In [ ]:
```
# -*- coding: utf-8 -*-
import numpy as np
import platform
import tempfile
import os
import matplotlib.pyplot as plt
from scipy import ndimage as ndi

# necessite scikit-image
from skimage import io as skio

# POUR LA MORPHO
from skimage.segmentation import watershed
from skimage.feature import peak_local_max
```

In [ ]:
```
#%%
# VOUS DEVEZ FIXER LES DEUX VARIABLES SUIVANTES:
colaboratory=False #mettre True si vous utilisez google colab
notebook=True   # mettre True si vous utilisez un notebook local
# les seuls couples possibles sont (False,False)= travailler localement sans notebook
# (False,True): jupyternotebook local
# (True, False): google colab


assert (not (colaboratory and notebook)), "Erreur, choisissez google colab ou notebook

if colaboratory: #Si google colab on installe certaines librairies
    !pip install soundfile
    from IPython.display import Audio
    !pip install bokeh
    from bokeh.plotting import figure, output_file, show
    from bokeh.plotting import show as showbokeh
    from bokeh.io import output_notebook
    output_notebook()
    !wget https://perso.telecom-paristech.fr/ladjal/donnees_IMA203.tgz
    !tar xvzf donnees_IMA203.tgz
    os.chdir('donnees_IMA203')

if notebook: # si notebook normal dans une machine locale vous devez installer bokeh v
    from bokeh.plotting import figure, output_file, show
    from bokeh.plotting import show as showbokeh
    from bokeh.io import output_notebook
    output_notebook()
```

BokehJS 3.3.2 successfully loaded.

In [ ]:
```
#%% fonction pour voir une image

def viewimage(im,normalise=True,MINI=0.0, MAXI=255.0,titre=''):
    """ Cette fonction fait afficher l'image EN NIVEAUX DE GRIS
        dans gimp. Si un gimp est deja ouvert il est utilise.
        Par defaut normalise=True. Et dans ce cas l'image est normalisee
        entre 0 et 255 avant d'être sauvegardee.
        Si normalise=False MINI et MAXI seront mis a 0 et 255 dans l'image resultat
```

```
    """
    imt=np.float32(im.copy())
    if platform.system()=='Darwin': #on est sous mac
        prephrase='open -a /Applications/GIMP.app '
        endphrase=' &'
    elif platform.system()=='Linux': #SINON ON SUPPOSE LINUX (si vous avez un windows
        prephrase='gimp -a '
        endphrase= ' &'
    elif platform.system()=='Windows':
        prephrase='start /B "D:/GIMP/bin/gimp-2.10.exe" -a '#Remplacer D:/... par le c
        endphrase= ''
    else:
        print('Systeme non pris en charge par l affichage GIMP')
        return 'erreur d afficahge'

    if normalise:
        m=imt.min()
        imt=imt-m
        M=imt.max()
        if M>0:
            imt=imt/M

    else:
        imt=(imt-MINI)/(MAXI-MINI)
        imt[imt<0]=0
        imt[imt>1]=1

    if titre!='':
        titre='_'+titre+'_'
    nomfichier=tempfile.mktemp('TPIMA'+titre+'.png')
    commande=prephrase +nomfichier+endphrase
    skio.imsave(nomfichier,imt)
    os.system(commande)

#si on est dans un notebook (y compris dans colab), on utilise bokeh pour visualiser

usebokeh= colaboratory or notebook
if usebokeh:
    def normalise_image_pour_bokeh(X,normalise,MINI,MAXI):
        imt=np.copy(X.copy())
        if normalise:
            m=imt.min()
            imt=imt-m
            M=imt.max()
            if M>0:
                imt=imt/M

        else:

            imt=(imt-MINI)/(MAXI-MINI)
            imt[imt<0]=0
            imt[imt>1]=1
        imt*=255

        sortie=np.empty((*imt.shape,4),dtype=np.uint8)
        for k in range(3):
            sortie[:,:,k]=imt
        sortie[:,:,3]=255
        return sortie
```

```
#FONCTION viewimage universelle
import tempfile
import numpy as np
import IPython
import matplotlib.pyplot as plt

def viewimage(im, normalize=True,titre='',displayfilename=False):

    imin=im.copy().astype(np.float32)

    if normalize:
        imin-=imin.min()
    if imin.max()>0:
        imin/=imin.max()
    else:
        imin=imin.clip(0,255)/255

    imin=(imin*255).astype(np.uint8)
    filename=tempfile.mktemp(titre+'.png')

    if displayfilename:
        print (filename)
    plt.imsave(filename, imin, cmap='gray')
    IPython.display.display(IPython.display.Image(filename))
```

