Travaux Pratiques - IMA201 Filtrage et Restauration

1- Préliminaires : utilisation de python

2- Transformation géométrique

Utiliser la fonction (rotation) pour transformer une image de votre choix.

```
im = skio.imread('images/lena.tif')
theta = 45
im_rp = rotation(im, theta, clip = False, ech = 0)
im_rb = rotation(im, theta, clip = False, ech = 1)
viewimage(im)
viewimage(im_rp)
viewimage(im_rb)
```

Image originale

Image après rotation (plus proche voisin)

Image après rotation (bilinéaire)

Quelle différence y-a-t-il entre la méthode à plus proche voisin et la méthode bilinéaire ?

On remarque dans le cas d'une rotation effectuée en utilisant la méthode à plus proche voisin que l'image résultante semble pixelisée et présente des bords rugueux. Par contre pour le cas de la méthode bilinéaire l'image résultante est de meilleure qualité car les transitions entre les pixels sont lisses.

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

```
im_8_rp = im_rp
im_8_rb = im_rb
for i in range(7):
    im_8_rp = rotation(im_8_rp, theta, ech = 0)
    im_8_rb = rotation(im_8_rb, theta, ech = 1)
viewimage(im_8_rp)
viewimage(im_8_rb)
```

Image après 8 rotations (plus proche voisin)

Image après 8 rotations (bilinéaire)

Lorsqu'une image subit huit rotations de 45 degrés (ou 360 degrés au total), cela signifie que l'image a été tournée complètement donc les pixels de l'image résultante seront alignés avec les pixels d'origine. Mais la méthode utilisée pour la rotation (bilinéaire ou plus proche voisin) aura un impact significatif sur le résultat final qu'une seule rotation (bilinéaire : image floue, plus proche voisin : image pixélisée).

Que constatez-vous si vous appliquez la rotation avec un facteur de zoom inférieur à 1 (par exemple 1/2) ?

```
alpha = 0.5
im_zr = rotation(im, theta, alpha,ech = 1, clip = False)
viewimage(im_zr)
```

Image après rotation avec un facteur de zoom 1/2 (bilinéaire)

Lorsqu'on effectue une rotation avec un facteur de zoom inférieur à 1, on observe la présence d'aliasing (exemple : des traits dans le chapeau de Lena), ce qui génère des artefacts visuels non souhaités. Pour minimiser cet effet, il a été nécessaire d'appliquer un filtre passe-bas avant de procéder au sous-échantillonnage.

3- Filtrage linéaire et médian

Dans cette section on se propose de comparer les filtrages linéaire et m´médian. Pour appliquer un filtrage linéaire il faut d'abord générer un noyau. Le filtrage linéaire consiste à convoluer le noyau avec l'image. Vous pouvez utiliser les commandes « get_gau_ker » et « get_cst_ker » pour générer un noyau gaussien ou constant. Pour appliquer ces noyaux il faut utiliser la commande filtre linéaire. La commande « median_filter » applique un filtre médian.

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

```
def get_gau_ker(s):
    ss=int(max(3,2*np.round(2.5*s)+1))
    ms=(ss-1)//2
    X=np.arange(-ms,ms+0.99)
    y=np.exp(-X**2/2/s**2)
    out=y.reshape((ss,1))@y.reshape((1,ss))
    out=out/out.sum()
    return out
```

Le paramètre "s" désigne la valeur de l'écart-type de la distribution gaussienne utilisée pour créer le noyau gaussien.

"ss" représente la dimension souhaitée du noyau gaussien en pixels. Elle est calculée en fonction de "s". Pour garantir une taille minimale de 3x3, la formule suivante est employée « int(max(3, 2 * arrondi(2,5 * s) + 1)) ».

"ms" correspond à l'indice central au sein de la matrice du noyau.

"X" contient une séquence d'entiers qui représente les positions horizontales des pixels dans le noyau.

"y" contient les valeurs du noyau gaussien associées à chaque position "X".

"out" fait référence à la matrice résultante du noyau gaussien.

Après avoir ajouté du bruit à une image simple telle que « pyramide.tif » ou carre « orig.tif » et avoir filtré le résultat avec des filtres linéaires, expliquez comment on peut évaluer (sur des images aussi simples) la quantité de bruit résiduel (la commande « var_image » donne la variance d'une partie d'une image).

```
imp = skio.imread('images/pyramide.tif')
viewimage(imp)
s = 2
gau_ker = get_gau_ker(s)
imp_b = noise(imp,10)
viewimage(imp_b)
imp_gb = filtre_lineaire(imp_b, gau_ker)
viewimage(imp_gb)
```

