TP02 IMA

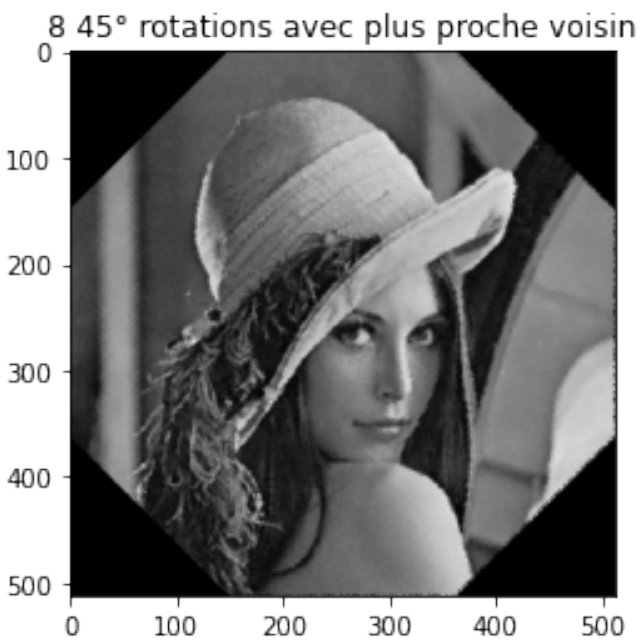
Luiz Augusto Facury de Souza

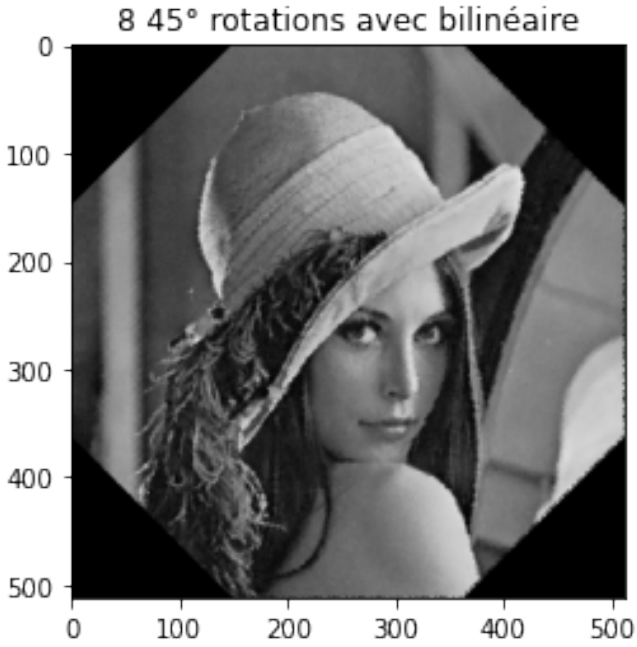
October 3, 2022

1 Préliminaires : utilisation de python

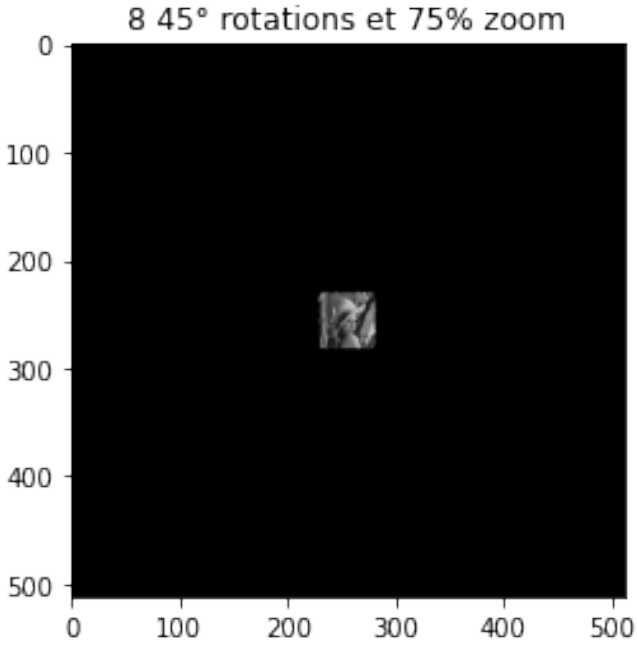
2 Transformation géométrique

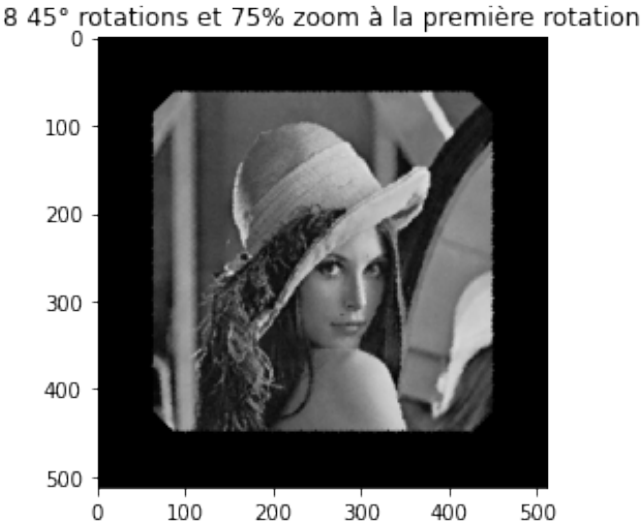
- La méthode à plus proche voisin utilise le même pixel que le plus proche voisin du nouveau pixel inconnu, tandis que l'interpolation bilinéaire utilise une moyenne pondérée des centres des quatre cellules les plus proches.