In [ ]:

```
#%% fonctions utiles au TP

def appfiltre(u,K):
    """ applique un filtre lineaire (en utilisant une multiplication en Fourier) """

    fft2=np.fft.fft2
    ifft2=np.fft.ifft2
    out=np.real(ifft2(fft2(u)*fft2(K)))
    return out

def degrade_image(im,br):
    """degrade une image en lui ajoutant du bruit"""
    out = im + br*np.random.randn(*im.shape)
    # Printing the mean and standard deviation of the noise
    print('Mean of the noise: ', np.mean(br*np.random.randn(*im.shape)))
    print('Standard deviation of the noise: ', np.std(br*np.random.randn(*im.shape)))
    return out

def  grady(I):
    """ Calcule le gradient en y de l'image I, avec condition de vonnewman au bord
     i.e. l'image est symétrisée et le gradient en bas est nul"""

    (m,n)=I.shape
    M=np.zeros((m,n))
    M[:-1,:]=-I[:-1,:]+I[1:,:]
    M[-1,:]=np.zeros((n,))
    return M

def  gradx(I):
    """ Calcule le gradient en y de l'image I, avec condition de vonnewman au bord
     i.e. l'image est symétrisée et le gradient a droite est nul"""

    (m,n)=I.shape
    M=np.zeros((m,n))
```

```
    M[:,:-1]=-I[:,:-1]+I[:,1:]
    M[:,-1]=np.zeros((m,))
    return M

def div(px,py):
    """calcule la divergence d'un champ de gradient"""
    """ div= - (grad)^*, i.e. div est la transposee de l'operateur gradient"""
    (m,n)=px.shape
    assert px.shape==py.shape , " px et py n'ont pas la meme taille dans div"
    Mx=np.zeros((m,n))
    My=np.zeros((m,n))

    My[1:-1,:]=py[1:-1,:]-py[:-2,:]
    My[0,:]=py[0,:]
    My[-1,:]=-py[-2,:]

    Mx[:,1:-1]=px[:,1:-1]-px[:,:-2]
    Mx[:,0]=px[:,0]
    Mx[:,-1]=-px[:,-2]
    return Mx+My

def gradient_TV(v,u,lamb):
    """ calcule le gradient de la fonctionnelle E2 du TP"""
# on n'utilise pas gradx et grady car pour minimiser
# la fonctionnelle E2 par descente de gradient nous avons choisi
# de prendre les memes conditions au bords que pour la resolution quadratique
    (sy,sx)=v.shape
    Kx=np.zeros((sy,sx))
    Ky=np.zeros((sy,sx))
    Kx[0,0]=1
    Kx[0,1]=-1
    Ky[0,0]=1
    Ky[1,0]=-1
    Kxback=np.zeros((sy,sx))
    Kyback=np.zeros((sy,sx))
    Kxback[0,0]=-1
    Kxback[0,-1]=1
    Kyback[0,0]=-1
    Kyback[-1,0]=1

    Dx=appfiltre(u,Kx)
    Dy=appfiltre(u,Ky)
    ng=(Dx**2+Dy**2)**0.5+1e-5
    div=appfiltre(Dx/ng,Kxback)+appfiltre(Dy/ng,Kyback)
    return 2*(u-v)-lamb*div

def norme_VT(I):
    """ renvoie la norme de variation totale de I"""
    (sy,sx)=I.shape
    Kx=np.zeros((sy,sx))
    Ky=np.zeros((sy,sx))
    Kx[0,0]=1
    Kx[0,1]=-1
    Ky[0,0]=1
    Ky[1,0]=-1
    Dx=appfiltre(I,Kx)
    Dy=appfiltre(I,Ky)
    ng=(Dx**2+Dy**2)**0.5
    return ng.sum()
```

```
def norme_VT_nonperiodique(u):
    gx=gradx(u)
    gy=grady(u)
    ng=((gx**2)+(gy**2))**0.5
    return ng.sum()

def norm2(x):
    return ((x**2).sum())**0.5

def projection(I,a,itmax):
    """ calcule la projection de I sur G_a
        G_a est le sous-gradient de TV en zero
        Comme vu dans le poly cette projection permet de resoudre le probleme
        de debruitage TV (E2)"""
    # ici on utilise les conditions au bord de von neuman
    # i.e. on utilise gradx et grady definis plus haut et non pas une convolution circ
    (m,n)=I.shape
    t=0.1249
    px=np.zeros((m,n))
    py=np.zeros((m,n))
    un=np.ones((m,n))

    for it in range(itmax):
        N=div(px,py)-I/a
        Gx=gradx(N)
        Gy=grady(N)
        G=(Gx**2+Gy**2)**0.5
        pxnew=(px+t*Gx)/(un+t*G)
        pynew=(py+t*Gy)/(un+t*G)
        px=pxnew
        py=pynew
    # la projection est la divergence du champ px,py
    P=a*div(px,py)
    return P

def imread(fichier):
    return np.float32(skio.imread(fichier))
```

