TP3. Traitement du signal.

UE apprentissage statistique et réseaux de neurones.

Master mathématiques appliquées, statistique - parcours data science.

Frédéric Richard, AMU, 2025.

Antoine Legendre

Exercice 1. Propriétés des coefficients de Fourier.

Rappel de cours.

Soit $d > 0$ un entier représentant une dimension.

On considère une application g dans $L^2_P([0, 2\pi[^d)$, c'est à dire une application définie sur $\mathbb{R}^d$ à valeurs dans $\mathbb{C}$, 2π-périodique, de carré intégrable sur $[0, 2\pi[^{\mathrm{d}}$.

Pour $n \in \mathbb{Z}^d$, on note $c_n(g)$, le **coefficient de Fourier** de g, défini par

$$c_n(g) = \frac{1}{(2\pi)^d} \int_{[0,2\pi]^d} g(s) e^{-i\langle n,s\rangle} ds.$$

Dans $L^2_P([0, 2\pi[^d)$, on a la **série de Fourier**

$$g(y) = \sum_{n \in \mathbb{Z}^d} c_n(g)\, e^{i\langle n,y\rangle}.$$

Soieng g et h dans $L^2_P([0, 2\pi[^d)$. On note $g \circledast h$ le produit de convolution périodique de g et h, application définie, pour tout $y \in \mathbb{R}$, par

$$g \circledast h(y) = \int_{[0,2\pi[^d} g(s) h(y-s) ds = \int_{[0,2\pi[^d} g(y-s) h(s) ds.$$

Partie 1.

On fixe $d = 1$.

1. Soit $g \in L^2_P([0, 2\pi[$. On suppose que g est dérivable sur $\mathbb{R}$ et que $g' \in L^2_P([0, 2\pi[)$. Montrer que, pour tout $n \in \mathbb{Z}$,

$$c_n(g') = inc_n(g).$$

2. On suppose à présent que g est deux fois dérivable sur $\mathbb{R}$ et que $g'' \in L^2_P([0, 2\pi[)$. Montrer que, pour tout $n \in \mathbb{Z}$,

$$c_n(g'') = -n^2 c_n(g).$$

3. Soient f et g dans $L^2_P([0, 2\pi[)$. Etablir que, pour tout $n \in \mathbb{Z}$,

$$c_n(g * h) = c_n(g)\, c_n(h).$$

Exercice 1 :

Partie 1 :

1 : Par définition, le coefficient de Fourier d'une fonction g est donné par :

$$c_n(g) = \int_0^{2\pi} g(s)e^{-ins}ds$$

En dérivant sous le signe intégral :

$$c_n(g') = \int_0^{2\pi} g'(s)e^{-ins}ds$$

En intégrant par parties, avec $u = e^{-ins}$ et $dv = g'(s)ds$, on a :

$$du = -ine^{-ins}ds, \quad v = g(s)$$

D'où :

$$c_n(g') = \left[g(s)e^{-ins}\right]_0^{2\pi} - \int_0^{2\pi} g(s)(-ine^{-ins})ds$$

Or g est 2π-périodique, donc $g(0) = g(2\pi)$, ce qui annule le premier terme :

$$c_n(g') = in \int_0^{2\pi} g(s)e^{-ins}ds = inc_n(g)$$

Donc : $c_n(g') = inc_n(g)$.

2 : En reprenant le résultat précédent pour la dérivée seconde :

$$c_n(g'') = \int_0^{2\pi} g''(s)e^{-ins}ds$$

En intégrant par parties deux fois :

$$c_n(g'') = \left[g'(s)e^{-ins}\right]_0^{2\pi} - \int_0^{2\pi} g'(s)(-ine^{-ins})ds$$

$$= in\int_0^{2\pi} g'(s)e^{-ins}ds = inc_n(g')$$

En utilisant le résultat du premier point $c_n(g') = inc_n(g)$, on obtient :

$$c_n(g'') = in(inc_n(g)) = -n^2c_n(g)$$

Donc : $c_n(g'') = -n^2c_n(g)$.

3 : Par définition, la convolution périodique de g et h est :

$$(g*h)(y) = \int_0^{2\pi} g(s)h(y-s)ds$$

En prenant la transformée de Fourier :

$$c_n(g*h) = \int_0^{2\pi}\left(\int_0^{2\pi} g(s)h(y-s)ds\right)e^{-iny}dy$$

En inversant l'ordre des intégrales :

$$c_n(g*h) = \int_0^{2\pi} g(s)\left(\int_0^{2\pi} h(y-s)e^{-iny}dy\right)ds$$

En effectuant le changement de variable $u = y - s$, on reconnaît $c_n(h)$:

$$c_n(g*h) = \int_0^{2\pi} g(s)e^{-ins}ds \cdot c_n(h)$$

$$= c_n(g)c_n(h)$$

Donc : $c_n(g*h) = c_n(g)c_n(h)$.

Partie 2.

On fixe $d = 2$.

Soit g dans $L^2_P([0,2\pi[^2)$.

1. On suppose que g est différentiable sur $\mathbb{R}^2$, de dérivées partielles $\partial_j g$ dans $L^2_P([0,2\pi[^2)$. Montrer que, pour $j \in [\![1,2]\!]$ et $n \in \mathbb{Z}^2$,

$$c_n(\partial_j g) = i n_j c_n(g).$$

2. On suppose que g est deux fois différentiable sur $\mathbb{R}^2$ et de dérivées partielles secondes $\partial^2_{jk} g$ dans $L^2_P([0, 2\pi[^2)$. Montrer que, pour $j, k \in [\![1, 2]\!]$ et $n \in \mathbb{Z}^2$,

$$c_n(\partial^2_{jk} g) = -n_j n_k c_n(g).$$

Partie 2 :

1 : Le coefficient de Fourier de g en 2D est donné par :

$$c_n(g) = \int_{[0,2\pi]^2} g(s_1, s_2) e^{-i(n_1 s_1 + n_2 s_2)} ds_1 ds_2$$

Le coefficient de Fourier de $\partial_j g$ est donc :

$$c_n(\partial_j g) = \int_{[0,2\pi]^2} \partial_j g(s_1, s_2) e^{-i(n_1 s_1 + n_2 s_2)} ds_1 ds_2$$

On intègre par parties en prenant :

- $u = e^{-i(n_1 s_1 + n_2 s_2)}$
- $dv = \partial_j g(s_1, s_2) ds_j$
- $du = -i n_j e^{-i(n_1 s_1 + n_2 s_2)} ds_j$
- $v = g(s_1, s_2)$

L'intégration par parties donne :

$$\int u\, dv = \Big[g(s_1, s_2) e^{-i(n_1 s_1 + n_2 s_2)} \Big]_0^{2\pi} - \int v\, du$$

Grâce à la périodicité de g, on a :

$$g(0, s_2) = g(2\pi, s_2) \quad \text{et} \quad g(s_1, 0) = g(s_1, 2\pi)$$

ce qui annule le terme de gauche.

Il reste donc :

$$c_n(\partial_j g) = -\int_{[0,2\pi]^2} g(s_1, s_2) \cdot (-i n_j e^{-i(n_1 s_1 + n_2 s_2)}) ds_1 ds_2$$

$$= i n_j \int_{[0,2\pi]^2} g(s_1, s_2) e^{-i(n_1 s_1 + n_2 s_2)} ds_1 ds_2$$

$$= i n_j c_n(g)$$

Donc : $c_n(\partial_j g) = i n_j c_n(g)$

2 :

$$c_n(\partial^2_{jk} g) = \int_{[0,2\pi]^2} \partial^2_{jk} g(s_1, s_2) e^{-i(n_1 s_1 + n_2 s_2)} ds_1 ds_2$$

Nous appliquons une première intégration par parties sur la variable s_j, en utilisant la périodicité comme précédemment :

$$c_n(\partial^2_{jk} g) = \int_{[0,2\pi]^2} \partial_k \left(\partial_j g(s_1, s_2) e^{-i(n_1 s_1 + n_2 s_2)} \right) ds_1 ds_2$$

D'après le premier résultat :

$$c_n(\partial_j g) = i n_j c_n(g)$$

En dérivant de nouveau, on applique l'intégration par parties sur s_k et, de la même manière :

$$c_n(\partial^2_{jk} g) = i n_k c_n(\partial_j g)$$

En remplaçant $c_n(\partial_j g) = i n_j c_n(g)$, on obtient :

$$c_n(\partial^2_{jk} g) = i n_k (i n_j c_n(g))$$

$$= -n_j n_k c_n(g)$$

Donc : $c_n(\partial^2_{jk} g) = -n_j n_k c_n(g)$

Exercice 2. Mise en oeuvre de filtres linéaires.

Rappel de cours.

Soient d et N dans $\mathbb{N}^*$. On considère deux signaux discrets f et g, définies sur la grille uniforme $[\![0, N-1]\!]^d$. On note $f[n]$ et $g[n]$ les valeurs de ces signaux en une position n de la grille. On étend la définition de ces signaux à $\mathbb{Z}^d$ par périodisation.