- Sur une image qui aurait subi huit rotations de 45 degrés, la méthode à plus proche voisin déforme la qualité de l'image, tandis que la methode bilinéaire rend l'image plus nette et lise après les rotations, avec un effet semblable à un filtre passe-bas.





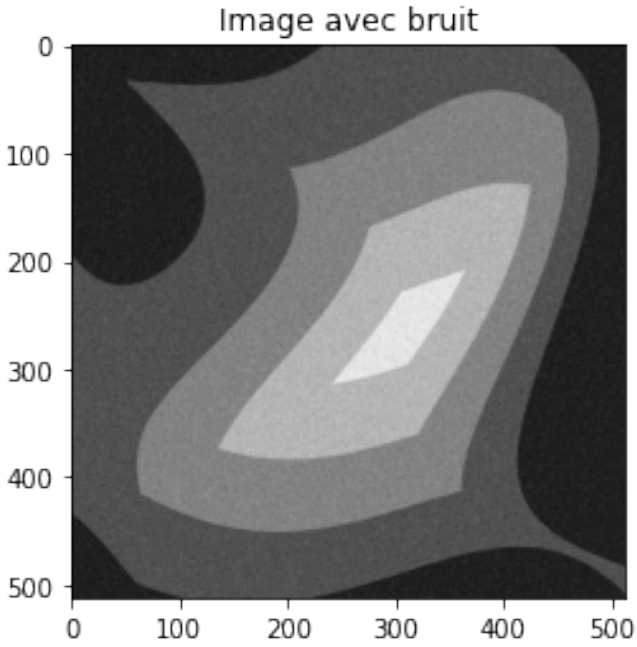
- En la rotation avec un facteur de zoom inférieur à 1, la taille de l'image sera réduite après chaque rotation. Si cet effet n'est pas souhaitable, il est nécessaire d'appliquer la fonction zoom une seule fois entre les rotations.