Image originale

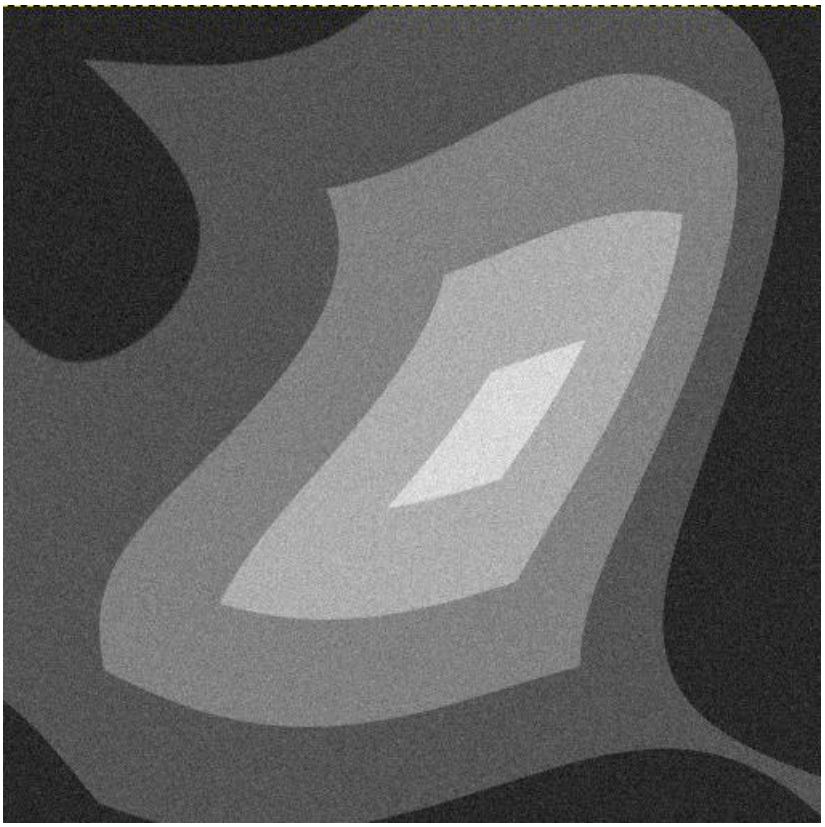


Image bruitée par un bruit gaussien.

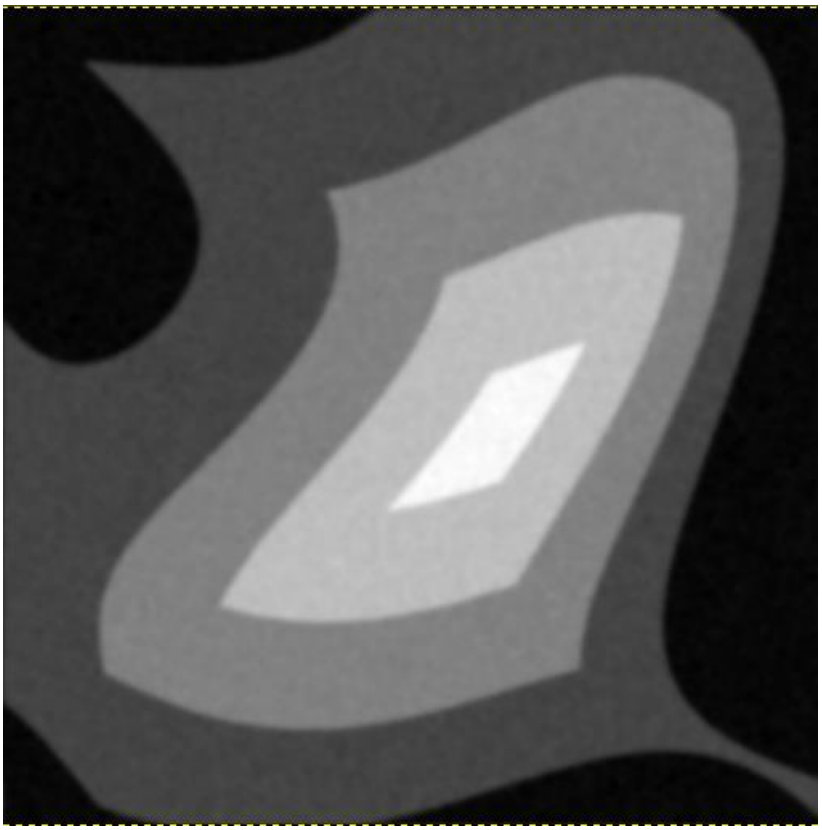


Image bruitée et filtrée par un filtre linéaire à noyau gaussien.

```
n = imp.shape[0]
x0 = 0; y0 = n//2; x1 = n//10 ; y1 =y0 + n//10
v = var_image(imp, x0, y0, x1, y1)
v_b = var_image(imp_b, x0, y0, x1, y1)
v_gb = var_image(imp_gb, x0, y0, x1, y1)
(v,v_b, v_gb)
```

```
(0.0, 100.23787235411565, 11.500014440042587)
```

Pour évaluer le bruit résiduel, on commence par sélectionner une zone d'intensité de gris uniforme dans l'image d'origine. Dans notre cas, étant donné que la taille de l'image est de n*n pixels avec n = 512, nous choisissons x dans l'intervalle [0, n/10] et y dans l'intervalle [n/2, n/2 + n/10]. Nous constatons que la variance de cette zone est nulle.

En revanche, pour la même zone dans l'image bruitée, nous observons que la variance est presque égale à la variance du bruit (variance (zone bruitée) = 100 = (écart type (bruit)) ²). Après l'application du filtrage, nous notons que la variance du bruit résiduel est presque égale à la racine carrée de celle du bruit initial (var (bruit résiduel) = 11.5).

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

```
imp_med =  median_filter(imp_b)
viewimage(imp_med)
```

(Le médian est calculé sur un carré de cote 3)