On rappelle que

- le produit de convolution périodique discret $h = f \circledast g$ est le signal défini pour tout $n \in \mathbb{Z}^d$ par

$$f \circledast g[n] = \sum_{k \in [0,N-1]^d} f[n-k] g[k].$$

- la transformée de Fourier discrète $\hat{f}$ du signal N-périodique f est le signal défini pour tout $n \in \mathbb{Z}^d$ par

$$\hat{f}\,[n] = \sum_{k \in [0,N-1]^d} f[k] \exp\left(-i \frac{2\pi \langle k, n \rangle}{N}\right).$$

- la transformée de Fourier discrète inverse $f^{\check{}}$ du signal N-périodique f est le signal défini pour tout $n \in \mathbb{Z}^d$ par

$$f^{\check{}}[n] = \frac{1}{N^d} \sum_{k \in [0,N-1]^d} f[k] \exp\left(i \frac{2\pi \langle k, n \rangle}{N}\right).$$

Partie 1. TFD et produit de convolution.

On fixe $d = 2$.

1. Vérifier que $f \circledast g$ et $\hat{f}$ sont tous les deux des signaux N-périodiques.
2. Montrer que, pour tout $n \in \mathbb{Z}^2$,

$$\widehat{f \circledast g}[n] = \hat{f}\,[n]\, \hat{g}[n].$$

3. Vérifier que, pour tout $n \in \mathbb{Z}^2$,

$$f[n] = (\hat{f}\,)^{\check{}}[n].$$

4. En déduire une méthode basée sur la transformée de Fourier discrète permettant de calculer le produit de convolution périodique discret.

Exercice 2 :

Partie 1 :

1 : Le produit de convolution périodique discret est défini par :

$$(f * g)[n] = \sum_{k \in [[0,N-1]]^2} f[n-k]g[k]$$

Nous devons vérifier que $(f * g)[n + Ne_j] = (f * g)[n]$ pour $j \in \{1, 2\}$, où $e_1 = (1, 0)$ et $e_2 = (0, 1)$ sont les vecteurs de base de $\mathbb{Z}^2$.

Calculons :

$$(f * g)[n + Ne_j] = \sum_{k \in [[0,N-1]]^2} f[n + Ne_j - k]g[k]$$

Puisque f est N-périodique, nous avons :

$$f[n + Ne_j - k] = f[n - k]$$

Donc :

$$(f*g)[n+Ne_j] = \sum_{k\in[[0,N-1]]^2} f[n-k]g[k] = (f*g)[n]$$

Donc : $f*g$ est N-périodique.

La transformée de Fourier discrète d'un signal N-périodique f est définie par :

$$\hat{f}\,[m] = \sum_{k\in[[0,N-1]]^2} f[k]e^{-i2\pi\frac{\langle k,m\rangle}{N}}$$

Nous voulons montrer que :

$$\hat{f}\,[m+Ne_j] = \hat{f}\,[m], \quad \text{pour } j\in\{1,2\}$$

Calculons :

$$\hat{f}\,[m+Ne_j] = \sum_{k\in[[0,N-1]]^2} f[k]e^{-i2\pi\frac{\langle k,m+Ne_j\rangle}{N}}$$

En utilisant la linéarité du produit scalaire :

$$\langle k, m+Ne_j\rangle = \langle k,m\rangle + Nk_j$$

Donc :

$$\hat{f}\,[m+Ne_j] = \sum_{k\in[[0,N-1]]^2} f[k]e^{-i2\pi\frac{\langle k,m\rangle}{N}}e^{-i2\pi k_j}$$

Or, $e^{-i2\pi k_j} = 1$ car k_j est un entier. Ainsi :

$$\hat{f}\,[m+Ne_j] = \sum_{k\in[[0,N-1]]^2} f[k]e^{-i2\pi\frac{\langle k,m\rangle}{N}} = \hat{f}\,[m]$$

Donc : $\hat{f}$ est N-périodique.

2 : On applique la TFD à la convolution :

$$\widehat{(f*g)}[m] = \sum_{n\in[[0,N-1]]^2} (f*g)[n]e^{-i2\pi\frac{\langle n,m\rangle}{N}}$$

En utilisant la définition de la convolution, on a :

$$\widehat{(f*g)}[m] = \sum_{n\in[[0,N-1]]^2} \left(\sum_{k\in[[0,N-1]]^2} f[n-k]g[k]\right) e^{-i2\pi\frac{\langle n,m\rangle}{N}}$$

En intervertissant les sommes, on obtient :

$$\widehat{(f*g)}[m] = \sum_{k\in[[0,N-1]]^2} g[k] \sum_{n\in[[0,N-1]]^2} f[n-k]e^{-i2\pi\frac{\langle n,m\rangle}{N}}$$

En posant $n' = n - k$ (ce qui ne change pas la somme car f est N-périodique et la somme est calculée sur un intervalle de taille N), on obtient:

$$\widehat{(f*g)}[m] = \sum_{k\in[[0,N-1]]^2} g[k]e^{-i2\pi\frac{\langle k,m\rangle}{N}} \sum_{n'\in[[0,N-1]]^2} f[n']e^{-i2\pi\frac{\langle n',m\rangle}{N}}$$

On reconnaît alors $\hat{f}\,[m]$ et $\hat{g}[m]$.

Donc : $\widehat{(f*g)}[m] = \hat{f}\,[m] \cdot \hat{g}[m]$

3 : La transformée de Fourier discrète d'un signal f est donnée par :

$$\hat{f}\,[m] = \sum_{k\in[[0,N-1]]^2} f[k]e^{-i2\pi\frac{\langle k,m\rangle}{N}}$$

La transformée de Fourier inverse discrète est définie par :

$$\hat{(\hat{f}\,)}[n] = \frac{1}{N^2} \sum_{m\in[[0,N-1]]^2} \hat{f}\,[m]e^{i2\pi\frac{\langle m,n\rangle}{N}}$$

En remplaçant $\hat{f}\,[m]$ par son expression, on obtient :

$$\hat{(\hat{f}\,)}[n] = \frac{1}{N^2} \sum_{m\in[[0,N-1]]^2} \left(\sum_{k\in[[0,N-1]]^2} f[k]e^{-i2\pi\frac{\langle k,m\rangle}{N}} \right) e^{i2\pi\frac{\langle m,n\rangle}{N}}$$

Nous pouvons intervertir les sommes :

$$\hat{(\hat{f}\,)}[n] = \frac{1}{N^2} \sum_{k\in[[0,N-1]]^2} f[k] \sum_{m\in[[0,N-1]]^2} e^{-i2\pi\frac{\langle k,m\rangle}{N}} e^{i2\pi\frac{\langle m,n\rangle}{N}}$$

En réécrivant le produit de deux exponentielles comme une exponentielle unique, on obtient :

$$f[n] = \frac{1}{N^2} \sum_{k\in[[0,N-1]]^2} f[k] \sum_{m\in[[0,N-1]]^2} e^{i2\pi\frac{\langle m,n-k\rangle}{N}}$$

Nous allons maintenant étudier la somme de droite :

$$\sum_{m\in[[0,N-1]]^2} e^{i2\pi\frac{\langle m,n-k\rangle}{N}}$$

Si $n = k$, alors $\langle m, 0\rangle = 0$, et chaque terme vaut $e^0 = 1$, donc :

$$\sum_{m\in[[0,N-1]]^2} e^{i2\pi\frac{\langle m,n-k\rangle}{N}} = \sum_{m\in[[0,N-1]]^2} 1 = N^2$$

Si $n \neq k$, on pose $p = n - k$ tel que $p = (p_1, p_2) \neq (0,0)$, et on obtient :

$$\sum_{m\in[[0,N-1]]^2} e^{i2\pi\frac{\langle m,p\rangle}{N}}$$

$$= \sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} e^{i2\pi\frac{(m_1p_1+m_2p_2)}{N}}$$

$$= \left(\sum_{m_1=0}^{N-1} e^{i2\pi\frac{m_1p_1}{N}}\right)\cdot\left(\sum_{m_2=0}^{N-1} e^{i2\pi\frac{m_2p_2}{N}}\right)$$

On reconnaît ici deux séries géométriques de raisons $r_1 = e^{i2\pi\frac{p_1}{N}}$ et $r_2 = e^{i2\pi\frac{p_2}{N}}$.