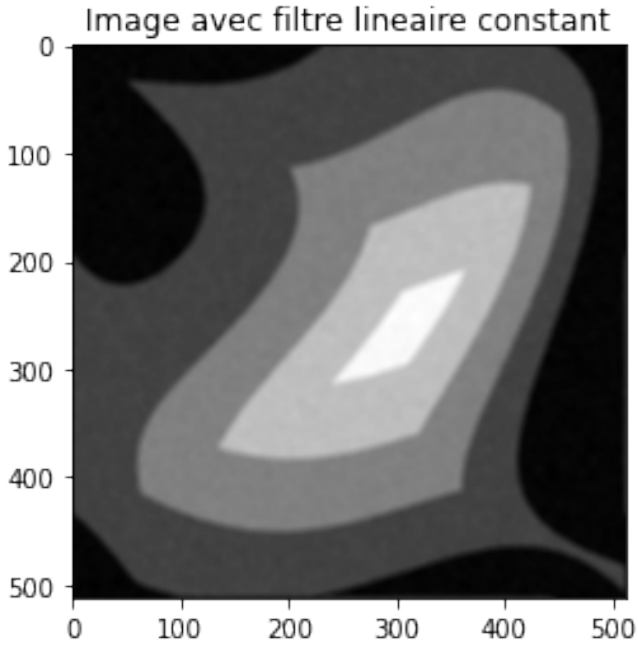


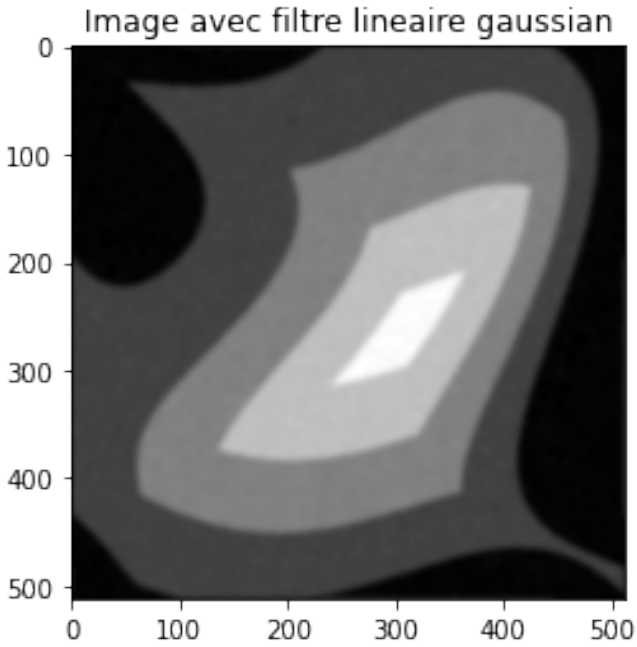


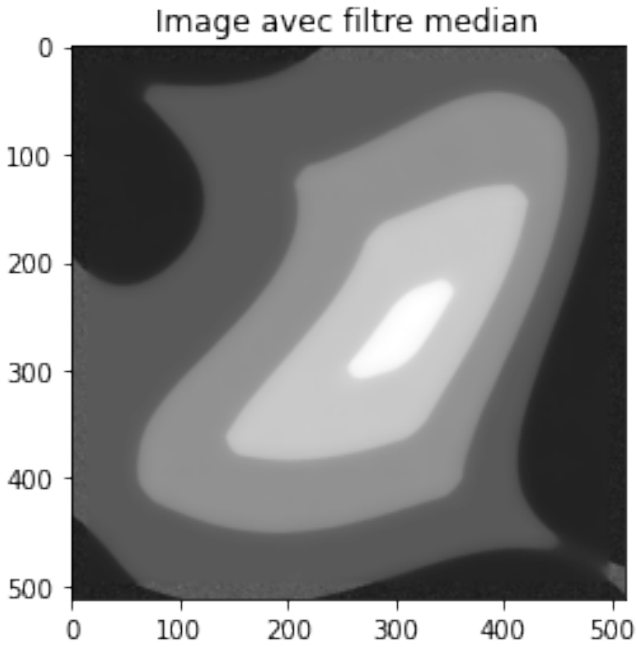
3 Filtrage liéaire et médian

- La taille de la matrice renvoyée par la fonction dépend des paramètres sigma passés à la fonction. Lorsque sigma est plus grand, la variance et la taille de la matrice sont plus grandes.
- Dans une image simple, les couleurs sont pratiquement homogènes, donc la variance est due au bruit résiduel. De cette façon, il est possible d'évaluer la quantité de bruit.