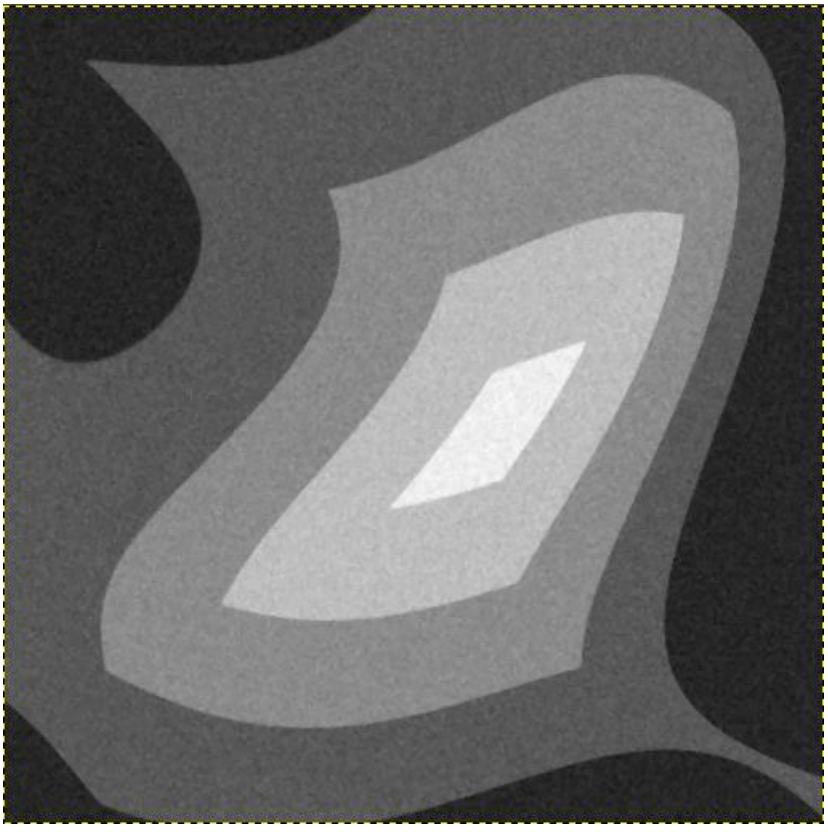


Image bruitée et filtrée par un filtre médian

```
v_med = var_image(imp_med, x0, y0, x1, y1)
(v,v_b, v_med)
```

```
(0.0, 100.23787235411565, 19.144736695559654)
```

On remarque que la variance du bruit résiduel après le filtrage médian (variance_median (bruit résiduel) = 19.145) est plus élevée que celle après le filtrage linéaire à noyau gaussien (variance_linéaire (bruit résiduel) = 11.5). Par conséquent, on peut conclure que le filtrage médian n'est pas aussi efficace pour réduire le bruit gaussien dans les zones uniformes. En revanche, le filtrage médian a mieux préservé les contours de l'image que le filtrage linéaire, qui a tendance à lisser les contours.

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

```
imp_pi = skio.imread('images/pyra-impulse.tif')
viewimage(imp_pi)
imp_pi_lg = filtre_lineaire(imp_pi, gau_ker)
viewimage(imp_pi_lg)
imp_pi_med =  median_filter(imp_pi)
viewimage(imp_pi_med)
v_pi = var_image(imp_pi, x0, y0, x1, y1)
v_pi_lg = var_image(imp_pi_lg, x0, y0, x1, y1)
v_pi_med = var_image(imp_pi_med, x0, y0, x1, y1)
(v_pi,v_pi_lg, v_pi_med)
```

```
(88.52625725968102, 17.555726557944652, 4.654129401762894)
```

Image bruitée par un bruit impulsionnel.

Image bruitée et filtrée par un filtre linéaire à noyau gaussien.

Image bruitée et filtrée par un filtre médian.

On constate que le filtrage linéaire, comme le filtrage gaussien par exemple, est moins efficace pour traiter le bruit impulsionnel que le bruit gaussien. Il a tendance à réduire légèrement le bruit en lissant les valeurs des pixels, mais lorsqu'il est appliqué à une image bruitée par un bruit impulsionnel, il adoucit les bords et les contours, ce qui rend l'image moins nette.

En revanche, le filtrage médian fonctionne en remplaçant la valeur de chaque pixel par la médiane des valeurs des pixels voisins dans une fenêtre donnée, ce qui se révèle très efficace pour éliminer le bruit impulsionnel tout en préservant les contours et les détails de l'image. Cela signifie que l'image résultante conserve une meilleure netteté.

Expliquer la différence de comportement entre filtrage linéaire et médian sur le point lumineux situé en haut à droite de l'image « carre_orig.tif ».

```
imc_pi = skio.imread('images/carre_orig.tif')
viewimage(imc_pi)
s = 1
gau_ker = get_gau_ker(s)
imc_pi_lg = filtre_lineaire(imc_pi, gau_ker)
viewimage(imc_pi_lg)
imc_pi_med =  median_filter(imc_pi)
viewimage(imc_pi_med)
```

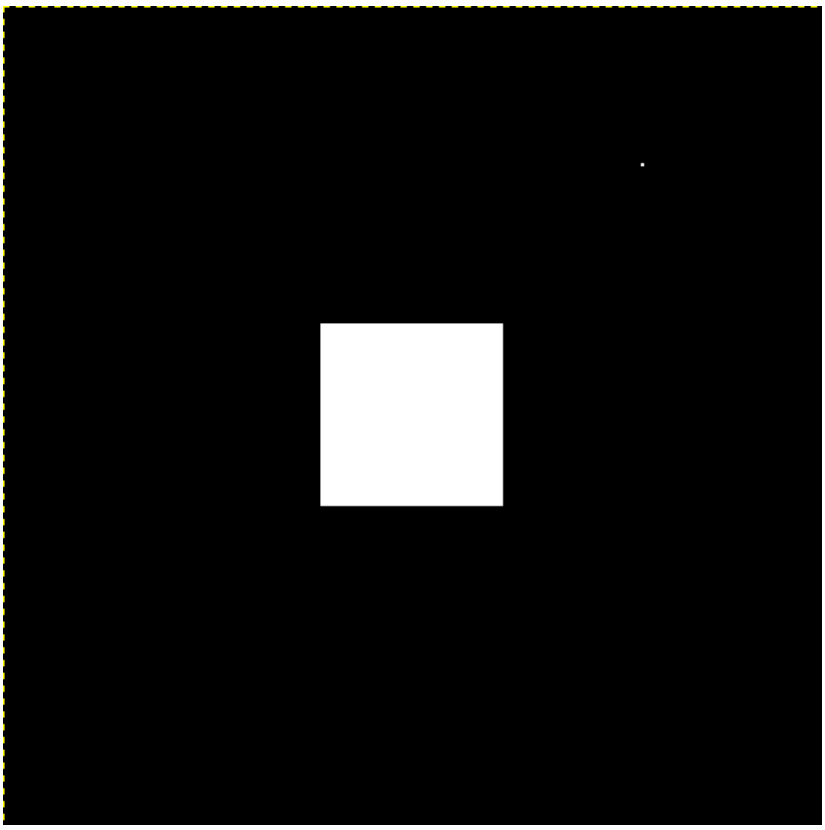


Image originale (zoomée x2)