1. Débruitage par régularisation quadratique

Le modèle d'observation est

$$v = u + b$$

où v est l'image observée (la donnée), u l'image parfaite et b le bruit. On cherche à retrouver u comme minimiseur de l'énergie

$$E_1(u) = \|u - v\|^2 + \lambda \|\nabla u\|^2$$

Le premier terme est simplement la norme au carré de la différence entre u et v. Le second est

$$\iint \|\nabla u(x,y)\|^2 \, dx \, dy$$

C'est-à-dire l'intégrale du carré du gradient de l'image en tout point.

In [ ]:
```python
def resoud_quad_fourier(K, V):
    """trouve une image im qui minimise sum_i || K_i conv im - V_i||^2
     ou les K_i et les Vi sont des filtres et des images respectivement """

    n = len(K)
    assert len(K) == len(V) , "probleme de nombre de composantes dans resoud_quad"
    (sy,sx) = K[0].shape

    numer = np.vectorize(complex)(np.zeros((sy,sx)))
    denom = np.vectorize(complex)(np.zeros((sy,sx)))
    fft2 = np.fft.fft2
    ifft2 = np.fft.ifft2

    for k in range(n):
        fV = fft2(V[k])
        fK = fft2(K[k])
        #print('type de fV',fV.dtype,' type de fK',fK.dtype)
        numer += np.conj(fK)*fV
        denom += abs(fK)**2
    return np.real(ifft2(numer/denom))



def minimisation_quadratique(v, lamb):
    """ minimise la fonctionnelle E1 du TP"""

    (sy,sx) = v.shape
    Kx = np.zeros((sy,sx))
    Ky = np.zeros((sy,sx))
    Kx[0,0] = 1
    Kx[0,1] = -1
    Ky[0,0] = 1
    Ky[1,0] = -1
    delta = np.zeros((sy,sx))

    delta[0,0] = 1.0
    s = lamb**0.5
    K = (s*Kx, s*Ky, delta)
    V = (np.zeros((sy,sx)), np.zeros((sy,sx)), v)
    return resoud_quad_fourier(K, V)
```

1. Comment utiliser l'outil `resoud_quad_fourier` pour trouver le minimiseur de cette énergie (voir le programme `minimisation_quadratique`) ?