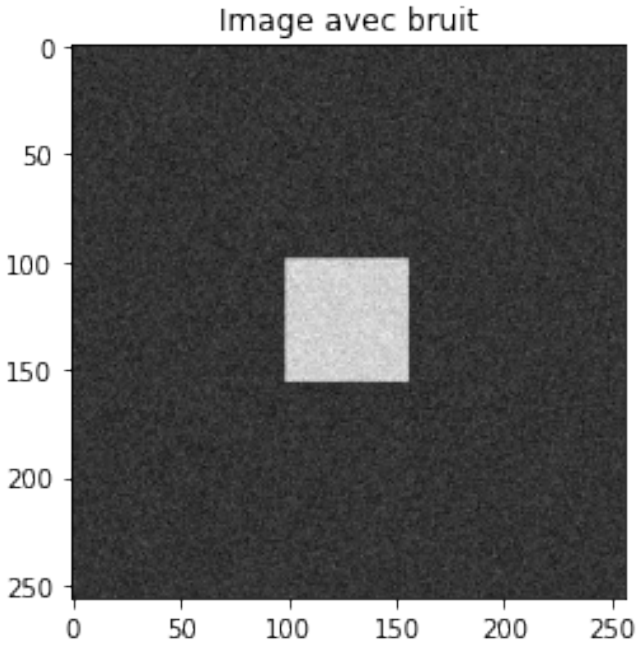


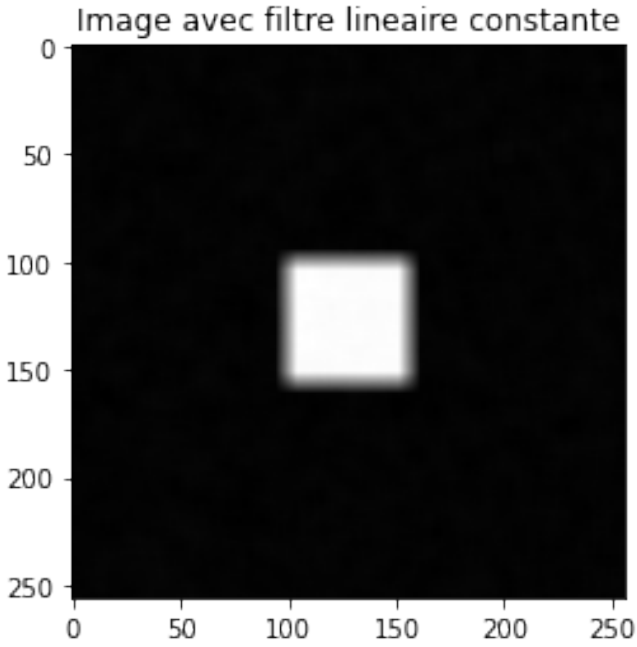


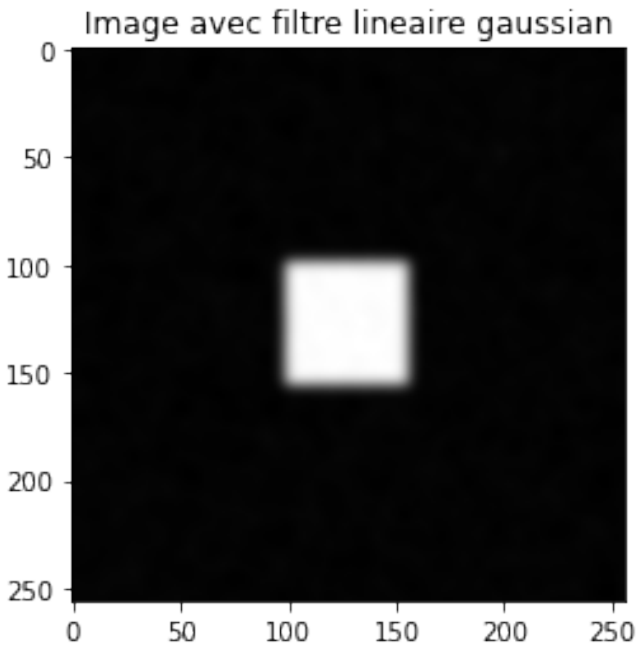


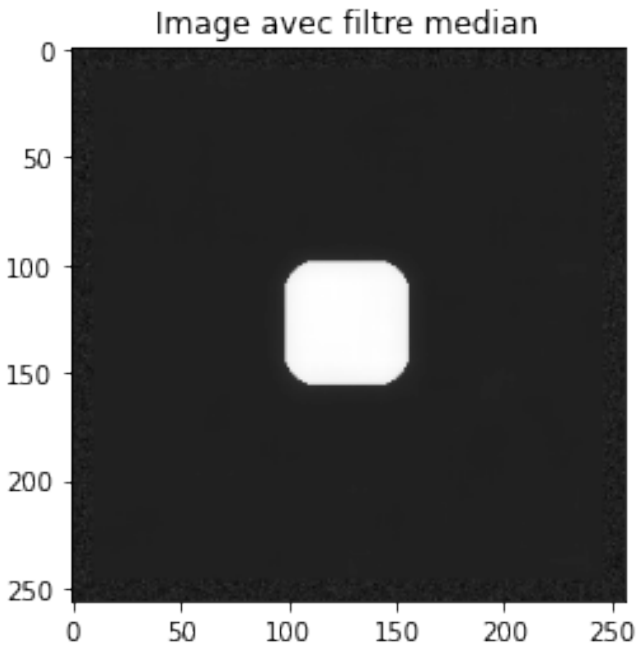
- D'après les images générées, il apparaît que les filtres linéaires éliminent le bruit des régions homogènes, laissant l'image avec un aspect plus lisse et préservant les formats des régions dans lesquelles le gradient est élevé, c'est-à-dire qu'elles sont entourées de pixels d'intensité différente. Un tel effet est donné par le fait que la convolution prend en compte des pixels différents du standard.