On a donc :

$$\left(\sum_{m_1=0}^{N-1} e^{i2\pi\frac{m_1p_1}{N}}\right)\cdot\left(\sum_{m_2=0}^{N-1} e^{i2\pi\frac{m_2p_2}{N}}\right) = \frac{1-r_1^N}{1-r_1}\cdot\frac{1-r_2^N}{1-r_2}$$

Or, comme $r_1 = e^{i2\pi\frac{p_1}{N}}$ et $r_2 = e^{i2\pi\frac{p_2}{N}}$, nous avons :

$$r_1^N = e^{i2\pi p_1} = 1 \text{ et } r_2^N = e^{i2\pi p_2} = 1$$

On a donc :

$$\frac{1-r_1^N}{1-r_1}\cdot\frac{1-r_2^N}{1-r_2} = 0\cdot 0 = 0$$

Donc, si $n \neq k$:

$$\sum_{m\in[[0,N-1]]^2} e^{i2\pi\frac{\langle m,n-k\rangle}{N}} = 0$$

Nous avons donc :

$$(\hat{f})[n] = \frac{1}{N^2}\sum_{k\in[[0,N-1]]^2} f[k]\cdot N^2\delta_{nk}$$

Comme δ_{nk} vaut 1 si $n = k$ et 0 sinon, on obtient :

$$(\hat{f})[n] = f[n]$$

4 : Grâce aux résultats précédents, nous pouvons établir une méthode efficace pour calculer la convolution :

- Calculer la transformée de Fourier discrète des signaux f et g :

$$\hat{f} = \mathrm{TFD}(f), \quad \hat{g} = \mathrm{TFD}(g)$$

- Multiplier leurs transformées de Fourier :

$$\hat{h} = \hat{f} \cdot \hat{g}$$

- Appliquer la transformée de Fourier inverse pour récupérer h :

$$h = \mathrm{TFD}^{-1}(\hat{h})$$

Partie 2. Filtre gaussien.

Soit N un entier pair et $s > 0$. On considère un filtre linéaire dont la fonction de transfert est le signal gaussien discret N-périodique défini pour $n \in [\![-\frac{N}{2}, \frac{N}{2} - 1]\!]^2$ par

$$\hat{g}[n] = \frac{1}{Z_{N,s}} \exp\left(-s|n|^2\right) = \frac{1}{Z_{N,s}} \exp\left(-s(n_1^2 + n_2^2)\right),$$

où $Z_{N,s}$ est une constante de normalization donnée par

$$Z_{N,s} = \sum_{n \in [-N/2, N/2-1]^2} \exp\left(-s|n|^2\right),$$

1. En complétant la fonction ci-dessous, écrire une fonction python qui étant donnés les paramètres N et s renvoie la fonction de transfert $\hat{g}$.

In [324…
```
from numpy import floor, exp, power, arange, reshape, meshgrid
import numpy as np
import matplotlib.pyplot as plt
from imageio import imread
from scipy.ndimage import convolve, gaussian_filter
from numpy.fft import fft2, ifft2
from scipy.signal import convolve2d
from scipy.sparse.linalg import LinearOperator, cg
```

In [325…
```
def Gaussien(N=512, s=0.01):
    """Define a bi-dimensional N-periodic Gaussian kernel.

   Parameters
    ----------
    N : int
        image size. The default is 512.
    s : positive float, optional
        scale parameter. The default is 1.

    Returns
    -------
    The Gaussian kernel
    """
    M = floor(N / 2)
```

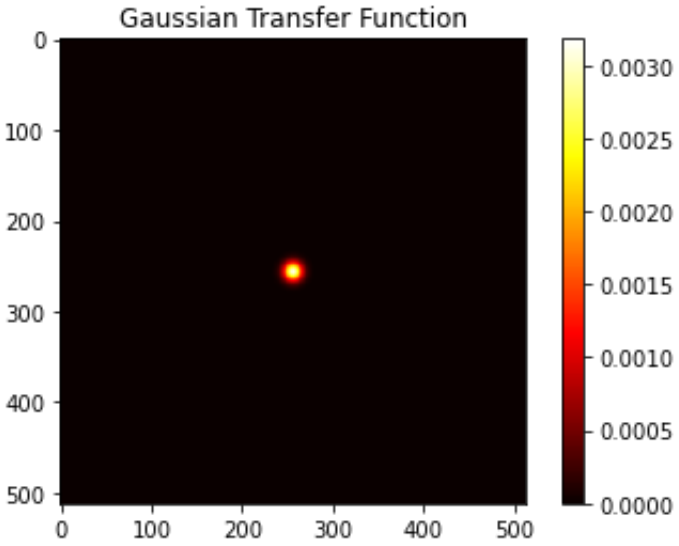
```
    # Definition of an unidimensional Gaussian kernel.
    g = reshape(exp(- s * power(arange(- M, M, 1), 2)), (N, 1))

    # Extension to a bi-dimensional Gaussian kernel.
    x = arange(-M, M, 1)
    y = arange(-M, M, 1)
    X, Y = meshgrid(x, y)
    g_2d = exp(-s * (X**2 + Y**2))
    g_2d /= g_2d.sum()
    return g_2d
```

2. Afficher la fonction de transfert $\hat{g}$ avec la commande *imshow* du module **matplotlib.pyplot**. Donner une interprétation du filtre.

3. Appliquer le filtre de réponse de transfert $\hat{g}$ à l'image *baboon.jpg*. On utilisera les méthodes *fft2* et *ifft2* du module **numpy.fft** pour réaliser les transformées de Fourier discrètes (directe et inverse). Avant d'appliquer le filtre, veiller à replacer le pixel $n = [0, 0]$ de la fonction de transfert à l'indice (0, 0) de la matrice en opérant une translation avec la commande *fftshift* de **numpy.fft**.

4. Faire varier les valeurs de s et commenter l'effet de ce paramètre sur la sortie du filtre.

In [326…
```
if __name__ == "__main__":
    N, s = 512, 0.01
    g_2d = Gaussien(N, s)
    plt.imshow(g_2d, cmap='hot', interpolation='nearest')
    plt.colorbar()
    plt.title("Gaussian Transfer Function")
    plt.show()
```

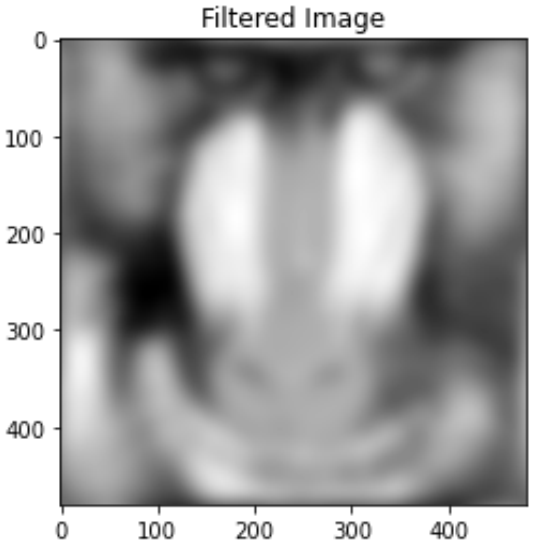


Le filtre gaussien agit comme un filtre passe-bas, ce qui signifie qu'il lisse l'image en réduisant les hautes fréquences.

In [327…
```
def apply_filter(image_path, N=512, s=0.01):
    """Applique un filtre Gaussien à une image en utilisant la FFT."""
```

```
    image = imread(image_path)
    g_2d = np.fft.fftshift(Gaussien(N, s))
    image_fft = np.fft.fft2(image)
    filtered_fft = image_fft * g_2d
    return np.fft.ifft2(filtered_fft).real

filtered_image = apply_filter("baboon.jpg", 480, s)
plt.imshow(filtered_image, cmap='gray')
plt.title("Filtered Image")

plt.show()
```