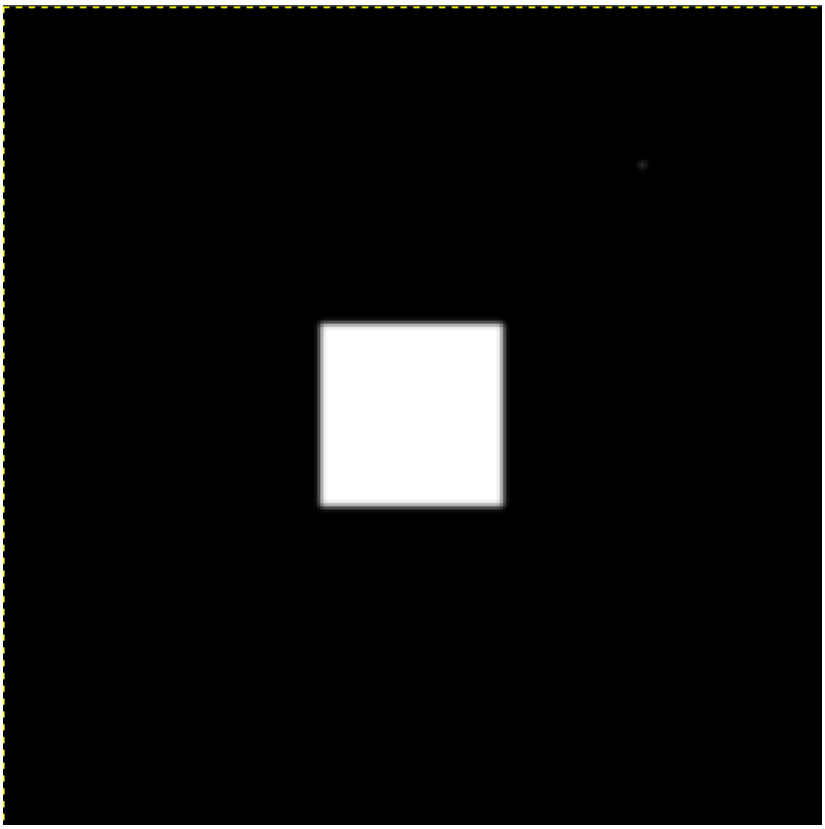


Image filtrée par un filtre linéaire à noyau gaussien (zoomée x2)

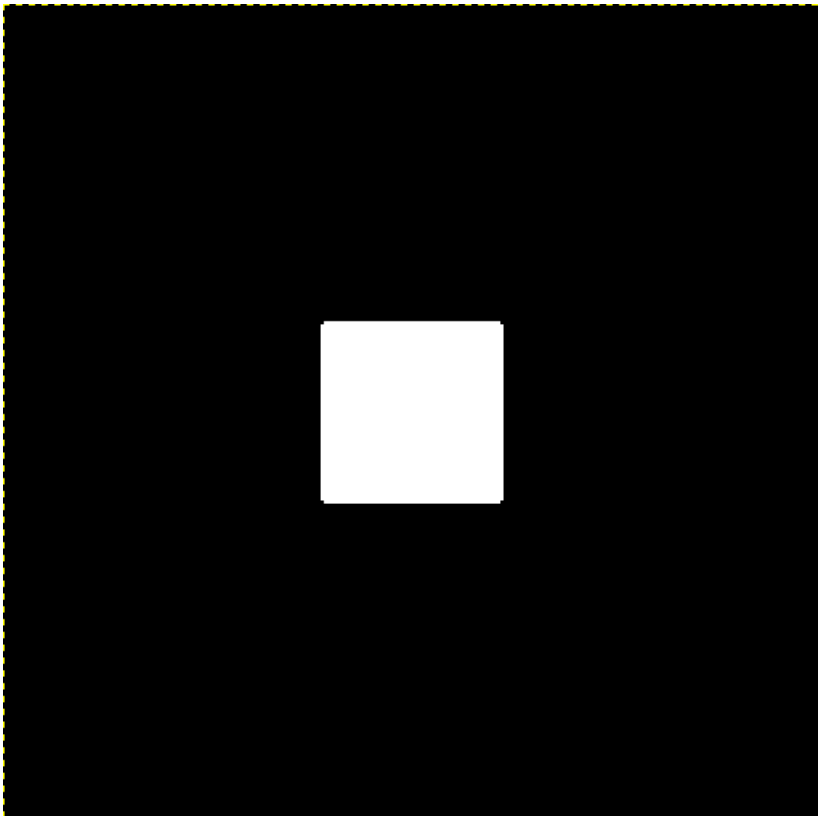


Image filtrée par un filtre médian (zoomée x2)

Si l'on applique un filtre linéaire sur le point lumineux, on obtient la réponse impulsionnelle du filtre, car le point lumineux joue le rôle d'une impulsion de Dirac. En revanche, lorsque la taille du filtre médian est supérieure à 1 pixel, la valeur médiane sera toujours prise par le pixel noir, ce qui a pour effet de faire disparaître le point lumineux.