→ The observation model is given by $v = u + b$.

To estimate u, we formulate an energy expression that quantifies how far our estimate is from the observed image and also incorporates our prior knowledge about images, such as the expectation that they are smooth. This energy expression is:

$$E_1(u) = \|u - v\|^2 + \lambda \|\nabla u\|^2$$

- The first term, $\|u - v\|^2$, is the fidelity term. It measures the Euclidean distance (L2 norm) between the estimated image u and the observed image v. We want this term to be small, which would mean that our estimate is close to the observation.

- The second term, $\lambda\|\nabla u\|^2$, is the regularization term. The gradient ∇u measures how much the image changes at each point, and by squaring its norm, we are effectively penalizing large changes, enforcing smoothness in the estimated image. The parameter λ is a regularization coefficient that controls the trade-off between the fidelity to the observed data and the smoothness of the estimated image.

On the other hand, the gradient ∇u can be approximated by discrete differences.

$K_x \approx [-1 \quad 1 \quad 0]$ and $K_y \approx \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$

Then, for the regularization term we have $\lambda\|\nabla u\|^2 = \lambda\|K_x * u\|^2 + \lambda\|K_y * u\|^2$.

And each part of the problem (Energy terms) can be expressed as follows:

$$\|\delta * u - v\|^2 = \|u - v\|^2$$

$$\lambda\|K_x * u\|^2 = \lambda\|K_x * u - 0\|^2$$

$$\lambda\|K_y * u\|^2 = \lambda\|K_y * u - 0\|^2$$

Then, the problem can be written as: $E_1(u) = \sum_{i=1}^{3} \|K_i * u - V_i\|^2$

→ Using Fourier transform we can transform the problem using the Parseval's theorem (easier to solve in this domain).

The function `resoud_quad_fourier` receives two matrices K and V constructed using K_x, K_x, v and the regularization term λ of the previous exrpession, and solve the following problem:

- Taking the derivative of the previous equation and equaling to zero we have this expression:

$$\sum \mathcal{F}(K_i)^* \cdot [\mathcal{F}(K_i) \cdot \mathcal{F}(u) - \mathcal{F}(V_i)] = 0$$

- Solving for $\mathcal{F}(u)$ we obtain:

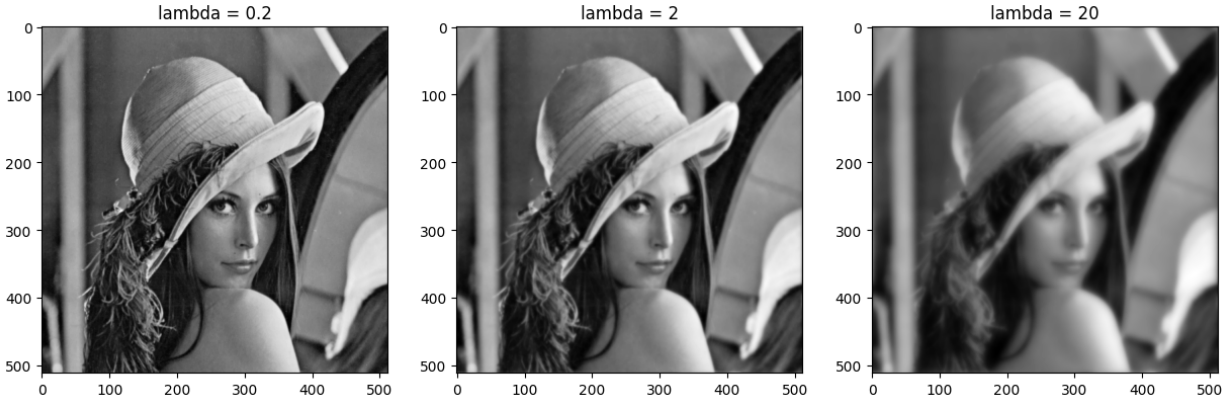
$$\mathcal{F}(u) = \frac{\sum \mathcal{F}(K_i)^* \cdot \mathcal{F}(V_i)}{\sum |\mathcal{F}(K_i)|^2}$$

- Taking the inverse fourier transform we found u Which is the return of the `resoud_quad_fourier`.