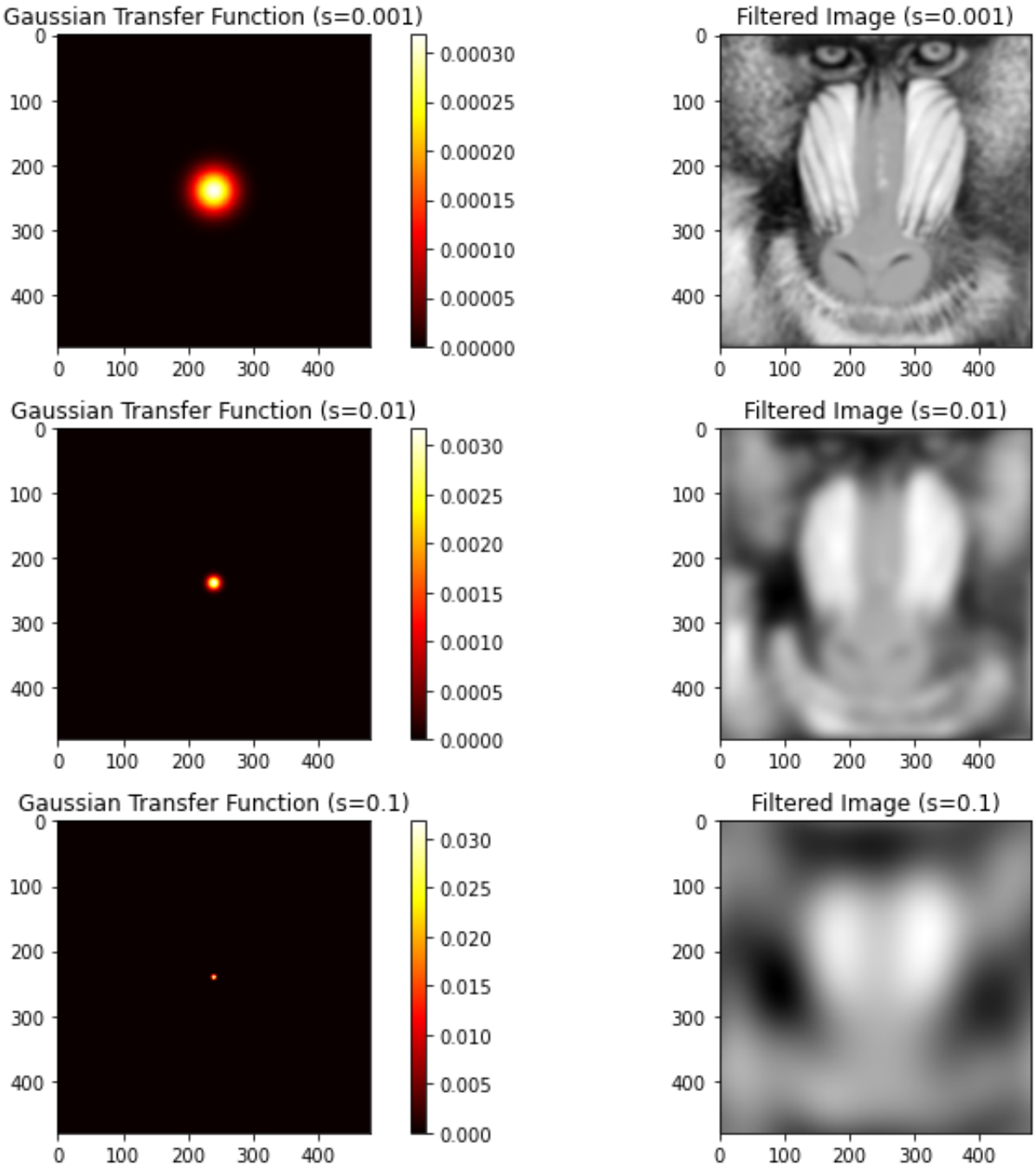
```
/tmp/ipykernel_7631/2366203110.py:3: DeprecationWarning: Starting with I
mageIO v3 the behavior of this function will switch to that of iio.v3.im
read. To keep the current behavior (and make this warning disappear) use
`import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imread(image_path)
```



In [328…

```
if __name__ == "__main__":
    N = 480
    s_values = [0.001, 0.01, 0.1]
    plt.figure(figsize=(12, 9))
    for i, s in enumerate(s_values):
        g_2d = Gaussien(N, s)
        filtered_image = apply_filter("baboon.jpg", N, s)
        plt.subplot(len(s_values), 2, 2*i + 1)
        plt.imshow(g_2d, cmap='hot', interpolation='nearest')
        plt.colorbar()
        plt.title(f"Gaussian Transfer Function (s={s})")
        plt.subplot(len(s_values), 2, 2*i + 2)
        plt.imshow(filtered_image, cmap='gray')
        plt.title(f"Filtered Image (s={s})")
    plt.tight_layout()
    plt.show()
```

```
/tmp/ipykernel_7631/2366203110.py:3: DeprecationWarning: Starting with I
mageIO v3 the behavior of this function will switch to that of iio.v3.im
read. To keep the current behavior (and make this warning disappear) use
`import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imread(image_path)
```



Plus s est grand, plus le flou est important.

Partie 3. Filtres dérivatifs.

Soit N un entier pair et $s > 0$. On considère un filtre linéaire dont la fonction de transfert est le signal gaussien discret N-périodique défini pour $n \in [\![-N/2, N/2-1]\!]^2$par

$$\hat{g}[n] = \frac{1}{Z_{N,s}} \exp\left(-s|n|^2\right) = \frac{1}{Z_{N,s}} \exp\left(-s(n_1^2 + n_2^2)\right),$$

où $Z_{N,s}$ est une constante de normalization définie par

$$Z_{N,s} = \sum_{n \in [-N/2, N/2-1]^2} \exp\left(-s|n|^2\right),$$

On considère trois nouvelles fonctions de transfert définies pour $n = (n_1, n_2) \in [\![-N/2, N/2-1]\!]^2$de la manière suivante:

- pour $j \in [\![1, 2]\!]$

$$\widehat{\partial_j g}[n] = i n_j \hat{g}[n]$$

et

$$\widehat{\Delta g}[n] = -|n|^2 \hat{g}[n].$$

1. En vous référant à l'exercice 1, donner une interprétation des filtres associés à ces fonctions de transfert.

2. Créer une fonction python qui définit ces fonctions de transfert. Visualiser ces fonctions de transfert et indiquer si les filtres associés sont passe-haut, passe-bas ou passe-bande ?

3. Appliquer les filtres associés à ces fonctions de transfert à l'image *baboon.jpg*. On notera

- $\partial_j f = f \circledast \partial_j g$, pour $j \in [\![1, 2]\!]$,
- $\Delta f = f \circledast \Delta g$.

Visualiser les réponses de ces filtres. Quelles informations apportent-elles ?

4. Pour une image donnée f, former l'image h définie pour tout $n \in \mathbb{Z}^2$ par

$$h[n] = |f \circledast \partial_1 g[n]|^2 + |f \circledast \partial g_2[n]|^2.$$

Que représente cette image ? A quoi correspondent les zones où les valeurs de cette image sont élevées ? Pour s'en rendre compte, on pourra seuiller l'image, c'est à dire définir l'image binaire

$$h_\eta[n] = \begin{cases} 1 & \text{si } \ h[n] > \eta, \\ 0 & \text{sinon} \end{cases}$$

pour η fixé convenablement.

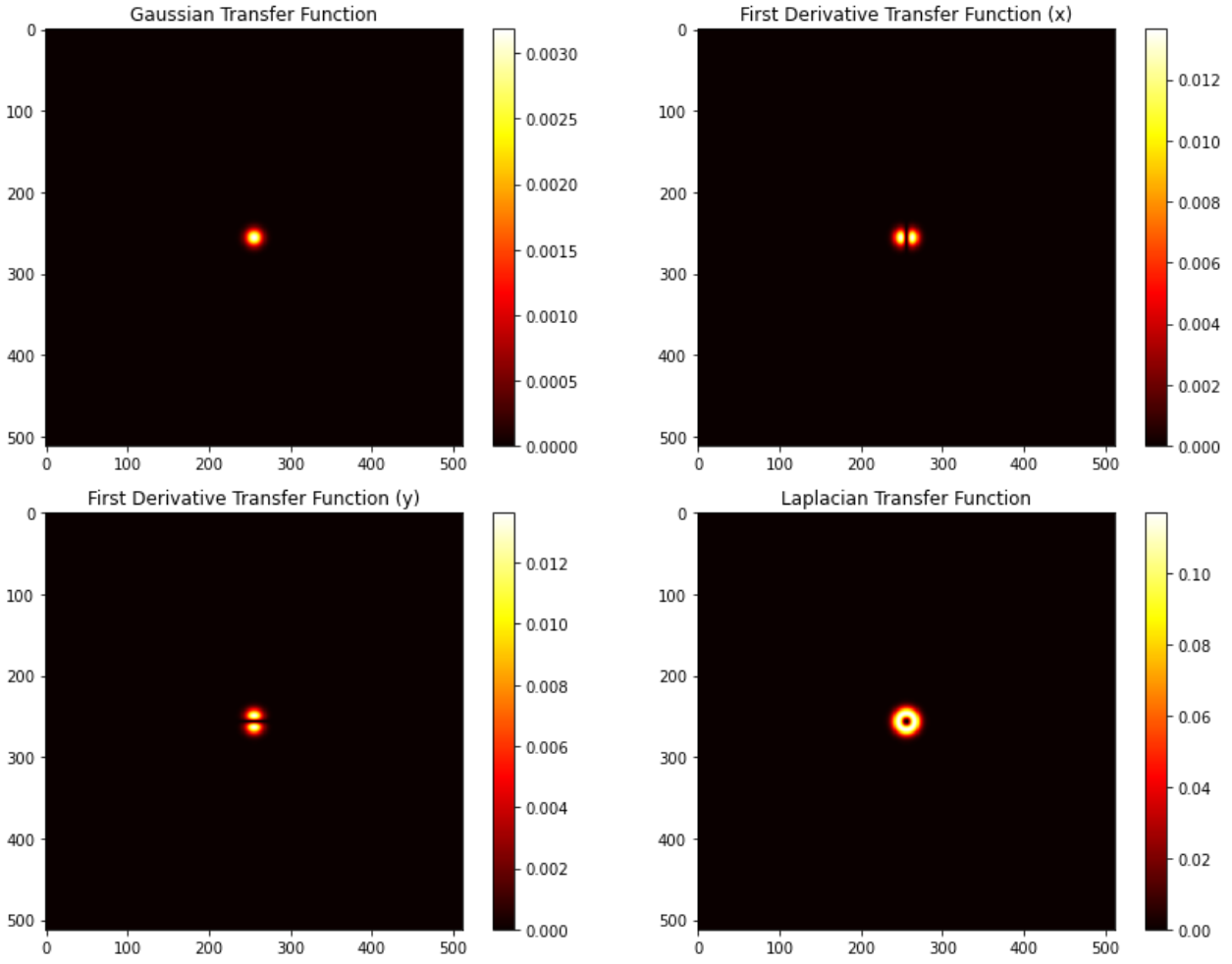
Partie 3 :

1 : En nous appuyant sur les résultats de l'exercice 1, on remarque que le filtre associé à la fonction de transfert $\widehat{\partial_j g}[n] = i n_j \hat{g}[n]$ correspond à un filtre dérivatif dans la direction j. Ce filtre va donc détecter les contours dans la direction j, car il met en évidence les variations d'intensité dans l'image.