4- Restauration

Appliquer un filtre linéaire à une image puis utilisez la fonction « filtre_inverse». Que constatez-vous ?

```
im = skio.imread('images/lena.tif')
viewimage(im)
s = 2
gau_ker = get_gau_ker(s)
im_f = filtre_lineaire(im, gau_ker)
viewimage(im_f)
im_1 = filtre_inverse(im_f,gau_ker)
viewimage(im_1)
```

Image originale

Image filtrée par un filtre linéaire gaussien

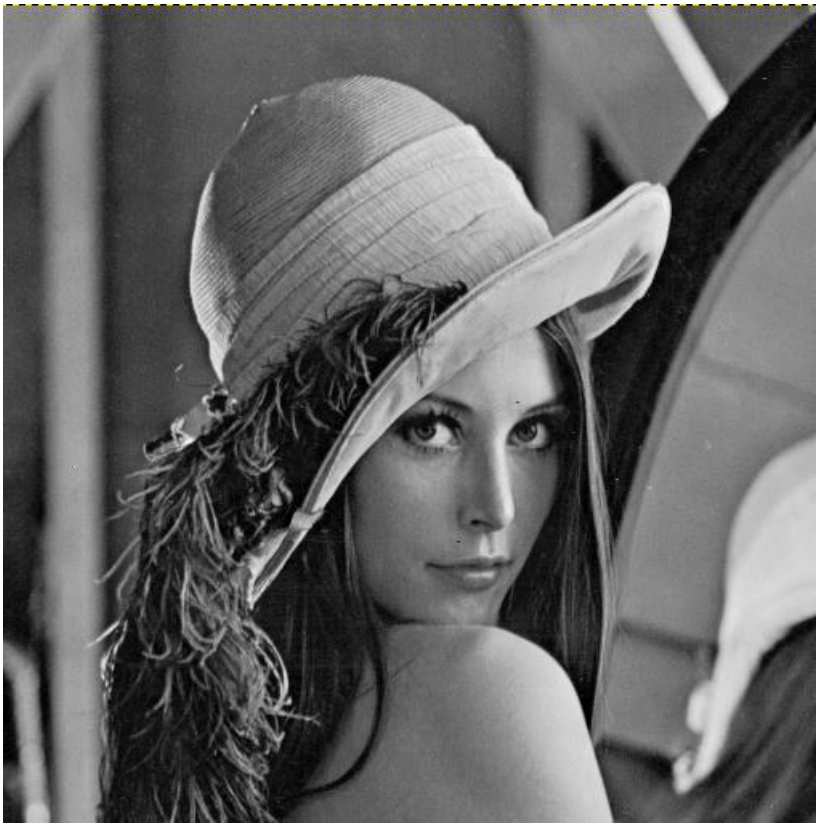


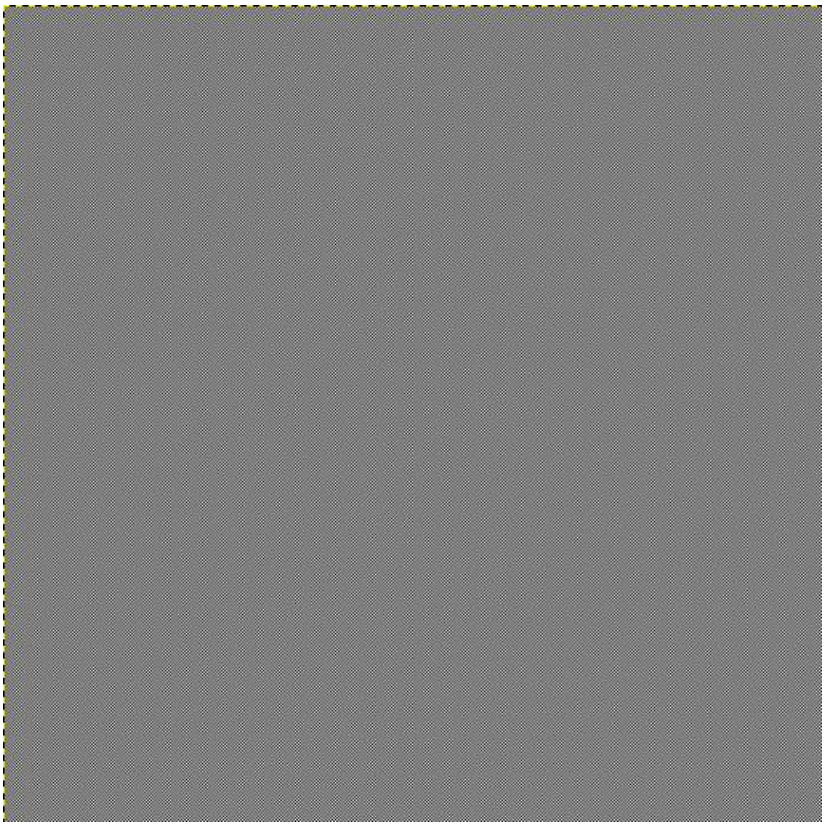
Image après transformation inverse

On constate que l'image la transformation inverse du filtrage linéaire a produit une image identique à celle d'origine.

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

```
im_fb = noise(im_f, 0.1)
viewimage(im_fb)
im_2 = filtre_inverse(im_fb,gau_ker)
viewimage(im_2)
```

Image filtrée puis bruitée avec un bruit d'écart type = 0.1



Résultat de la transformation inverse

On constate que si l'on applique la même transformation inverse à l'image filtrée après l'ajout d'un léger bruit, qui parfois ne semble pas remarquable sur l'image (écart type = 0,1), cela amplifie considérablement ce bruit et entraîne une perte de l'image d'origine.