1. Décrire le résultat de ce débruitage lorsque λ est très grand ou très petit.

In [ ]:
```
imb = imread('lena.tif')

# Computing the result for lambda = 0.2, 2 and 20
u02 = minimisation_quadratique(imb, 0.2)
u2 = minimisation_quadratique(imb, 2)
```

```
u20 = minimisation_quadratique(imb, 20)

# Plotting the results in a 1x3 grid
plt.figure(figsize=(15,5))
plt.subplot(131)
plt.imshow(u02, cmap='gray')
plt.title('lambda = 0.2')
plt.subplot(132)
plt.imshow(u2, cmap='gray')
plt.title('lambda = 2')
plt.subplot(133)
plt.imshow(u20, cmap='gray')
plt.title('lambda = 20')
plt.show()
```



When applying the regularization parameter lambda in the denoising algorithm, different values of lambda will affect the balance between fidelity to the observed image and the smoothness of the estimated image.

- **Lambda = 0.2**: At this point, we prioritize fidelity to the observed image over smoothness. The resulting image retain more of the noise because the regularization term does not sufficiently penalize the fluctuations in the image, which can be caused by noise.
- **Lambda = 2**: We could guess that this value is close to the optimal value because provide a good balance between reducing noise and maintaining image details.
- **Lambda = 20**: The algorithm in this case heavily prioritize the smoothness of the image, which means that the image lost important details. The image is smoother than the others, leading to blurring and the loss of fine textures and edges.

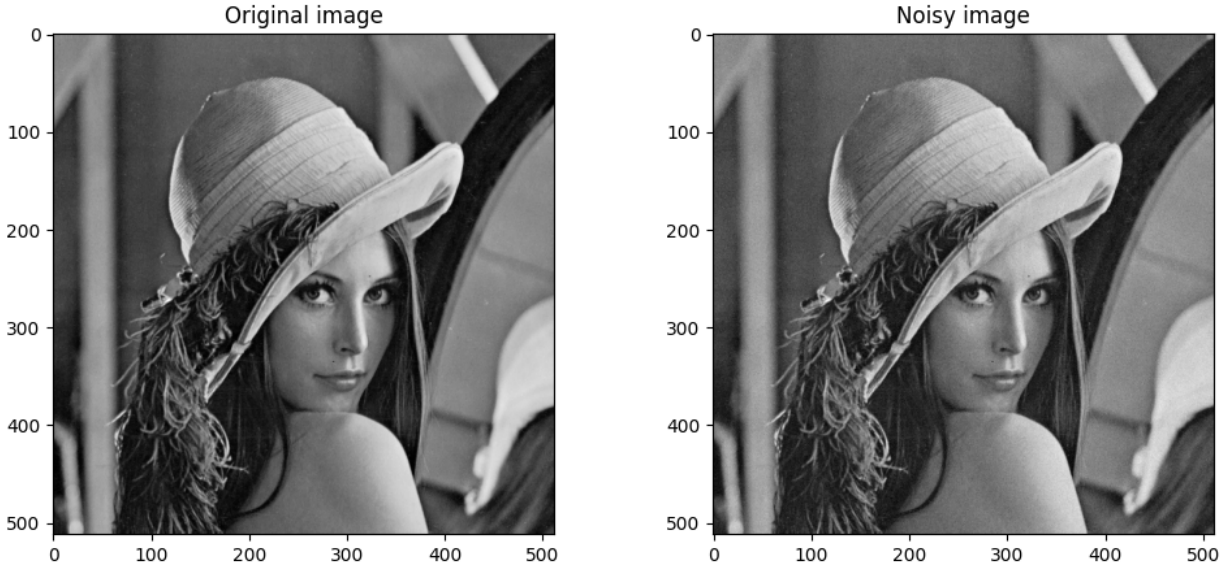
1. Après avoir ajouté un bruit d'écart type $\sigma = 5$ à l'image de lena, trouver (par dichotomie) le paramètre λ pour lequel:

