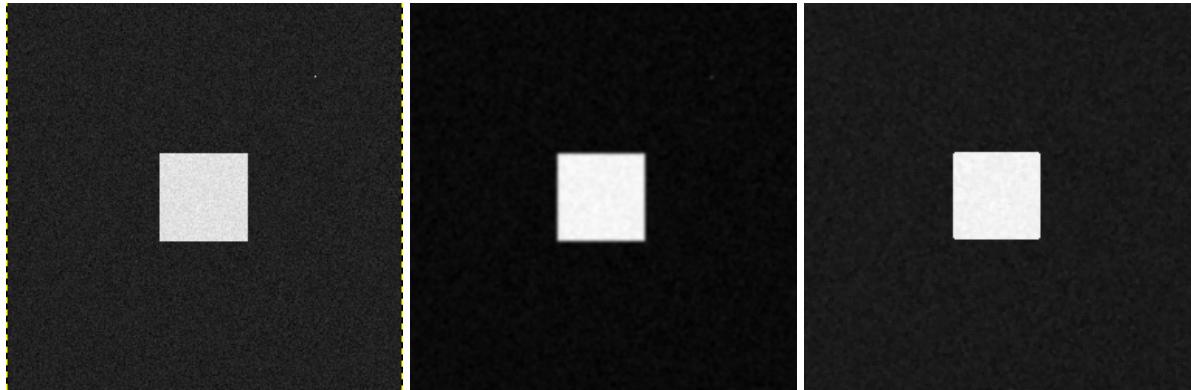


Faites une comparaison linéaire/médian sur l'image pyra-impulse.tif. Que constatez-vous ?



Le filtre médian calcule la médiane des pixels voisins, ce qui le rend excellent pour supprimer le bruit, notamment lorsque des pixels très différents sont isolés. En revanche, le filtre gaussien, qui favorise les pixels du centre, conserve mieux les contours de l'image mais peut rendre les zones sombres encore plus sombres.

Expliquer la différence de comportement entre l'usage linéaire et médian sur le point lumineux situé en haut à droite de l'image carre orig.tif.

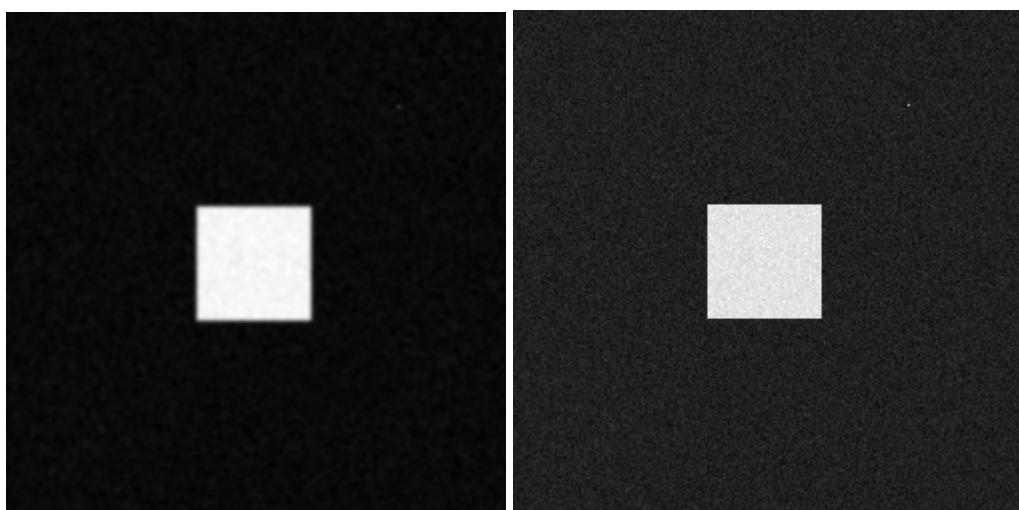


Lorsque nous avons un point blanc entouré de points noirs dans une image, cela affecte les filtres linéaires comme le filtre moyen ou gaussien. Bien que leur effet sur le point blanc soit atténué en raison de sa singularité, il subsiste une zone légèrement plus claire dans le résultat final de l'image.