Le filtre associé à la fonction de transfert $\widehat{\Delta g}[n] = -|n|^2 \hat{g}[n]$ correspond lui à la somme des dérivées secondes e va donc détecter les contours dans toutes les directions.

In [329…

```
def transfer_functions(N=512, s=0.01):
    """Calcule la fonction de transfert Gaussienne et ses dérivées."""
    M = floor(N / 2)
    x, y = arange(-M, M, 1), arange(-M, M, 1)
    X, Y = meshgrid(x, y)
    g_hat = exp(-s * (X**2 + Y**2)) / exp(-s * (X**2 + Y**2)).sum()
    return g_hat, 1j * X * g_hat, 1j * Y * g_hat, -(X**2 + Y**2) * g_hat

if __name__ == "__main__":
    N, s = 512, 0.01
    g_hat, d1_g, d2_g, laplacian_g = transfer_functions(N, s)
    plt.figure(figsize=(12, 9))
    functions = [(g_hat, "Gaussian Transfer Function"),
                 (np.abs(d1_g), "First Derivative Transfer Function (x)"
                 (np.abs(d2_g), "First Derivative Transfer Function (y)"
                 (np.abs(laplacian_g), "Laplacian Transfer Function")]
    for i, (func, title) in enumerate(functions):
        plt.subplot(2, 2, i + 1)
        plt.imshow(func, cmap='hot', interpolation='nearest')
        plt.colorbar()
        plt.title(title)
    plt.tight_layout()
    plt.show()
```

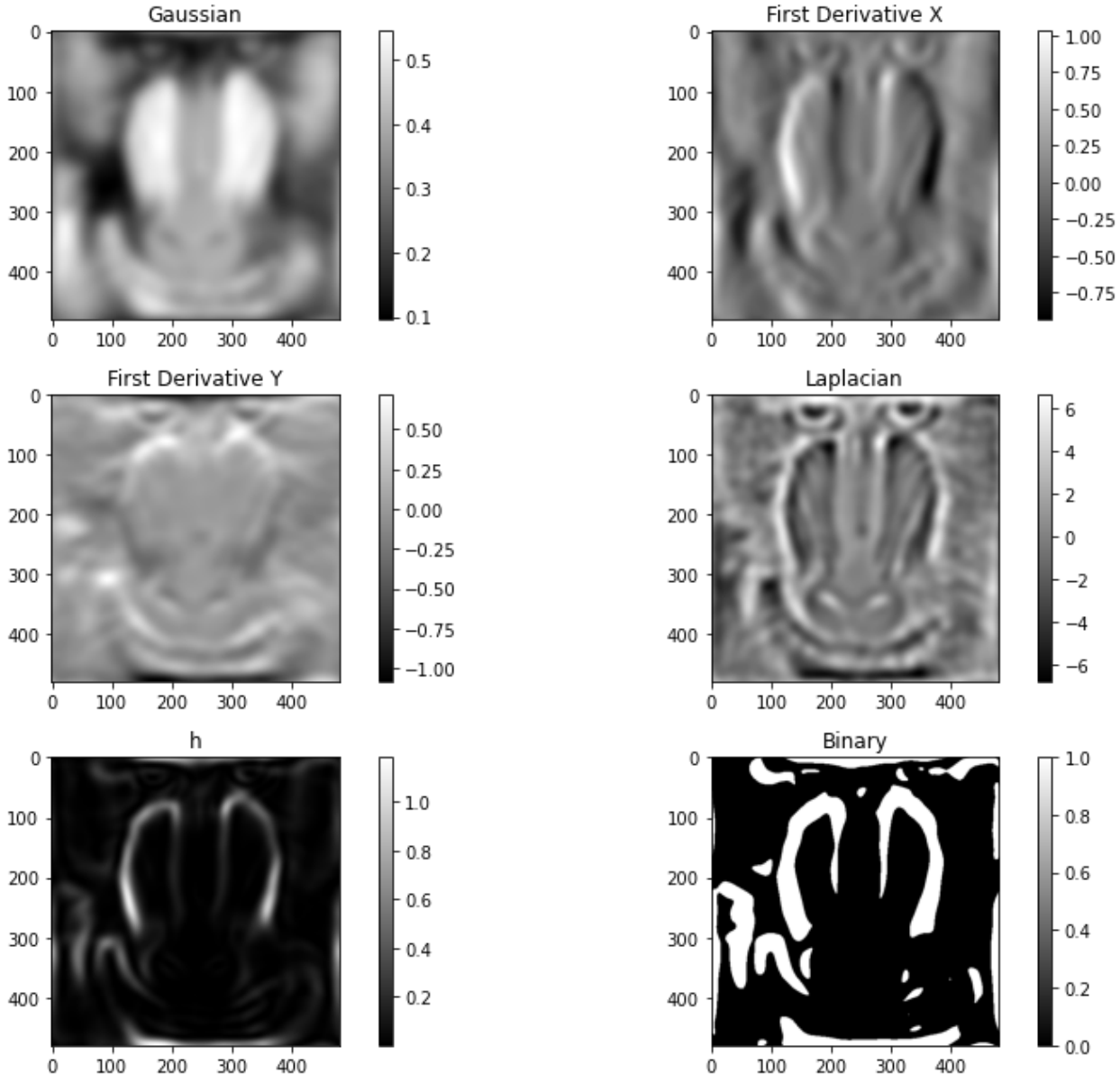


Le filtre gaussien est un filtre passe-bas. Les filtres dérivatifs premier (x et y) sont des filtres passe-bandes. Le filtre laplacien est un filtre passe-haut.

In [330…

```python
def apply_filter_with_derivatives(image_path, N=480, s=0.01):
    """Applique le filtre Gaussien et ses dérivées sur une image."""
    image = imread(image_path)[:N, :N]
    g_hat, d1_g, d2_g, laplacian_g = transfer_functions(N, s)
    image_fft = np.fft.fft2(image)
    d1_f = np.fft.ifft2(image_fft * np.fft.fftshift(d1_g)).real
    d2_f = np.fft.ifft2(image_fft * np.fft.fftshift(d2_g)).real
    h = d1_f**2 + d2_f**2
    eta = np.max(h) * 0.1
    h_binary = (h > eta).astype(int)
    return {"Gaussian": np.fft.ifft2(image_fft * np.fft.fftshift(g_hat))
            "First Derivative X": d1_f,
            "First Derivative Y": d2_f,
            "Laplacian": np.fft.ifft2(image_fft * np.fft.fftshift(laplac
            "h": h,
            "Binary": h_binary}

if __name__ == "__main__":
    N, s = 480, 0.01
    image_path = "baboon.jpg"
    filtered_images = apply_filter_with_derivatives(image_path, N, s)
    plt.figure(figsize=(12, 9))
    for i, (title, img) in enumerate(filtered_images.items()):
        plt.subplot(3, 2, i + 1)
        plt.imshow(img, cmap='gray')
        plt.title(title)
        plt.colorbar()
    plt.tight_layout()
    plt.show()
```

```
/tmp/ipykernel_7631/844702663.py:3: DeprecationWarning: Starting with Im
ageIO v3 the behavior of this function will switch to that of iio.v3.imr
ead. To keep the current behavior (and make this warning disappear) use
`import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imread(image_path)[:N, :N]
```



Le filtre gaussien a lissé l'image supprimant les détails fins comme les poils et mettant en avant des formes globales comme la forme du visage, du nez etc.

Sur l'image de la première dérivée x, les contours verticaux du visage comme le nez,

les poils, la moustache etc sont accentués car ils sont orientés verticalement. L'image est assez reconnaissable ce qui témoigne du grand nombre de contours verticaux.

A l'inverse, sur l'image de la première dérivée y, l'image est très floue ce qui confirme le faible nombre de contours horizontaux hormis les paupières et lèvres.

Le filtre laplacien accentue tous les contours qu'ils soient verticaux ou horizontaux, il ressemble donc au filtre de la première dérivée x avec cependant un peu plus de détails (les contours horizontaux).

L'image h met en évidence les contours, montrant des zones de forte variation d'intensité. L'image binaire isole les contours forts pour des applications de détection de contours.

Exercice 3. Détection de contours dans une image.

Introduction.

Les contours d'une image sont des courbes du domaine de l'image où sont localisées des fortes variations de niveaux de gris de l'image. Ces contours peuvent indiquer des frontières entre des objets présents dans l'image.