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

```
imcf = skio.imread('images/carre_flou.tif')
viewimage(imcf)
ker = imcf[48:51,198:201]
viewimage(ker)
```

```
array([[0. , 0. , 0.2],
       [0.2, 0.2, 0.2],
       [0. , 0. , 0.2]])
```

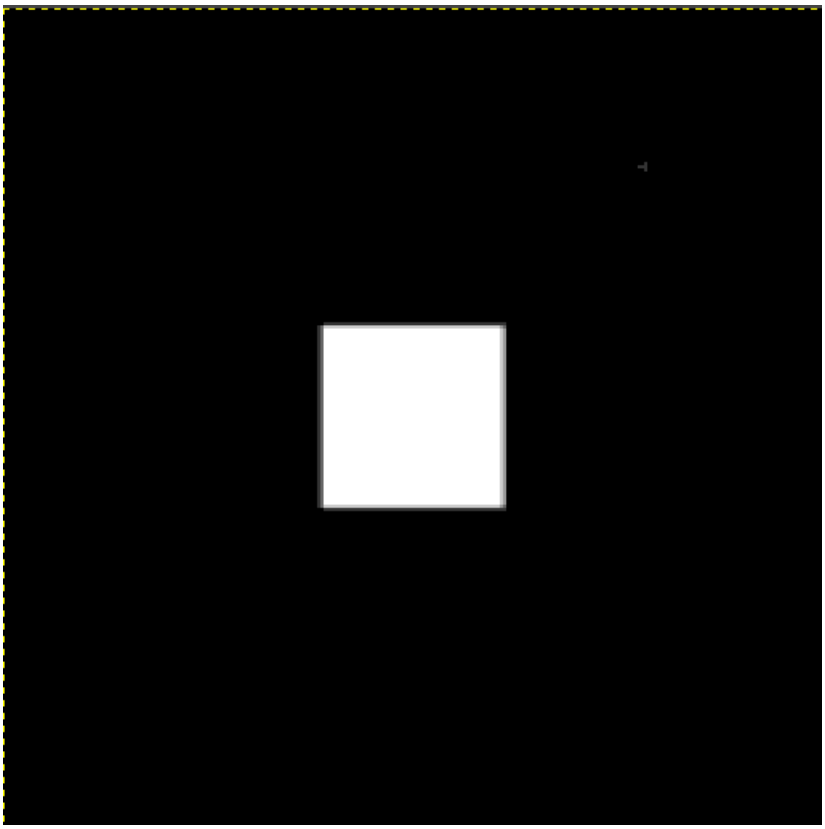
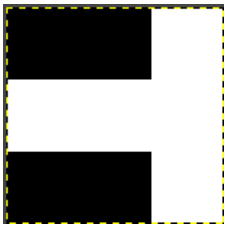


Image carre_flou.tif



Noyau du filtre zoomé 4500%

Puisque l'image d'origine contient une impulsion de Dirac discrète, la réponse impulsionnelle du filtre se trouvera également au même emplacement dans l'image filtrée, et il suffit de l'extraire de l'image.

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

```
imcf_b = noise(imcf, 10)
viewimage(imcf_b)
```

```
imcf_w0 = wiener(imcf_b,ker,lamb = 0)
viewimage(imcf_w0)
```

```
imcf_w1 = wiener(imcf_b,ker,lamb = 1)
viewimage(imcf_w1)
```

```
imcf_w10 = wiener(imcf_b,ker,lamb = 10)
viewimage(imcf_w10)
```

```
imcf_w100 = wiener(imcf_b,ker,lamb = 100)
viewimage(imcf_w100)
```

```
imcf_w1000 = wiener(imcf_b,ker,lamb = 1000)
viewimage(imcf_w1000)
```

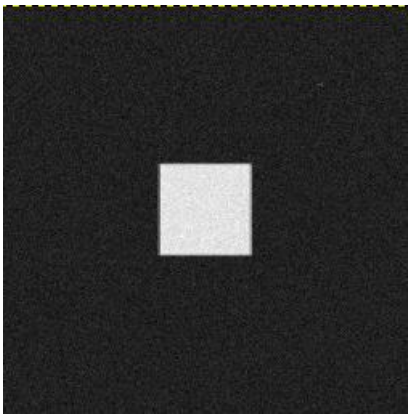


Image carre_flou.tif bruitée

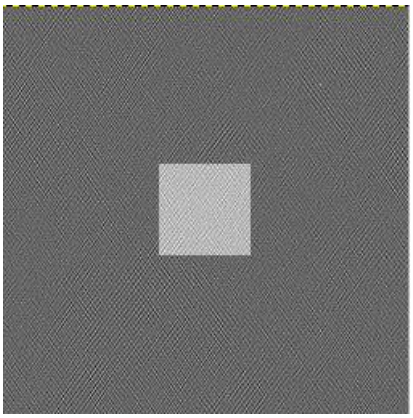


Image restaurée avec filtre de Wiener (λ = 0)

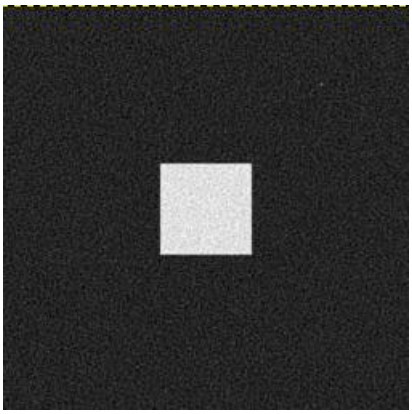


Image restaurée avec filtre de Wiener (λ = 1)