En revanche, le filtre médian traite le point blanc comme une valeur aberrante. Par exemple, si plusieurs points noirs sont présents avec un seul point blanc dans une série de valeurs, la médiane reste un point noir. Par conséquent, le point blanc disparaît complètement de l'image lorsque le filtre médian est appliqué, ce qui peut avoir un impact particulier sur les bords de la zone où le point blanc était situé.

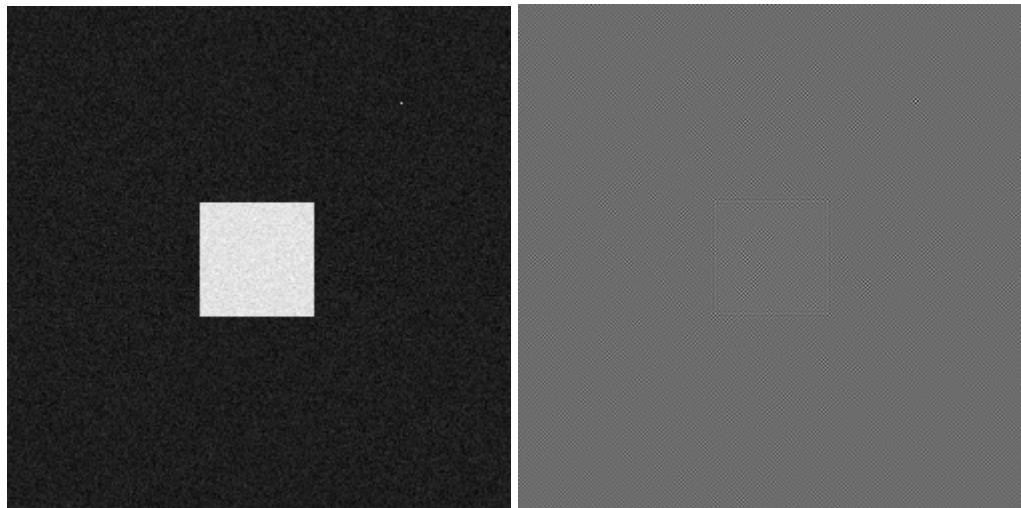
4 Restauration

Appliquer un filtre linéaire à une image puis utiliser la fonction lettre inverse.
Que constatez-vous?



Une fois que la transformée inverse est appliquée, nous obtenons de nouveau l'image d'origine.

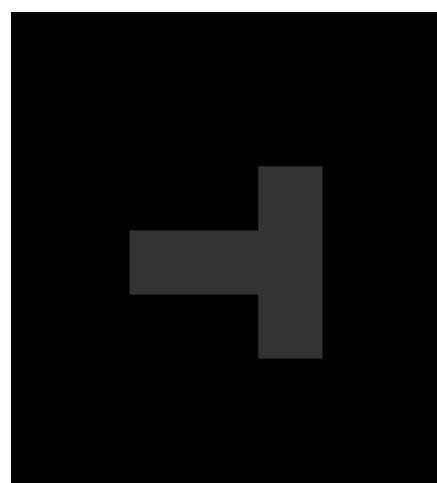
Que se passe-t-il si vous ajoutez très peu de bruit à l'image floue avant de la restaurer par la commande précédente ?



On obtient une image totalement bruitée.

Comment pouvez-vous déterminer le noyau de convolution qu'a subi l'image carre_flou.tif

On peut déterminer le noyau de convolution qu'a subi l'image **carre_flou.tif** en remarquant que le point lumineux s'est transformé en cette forme, qui est une sorte de T couché, ce qui correspond à un noyau $[[0,0,1],[1,1,1],[0,0,1]]$.



Pour tester ceci, on applique un filtre linéaire avec le noyau $[[0,0,1],[1,1,1],[0,0,1]]$ à notre image **carre_orig.tif**.