Pour détecter des zones où les variations des niveaux gris sont élevées, on peut utiliser la norme du gradient de l'image. En effet, une norme de gradient élevée est un indicateur de fortes variations des niveaux de gris. Ainsi, en seuillant la norme du gradient de l'image, il est possible de repérer des zones de fortes variations.

Cependant, les zones ainsi détectées ne sont généralement pas des courbes. Elles forment le plus souvent des bandes. Bien que ces bandes contiennent les contours, elles sont plus "larges" que ces derniers.

Pour réduire ces bandes à des courbes, on se restreint généralement aux pixels où la norme du gradient est localement maximale dans la direction du gradient. Pour déceler cette maximalité locale de la norme du gradient, Marr et Hildreth (1980) ont proposé de répérer les passages par zéro du laplacien de l'image.

Algorithme.

De cette approche, on tire l'algorithme de détection de contours suivant.

- Etape 1: pour une image donnée f, on calcule le laplacien de l'image Δf définie, pour tout $n \in \mathbb{Z}^2$, par

$$\Delta f[n] = f \circledast \Delta g[n],$$

où g est un filtre gaussien discret et Δg son laplacien.

- Etape 2: On détecte les passages par zéro de Δf:
  - Pour cela, on forme l'image du signe k_0 de Δf:

$$k_0[n] = \begin{cases} 1 & \text{si } \Delta f[n] > 0, \\ -1 & \text{sinon.} \end{cases}$$

  - On calcule la réponse $\tilde{k}$ de l'image k_0 au filtre dont le noyau est à support sur $[\![-1,1]\!]^2$et a pour valeurs sur $[\![-1,1]\!]^2$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

- Etape 3: On calcule la norme (au carré) du gradient de h:

$$h[n] = |f \circledast \partial_1 g[n]|^2 + |f \circledast \partial g_2[n]|^2.$$

- Etape 4: Les contours sont définis par les pixels n tels que $\tilde{k}[n] > 0$ et $h[n] > \eta$, pour $\eta > 0$ fixé convenablement.

Mise en oeuvre.

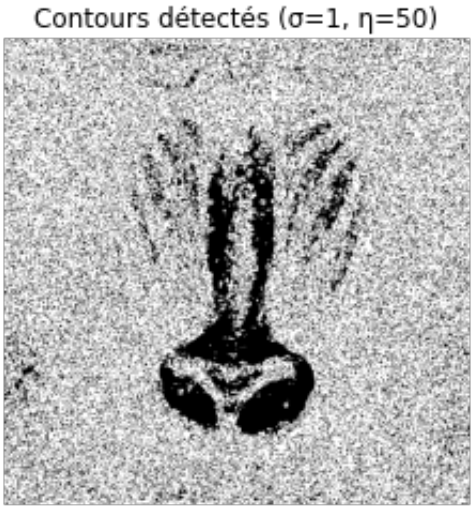
1. Mettre en oeuvre la méthode et l'appliquer à l'image *baboon.jpg*.
2. Faire varier les paramètres s et η de la méthode et analyser leur effets sur la détection de contours.

In [331…

```
image = imread('baboon.jpg')

sigma = 1.0  # Paramètre de lissage
eta = 50  # Seuil de détection des contours

def detect_contours(image, sigma, eta):

    # Calcul du laplacien
    g = gaussian_filter(image, sigma=sigma)
    laplacian = convolve(g, np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])

    # Détection des passages par zéro
    k0 = np.where(laplacian > 0, 1, -1)
    kernel = np.array([[0, 1, 0], [1, -1, 1], [0, 1, 0]])
    k_filtered = convolve(k0, kernel)

    # Calcul de la norme du gradient
    dx = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    dy = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
    grad_x = convolve(g, dx)
    grad_y = convolve(g, dy)
    norm_gradient = grad_x**2 + grad_y**2
```

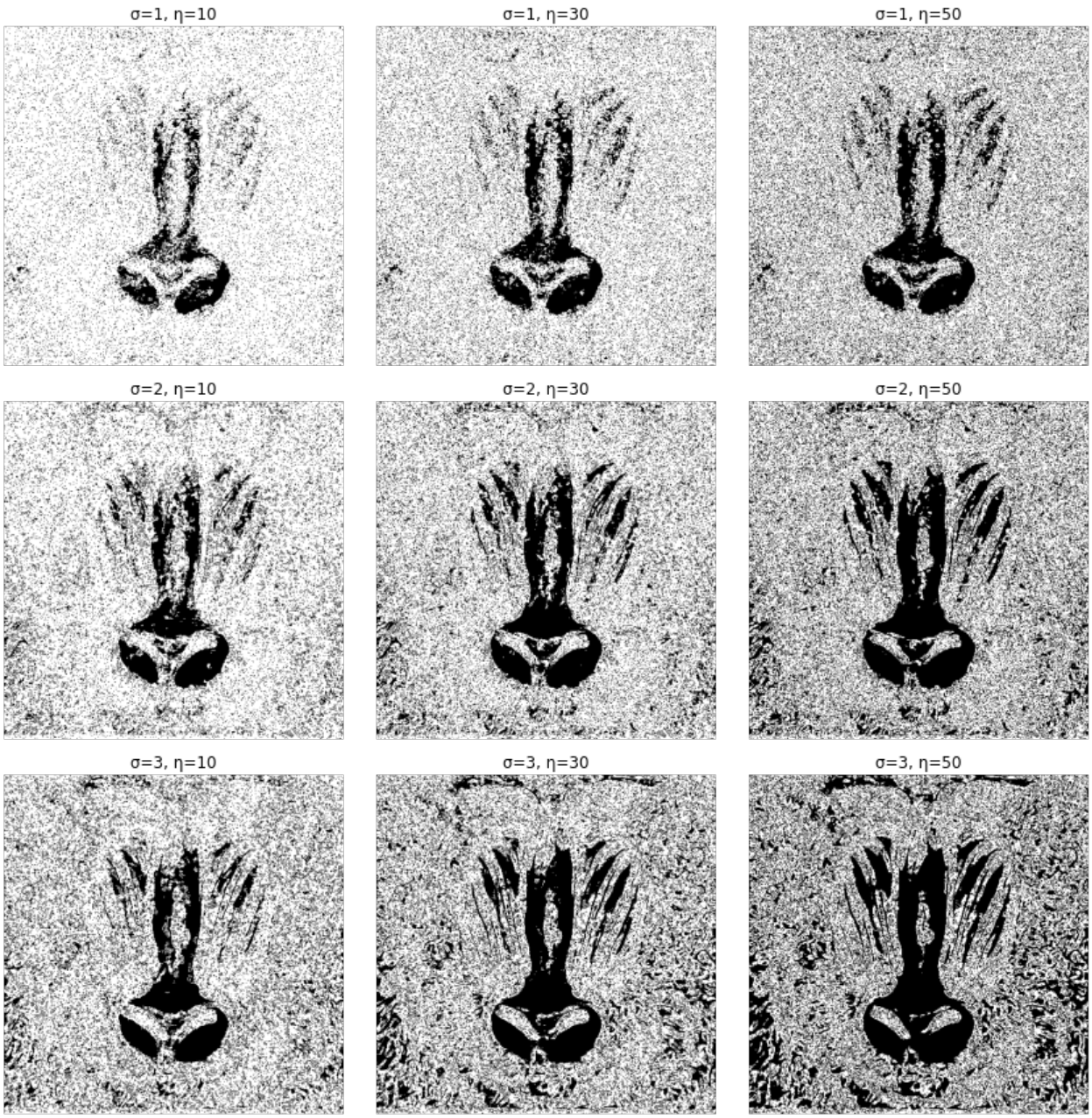
```
    # Détection des contours
    contours = np.where((k_filtered > 0) & (norm_gradient > eta), 1, 0)
    return contours

contours = detect_contours(image, sigma, eta)

plt.imshow(contours, cmap='gray')
plt.title("Contours détectés (σ=1, η=50)")
plt.axis('off')
plt.show()
```

```
/tmp/ipykernel_7631/2698151499.py:1: DeprecationWarning: Starting with I
mageIO v3 the behavior of this function will switch to that of iio.v3.im
read. To keep the current behavior (and make this warning disappear) use
`import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imread('baboon.jpg')
```



In [332…

```
sigma_values = [1, 2, 3]
eta_values = [10, 30, 50]

plt.figure(figsize=(12, 12))
for i, sigma in enumerate(sigma_values):
    for j, eta in enumerate(eta_values):
        contours = detect_contours(image, sigma, eta)
        plt.subplot(len(sigma_values), len(eta_values), i * len(eta_valu
        plt.imshow(contours, cmap='gray')
        plt.title(f"σ={sigma}, η={eta}")
        plt.axis('off')

plt.tight_layout()
plt.show()
```



Plus σ (paramètre de lissage) est petit, moins il y a de lissage, donc la détection de contours détecte plus de détails et est donc plus sensible au bruit. Plus η (seuil de détection de contours) est petit, plus il y a de contours détectés, y compris des contours faibles ou du bruit.