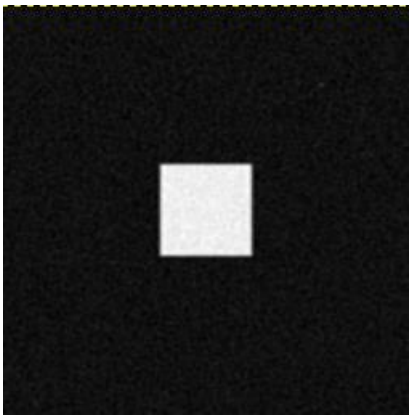


Image restaurée avec filtre de Wiener (λ = 10)

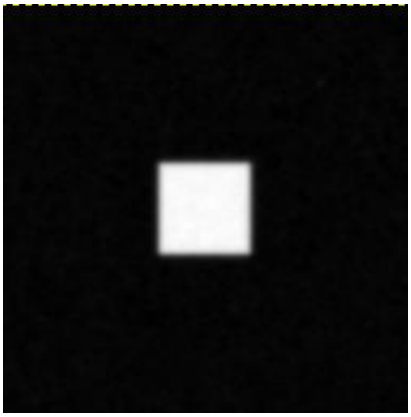


Image restaurée avec filtre de Wiener (λ = 100)

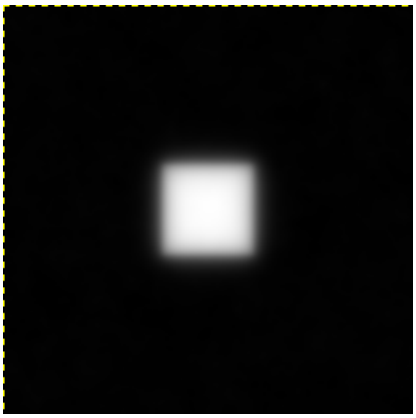


Image restaurée avec filtre de Wiener (lambda = 1000)

En augmentant le paramètre lambda du filtre de Wiener, on constate une amélioration initiale du résultat (cas de λ = 0, 1). On est donc dans la phase de sous-estimation du bruit. Cependant, une fois qu'on a avons atteint un certain point (λ ~10), augmenter davantage ce paramètre entraîne une détérioration du résultat. On sera donc dans la phase de surestimation du bruit (λ = 100, 1000).

5- Applications

a. 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. (Explicitez l'algorithme utilisé).

```
imc_pi = skio.imread('images/carre_orig.tif')
imc_bg = noise(imc_pi,5)
(n,m) = imc_pi.shape
```

```
l = n//5
p = m//5
min_eps = 256 * 256
ker_c = 0
x0 = var_image(imc_pi,0,0,l,p)
x = var_image(median_filter(imc_bg,typ=2,r=4),0,0,l,p)
for i in range(0,100):
    y=var_image(filtre_lineaire(imc_bg,get_cst_ker(i)),0,0,l,p)
    if(np.abs(y-x)<min_eps):
        ker_c = i
        min_eps = np.abs(y-x)
(x0,ker_c,min_eps)
```

```
(0.0, 3, 1.5676759528928903)
```

```
imc_c = median_filter(imc_bg,typ=2,r=4)
viewimage(imc_c)
imc_l = filtre_lineaire(imc_bg,get_cst_ker(3))
viewimage(imc_l)
```

On a sélectionné une région uniforme dans l'image d'origine (variance = 0), puis on a effectué une série d'itérations en modifiant la taille du noyau du filtre constant, cherchant ainsi la taille de noyau qui minimise la différence de variance par rapport au filtre médian. On a constaté que le minimum est atteint avec une taille de noyau de 3, où la différence minimale de variance est de 1.567.