  Le filtre médian, en revanche, réduit presque complètement l'effet de bruit de manière beaucoup plus naturelle, cependant, les informations des bords sont perdues, ce qui endommage considérablement l'image, comme vous pouvez le voir sur la figure ci-dessus, où la transition entre une couleur et une autre est arrondie.









- Pour les dernières figures, on constate que, là encore, les filtres linéaires brouillent et lissent l'image, en préservant le bord du carré, tandis que le filtre médian parvient à supprimer le bruit, mais perd l'information sur les bords.

4 Restauration

- A l'aide des fonctions en question, on constate que l'image d'origine est récupérée après passage dans le filtre. Cependant, l'ajout de bruit à l'image rend beaucoup plus difficile la récupération de l'original, ne renvoyant que plus de bruit.
- La convolution a été faite, probablement en forme de T, car il y avait un point blanc dans le coin supérieur droit de l'image, qui s'est transformé en T.
- En contrôlant λ, nous pouvons voir qu'il est possible de modifier la force du filtre à travers lequel l'image est passée, afin de la restituer de la meilleure façon possible.

5 Applications

5.1 Comparaison filtrage linéaire et médian

```
im = skio.imread('carre_orig.tif')
aux, store= 0,0
b = var_image(noise(im,5), 0, 0, 25, 25)
m = var_image(median_filter(noise(im,5), r=4, 0, 0, 25, 25)
for i in range(1,15):
    buffer = var_image(filtre_lineaire(noise(im, 5), get_cst_ker(i), 0, 0, 25, 25)
if (buffer < m) and aux > m:
    store = i
```

```
aux = buffer
```

Avec le code utilisé ci-dessus, il a été possible de vérifier que le meilleur résultat était obtenu avec un paramètre de taille 3.

5.2 Calcul théorique du paramètre de restauration

La modification de code suivante entraîne ce qui a été demandé:

```
def wiener(im,K,lamb=0):
"""effectue un filtrage de wiener de l'image im par le filtre K.
   lamb=0 donne le filtre inverse
   on rappelle que le filtre de Wiener est une tentaive d'inversion du noyau K
   avec une regularisation qui permet de ne pas trop augmenter le bruit.
   """
fft2=np.fft.fft2
ifft2=np.fft.ifft2
(ty,tx)=im.shape
(yK,xK)=K.shape
KK=np.zeros((ty,tx))
KK[:yK,:xK]=K
x2=tx/2
y2=ty/2

fX=np.concatenate((np.arange(0,x2+0.99),np.arange(-x2+1,-0.1)))
fY=np.concatenate((np.arange(0,y2+0.99),np.arange(-y2+1,-0.1)))
fX=np.ones((ty,1))@fX.reshape((1,-1))
fY=fY.reshape((-1,1))@np.ones((1,tx))
fX=fX/tx
fY=fY/ty

#tranformee de Fourier de l'image degradeee
g=fft2(im)
#transformee de Fourier du noyau
k=fft2(KK)

s = g**2
b = len(im) * im.var()

#fonction de mutilplication
mul=np.conj(k)/(abs(k)**2+ b/s)
#filtrage de wiener
fout=g*mul

# on effectue une translation pour une raison technique
mm=np.zeros((ty,tx))
y2=int(np.round(yK/2-0.5))
```

```
x2=int(np.round(xK/2-0.5))
mm[y2,x2]=1
out=np.real(ifft2(fout*(fft2(mm))))
return out
```