Exercice 4. Déflouage d'image.

Introduction.

A l'acquisition, une image peut subir des dégradations telles que du flou ou du bruit. Dans certains cas, on peut modéliser ces dégradations au moyen du modèle suivant.

$$h = f \circledast g + b,$$

où

- h est l'image observée,
- b est un bruit additif gaussien indépendant de f et g,
- f est une image non observée sans dégradation ni bruit,
- g est un noyau de convolution qui, appliquée à f, la dégrade. Par exemple, si g est la réponse impulsionnelle d'un filtre passe-bas, la convolution de f avec g va rendre l'image f floue.

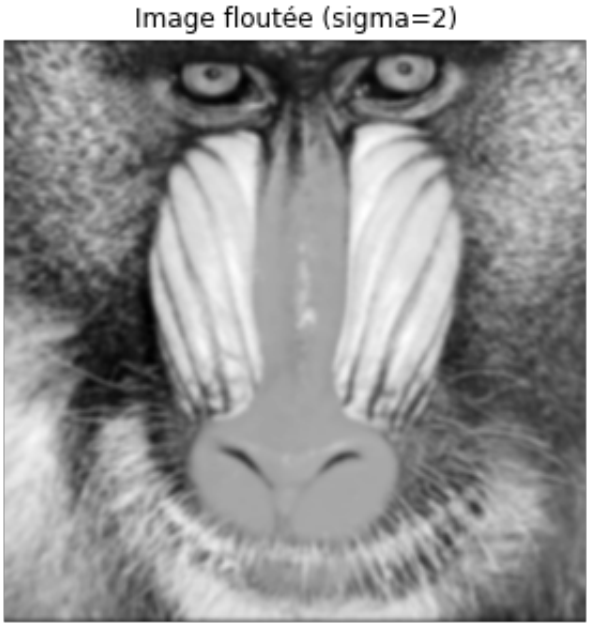
Déconvoluer l'image consiste à retrouver f à partir de l'image dégradée h. On parle de déflouage d'image lorsque l'image a été dégradée par un filtre passe-bas.

Partie 1. Floutage d'une image.

On prend un filtre gaussien discret g comme noyau de convolution

1. Flouter l'image *baboon.jpg* en lui appliquant le filtre g.
2. Ajouter un bruit gaussien à l'image floutée.

In [333…
```
sigma = 2  # Plus sigma est grand, plus le flou est intense

# Appliquer un flou gaussien
image_floutee = gaussian_filter(image, sigma=sigma)

plt.figure(figsize=(5,5))
plt.imshow(image_floutee, cmap='gray')
plt.title(f"Image floutée (sigma={sigma})")
plt.axis("off")
plt.show()
```



In [334…
```
sigma_b = 1  # Plus sigma_b est grand, plus le bruit est intense

# Ajouter un bruit gaussien à l'image floutée
bruit = np.random.normal(0, sigma_b, image_floutee.shape)
image_bruitee = image_floutee + bruit
```

```
# Affichage
plt.figure(figsize=(5,5))
plt.imshow(image_bruitee, cmap='gray')
plt.title(f"Image floutée + bruit (sigma_b={sigma_b})")
plt.axis("off")
plt.show()
```

Partie 2. Méthode d'inversion directe.

1. Dans un premier temps, on considère le modèle de dégradation sans bruit ($b = 0$). On suppose que la TFD $\hat{g}$ de g ne s'annule pas. Exprimer la TFD de f en fonction de celles de h et g.

2. En déduire une méthode pour déconvoluer h en utilisant la TFD et la TFDI.

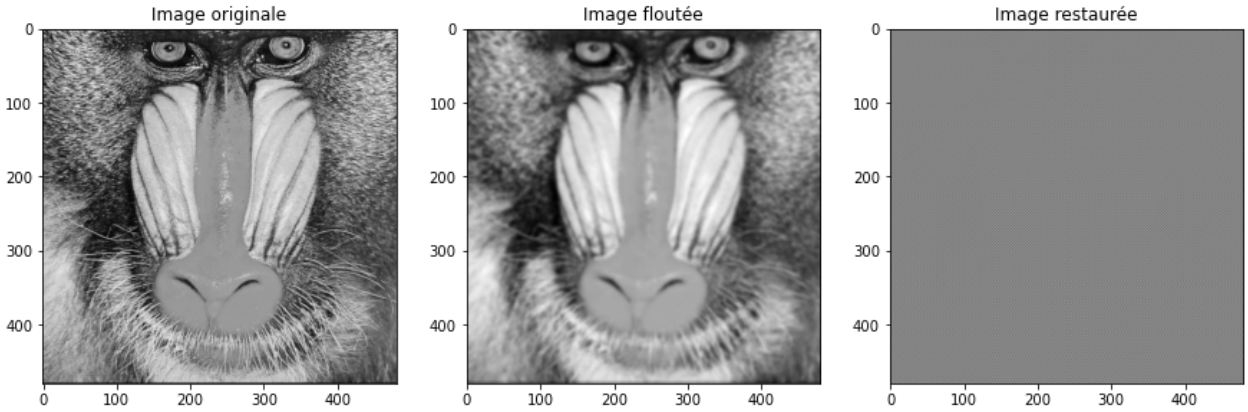
3. Appliquer cette méthode à l'image de *baboon.jpg* convoluée avec le filtre gaussien g. Evaluer l'erreur d'approximation de f en calculant la différence quadratique entre l'image déconvoluée et l'image originale.

4. Ajouter un bruit dans le modèle de dégradation. Appliquer la méthode d'inversion précédente à cette nouvelle image dégradée. Que constatez-vous et comment expliquez-vous ce résultat ?

In [335…

```
# Fonction de déconvolution directe
def deconvolution_directe(h, g):
    h_fft = fft2(h)
    g_fft = fft2(g, s=h.shape)  # Redimensionne g pour qu'il ait la même
    epsilon = 1e-10  # Petite constante pour éviter les divisions par ze
    f_fft = h_fft / (g_fft + epsilon)
    return np.real(ifft2(f_fft))

# Paramètres du filtre gaussien
sigma = 2
```

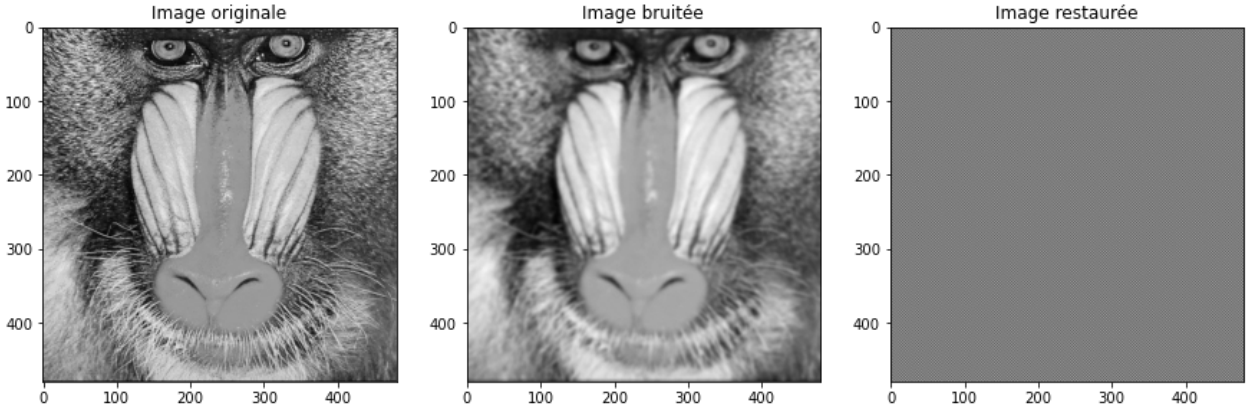
```
kernel_size = 5

# Création du noyau
x = np.arange(-kernel_size // 2 + 1, kernel_size // 2 + 1)
y = np.arange(-kernel_size // 2 + 1, kernel_size // 2 + 1)
xx, yy = np.meshgrid(x, y)
g = np.exp(-(xx**2 + yy**2) / (2 * sigma**2))
g = g / g.sum()  # Normalisation du noyau

h = image_floutee
image_restauree = deconvolution_directe(h, g)

# Calcul de l'erreur quadratique
error = np.mean((image - image_restauree) ** 2)

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(image, cmap='gray'); ax[0].set_title("Image originale")
ax[1].imshow(image_floutee, cmap='gray'); ax[1].set_title("Image floutée
ax[2].imshow(image_restauree, cmap='gray'); ax[2].set_title("Image resta
plt.show()

print(f"Erreur quadratique: {error:.6f}")
```




```
Erreur quadratique: 23563005.830433
```

In [336…

```
image_restauree = deconvolution_directe(image_bruitee, g)

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(image, cmap='gray'); ax[0].set_title("Image originale")
ax[1].imshow(image_bruitee, cmap='gray'); ax[1].set_title("Image bruitée
ax[2].imshow(image_restauree, cmap='gray'); ax[2].set_title("Image resta
plt.show()
```



Exercice 4 :

Partie 2 :

1 : On a $h = f * g$. Or, $\widehat{(f * g)} = \hat{f} \cdot \hat{g}$. Ainsi, la transformée de Fourier discrète (TFD) de h s'écrit :

$$\hat{h} = \hat{f} \cdot \hat{g}$$

Pour retrouver $\hat{f}$, nous supposons que $\hat{g}$ ne s'annule pas et on a :

$$\hat{f} = \frac{\hat{h}}{\hat{g}}$$

2 : Nous avons montré que $\hat{f}$ peut être obtenu par :

$$\hat{f} = \frac{\hat{h}}{\hat{g}}$$

Pour reconstruire f, il suffit d'appliquer la transformée de Fourier inverse (TFDI) :

$$f = \mathrm{TFDI}\left(\frac{\hat{h}}{\hat{g}}\right)$$

Donc, pour déconvoluer h, il faut :

- calculer la transformée de Fourier de l'image h : $\hat{h} = \mathrm{TFD}(h)$
- calculer la transformée de Fourier du noyau de convolution g : $\hat{g} = \mathrm{TFD}(g)$
- effectuer la division : $\hat{f} = \frac{\hat{h}}{\hat{g}}$
- appliquer la transformée de Fourier inverse : $f = \mathrm{TFDI}(\hat{f})$

Partie 3. Méthode par résolution de problème inverse.