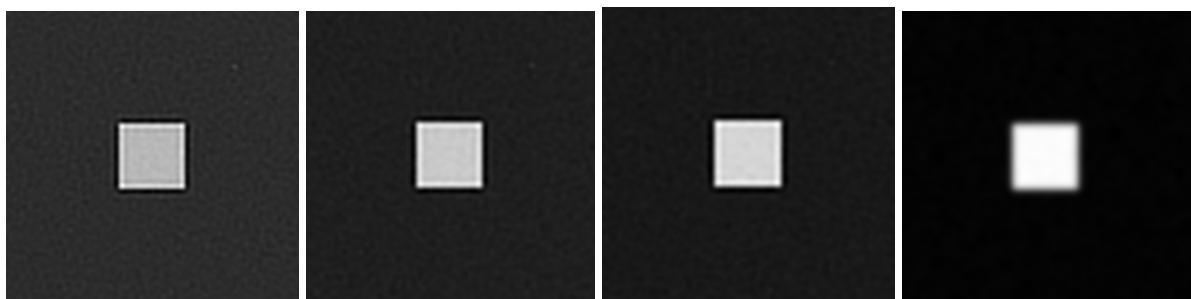
```

656 im=skio.imread('images/carré_orig.tif')
657 im= np.float32(im)
658 viewimage(im)
659 viewimage(filtre_lineaire(im,np.array([[0,0,1],[1,1,1],[0,0,1]])))

```

On obtient exactement l'image **carré_frou.tif**.

Après avoir ajouté du bruit à cette image utilisez la fonction wiener pour restaurer cette image. Faites varier le paramètre et commentez les résultats.



Quand nous augmentons la valeur du paramètre, nous constatons une amélioration progressive des résultats, avec une sous-estimation initiale du bruit, jusqu'à atteindre une performance optimale. Cependant, au-delà de ce point, lorsque nous continuons d'augmenter le paramètre, les résultats commencent à se détériorer, avec une surestimation du bruit.

5 Applications

5.1 Comparaison filtrage linéaire et médian

Pour une image simple telle que carre_orig.tif et un bruit d'écart-type 5, trouver la taille du noyau constant qui réduit le bruit dans les mêmes proportions qu'un filtre médian circulaire de rayon 4. (expliciter l'algorithme utilisé)

```
im=skio.imread('images/carré_orig.tif')
im=noise(im,5)
x=var_image(median_filter(im,typ=2,r=4),0,0,1,1)
for i in range(0,100):
    y=var_image(filtre_lineaire(im,get_cst_ker(i)),0,0,1,1)
    if(np.around(y,2)==np.around(x,2)):
        print(i)
✓ 0.7s
```

1

5.2 Calcul théorique du paramètre de restauration

Modifiez la fonction wiener afin qu'elle utilise le spectre de l'image dégradée à place de $\Lambda^*\omega^2$.

```

670 def new_wiener(input_image, kernel, regularization=0):
671     input_image = np.array(input_image)
672     kernel = np.array(kernel)
673
674     (image_height, image_width) = input_image.shape
675     (kernel_height, kernel_width) = kernel.shape
676
677     extended_kernel = np.zeros((image_height, image_width))
678     extended_kernel[:kernel_height, :kernel_width] = kernel
679
680     half_width = image_width / 2
681     half_height = image_height / 2
682
683     frequency_x = np.concatenate((np.arange(0, half_width + 0.99), np.arange(-half_width + 1, -0.1)))
684     frequency_y = np.concatenate((np.arange(0, half_height + 0.99), np.arange(-half_height + 1, -0.1)))
685
686     frequency_x = np.ones((image_height, 1)) @ frequency_x.reshape((1, -1))
687     frequency_y = frequency_y.reshape((-1, 1)) @ np.ones((1, image_width))
688
689     frequency_x = frequency_x / image_width
690     frequency_y = frequency_y / image_height
691
692     w_squared = frequency_x**2 + frequency_y**2
693     w = np.sqrt(w_squared)
694
695     float_image = np.float32(input_image.copy())
696
697     shifted_spectrum = np.fft.fftshift(abs(np.fft.fft2(float_image)))
698     a = shifted_spectrum**2
699     b = input_image.var() * input_image.size
700     x = b / a
701
702     g = np.fft.fft2(input_image)
703
704     k = np.fft.fft2(extended_kernel)
705
706     filter_multiplier = np.conj(k) / (abs(k)**2 + x)
707
708     filtered_spectrum = g * filter_multiplier
709
710     mm = np.zeros((image_height, image_width))
711     y_shift = int(np.round(kernel_height / 2 - 0.5))
712     x_shift = int(np.round(kernel_width / 2 - 0.5))
713     mm[y_shift, x_shift] = 1
714     output_image = np.real(np.fft.ifft2(filtered_spectrum * (np.fft.fft2(mm))))
715
716     return output_image

```