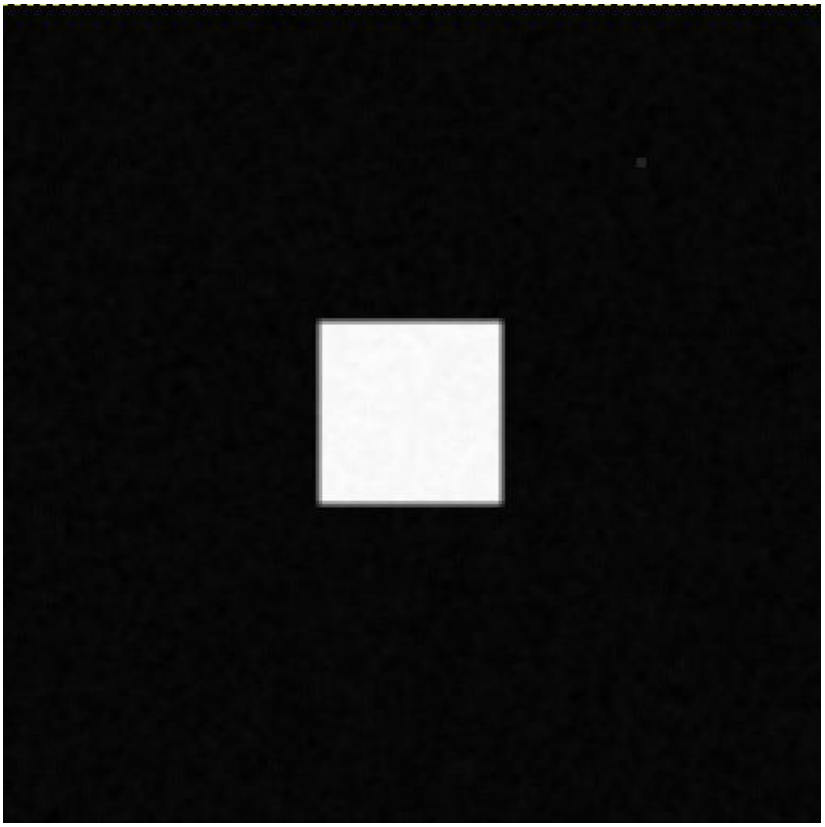


Image bruitée filtrée par un filtre linéaire constant

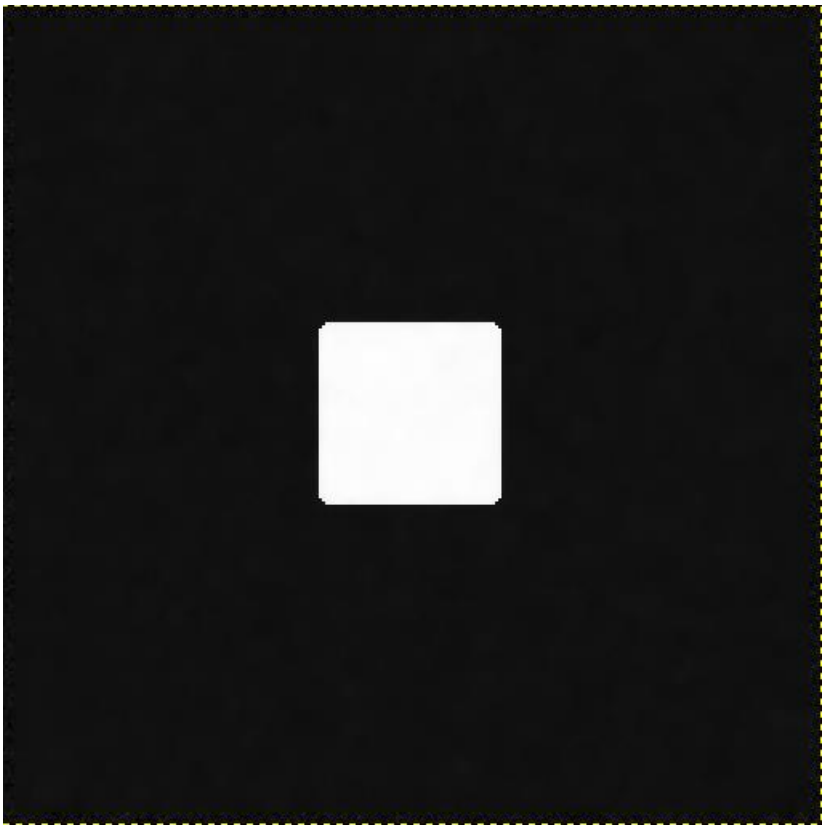


Image bruitée filtrée par un filtre médian

b. Calcul théorique du paramètre de restauration

Ici on travaille à nouveau sur l'image « carre_flou.tif » que l'on bruite et restaure par « wiener ». Modifiez la fonction « wiener » afin qu'elle utilise le spectre de l'image dégradée à place de $\lambda\omega^2$.

```
def wiener2(im,K,var):
    """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

    w2=fX**2+fY**2
    w=w2**0.5
```

```
    #tranformee de Fourier de l'image degradeee
    g=fft2(im)
    #transformee de Fourier du noyau
    k=fft2(KK)
    #nouveau quotien introduit
    q = (ty*tx * var) / (abs(g)**2 + 0.0001)
    #fonction de mutilplication
    mul=np.conj(k)/(abs(k)**2+ q)
    #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
```

```
imcf = skio.imread('images/carre_flou.tif')
#viewimage(imcf)
ker = imcf[48:51,198:201]/255
imcw = wiener2(imcf,ker,imcf.var())
viewimage(imcw)
```

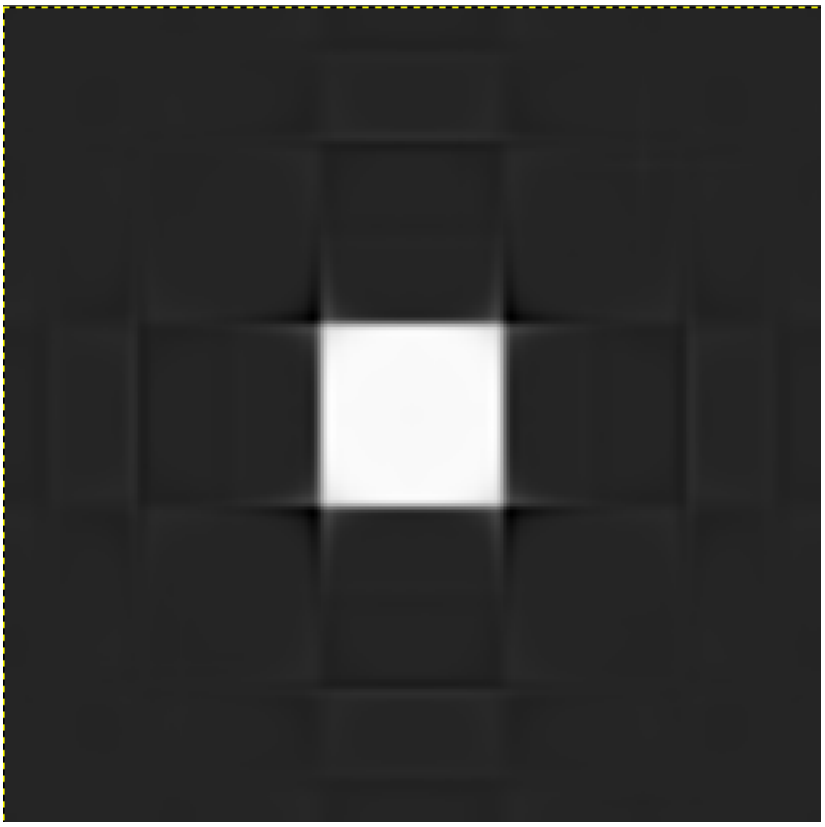


Image résultat de filtre de Wiener