On peut également déconvoluer une image dégradée h en minimisant un problème d'optimisation un critère de la forme

$$J(f) = \frac{1}{2} \sum_{n \in [0, N-1]^2} |f \circledast g[n] - h[n]|^2 + \lambda R(f),$$

Dans ce critère, le premier terme mesure l'écart entre l'image observée h et la valeur prédite de h à partir de f. Il s'agit d'un **terme d'attache aux données**. Le second terme induit des solutions f régulières. Il permet au problème d'optimisation d'être bien posé. Il s'agit d'un **terme de régularisation**. La valeur $\lambda > 0$ est un paramètre qui permet de régler le compromis entre l'attache aux données et la régularisation. Une

régularisation typique consiste à pénaliser le gradient ∇f de f avec un terme de la forme

$$R(f) = \frac{1}{2} \sum_{n \in [0,N-1]^2} |\nabla f[n]|^2 = \frac{1}{2} \sum_{n \in [0,N-1]^2} \left((\partial_1 f[n])^2 + (\partial_2 f[n])^2 \right).$$

Le critère J peut s'écrire sous une forme matricielle. On "aplatie" les images f et h en concaténant leurs colonnes en un seul vecteur. Le critère J se présente alors sous la forme

$$\tilde{J}(\tilde{f}) = \frac{1}{2}|G\tilde{f} - \tilde{h}|^2 + \frac{\lambda}{2}\left(|D_1\tilde{f}|^2 + |D_2\tilde{f}|^2 \right),$$

où

- $\tilde{f}$ et $\tilde{h}$ sont les versions aplaties de f et h, respectivement,
- G est une matrice de taille $N^2 \times N^2$ qui permet de réaliser le produit de convolution de f avec le noyau g,
- D_1 et D_2 sont des matrices de taille $N^2 \times N^2$ qui permettent de calculer les dérivées de f selon les lignes et les colonnes, respectivement.

1. En vous aidant de la forme matricielle de J, montrer que le problème d'optimisation admet une solution et l'expliciter en fonction de $\tilde{h}$.
2. Mettre en oeuvre et évaluer la méthode de déflouage par résolution du problème inverse. On pourra s'aider des fonctions ci-dessous qui permettent de mettre le problème d'optimisation sous une forme matricielle.

Partie 3 :

1 : On a :

$$J(\mathcal{f}) = \frac{1}{2}||G\mathcal{f} - \mathcal{h}||^2 + \frac{\lambda}{2}(||D_1\mathcal{f}||^2 + ||D_2\mathcal{f}||^2)$$

Pour minimiser $J(\mathcal{f})$, on dérive par rapport à $\mathcal{f}$ et on cherche où cette dérivée s'annule.

Le gradient d'un terme quadratique $||Ax - b||^2$ est donné par :

$$\nabla_x ||Ax - b||^2 = 2A^T(Ax - b)$$

Appliquons cette règle aux deux termes de $J(\mathcal{f})$:

$$\nabla_{\mathcal{f}} \frac{1}{2}||G\mathcal{f} - \mathcal{h}||^2 = G^T(G\mathcal{f} - \mathcal{h})$$

$$\nabla_{\mathcal{f}} \frac{\lambda}{2}(||D_1\mathcal{f}||^2 + ||D_2\mathcal{f}||^2) = \lambda(D_1^T D_1 + D_2^T D_2)\mathcal{f}$$

En annulant le gradient, on obtient l'équation :

$$G^T G f + \lambda (D_1^T D_1 + D_2^T D_2) f = G^T h$$

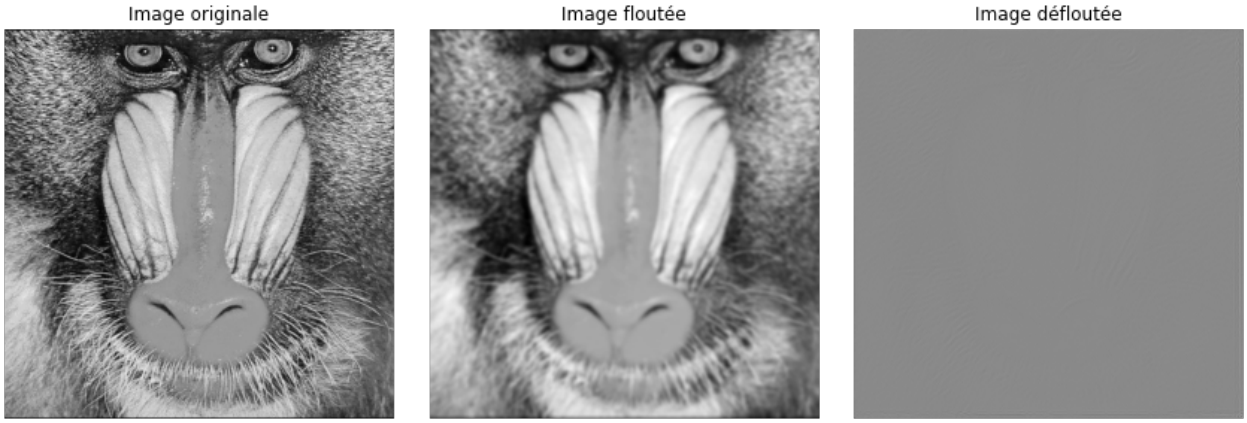
La solution de ce problème d'optimisation est donc :

$$f = (G^T G + \lambda (D_1^T D_1 + D_2^T D_2))^{-1} G^T h$$

In [337…

```python
def apply_convolution(image, kernel):
    return convolve2d(image, kernel, mode='same', boundary='wrap')

def apply_derivatives(image):
    dx = np.roll(image, -1, axis=1) - image
    dy = np.roll(image, -1, axis=0) - image
    return dx, dy

def deblur_image(h, g, lambda_reg, max_iter=100):
    P, Q = h.shape
    h_flat = h.flatten()

    # Définir l'opérateur de convolution et de dérivation
    def A(x):
        x = x.reshape(P, Q)
        conv = apply_convolution(x, g)
        dx, dy = apply_derivatives(x)
        reg = lambda_reg * (dx + dy)
        result = conv + reg
        return result.flatten()

    A_operator = LinearOperator(shape=(P * Q, P * Q), matvec=A)

    # Résoudre le système linéaire avec le gradient conjugué
    f_flat, _ = cg(A_operator, h_flat, maxiter=max_iter)

    # Reconstruction de l'image
    f = f_flat.reshape(P, Q)
    return f

def mean_squared_error(image1, image2):
    """Calcule l'erreur quadratique moyenne (MSE) entre deux images."""
    return np.mean((image1 - image2) ** 2)

if __name__ == "__main__":

    # Définir un noyau de flou
    kernel_size = 5
    g = np.outer(np.exp(-np.linspace(-2, 2, kernel_size)**2), np.exp(-np
    g = g / np.sum(g)

    # Défloutage de l'image
    lambda_reg = 0.1  # Paramètre de régularisation
    image_defloutee = deblur_image(image_floutee, g, lambda_reg)

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 3, 1)
```

```
    plt.imshow(image, cmap='gray')
    plt.title("Image originale")
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.imshow(image_floutee, cmap='gray')
    plt.title("Image floutée")
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.imshow(image_defloutee, cmap='gray')
    plt.title("Image défloutée")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    # Calcul de la qualité de la restauration
    mse_blurred = mean_squared_error(image, image_floutee)
    mse_deblurred = mean_squared_error(image, image_defloutee)

    print(f"MSE (Image floutée) : {mse_blurred:.4f}")
    print(f"MSE (Image défloutée) : {mse_deblurred:.4f}")
```




```
MSE (Image floutée) : 77.6252
MSE (Image défloutée) : 413434.6293
```