$$\|\tilde{u} - v\|^2 \approx \|u - v\|^2$$

C'est-à-dire le paramètre pour lequel l'image reconstruite $\tilde{u}$ est à la même distance de l'image dégradée v que ne l'est l'image parfaite. (on respecte la norme du bruit : La norme du bruit est connue même quand on ne connaît pas l'image parfaite).

In [ ]:
```
im = imread('lena.tif')

# Adding a gaussian noise to the image of standard deviation 5
std = 5
im5 = degrade_image(im, std)

# Ploting the two images
plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.imshow(im, cmap='gray')
plt.title('Original image')
plt.subplot(122)
plt.imshow(im5, cmap='gray')
plt.title('Noisy image')
plt.show()
```

```
Mean of the noise:  0.00673259776041828
Standard deviation of the noise:  5.004912663056496
```



In [ ]:
```
# Setting lambda_low and lambda_high between 0 and 10
lambda_low = 0
lambda_high = 10

# Iterate over the mid values of lambda
for k in range(20):
    lambda_mid = (lambda_low + lambda_high) / 2
    u_tilde = minimisation_quadratique(im5, lambda_mid)
    norm_diff_u_v = norm2(u_tilde - im5)**2

    # The expected error norm2 due to the noise in the original image
    norm_diff_expected =  im.size * std

    # Compare the error norm between the denoised and noisy images to the expected err
    if (norm_diff_u_v - norm_diff_expected) > 0:
        lambda_high = lambda_mid
    else:
        lambda_low = lambda_mid

    print(f'lambda_mid = {lambda_mid}, norm_diff_u_v = {norm_diff_u_v}, norm_diff_expe
```

```
lambda_mid = 5.0, norm_diff_u_v = 48111593.04594069, norm_diff_expected = 1310720
lambda_mid = 2.5, norm_diff_u_v = 32410786.426357567, norm_diff_expected = 1310720
lambda_mid = 1.25, norm_diff_u_v = 20594553.359296568, norm_diff_expected = 1310720
lambda_mid = 0.625, norm_diff_u_v = 11990140.776598567, norm_diff_expected = 1310720
lambda_mid = 0.3125, norm_diff_u_v = 6174544.749134313, norm_diff_expected = 1310720
lambda_mid = 0.15625, norm_diff_u_v = 2724995.082090668, norm_diff_expected = 1310720
lambda_mid = 0.078125, norm_diff_u_v = 1019645.3322082158, norm_diff_expected = 13107
20
lambda_mid = 0.1171875, norm_diff_u_v = 1847934.2852426257, norm_diff_expected = 1310
720
lambda_mid = 0.09765625, norm_diff_u_v = 1423790.5665801468, norm_diff_expected = 131
0720
lambda_mid = 0.087890625, norm_diff_u_v = 1218542.208697622, norm_diff_expected = 131
0720
lambda_mid = 0.0927734375, norm_diff_u_v = 1320467.9322030842, norm_diff_expected = 1
310720
lambda_mid = 0.09033203125, norm_diff_u_v = 1269319.29829847, norm_diff_expected = 13
10720
lambda_mid = 0.091552734375, norm_diff_u_v = 1294848.6123949229, norm_diff_expected =
1310720
lambda_mid = 0.0921630859375, norm_diff_u_v = 1307647.1975304422, norm_diff_expected
= 1310720
lambda_mid = 0.09246826171875, norm_diff_u_v = 1314054.8179185768, norm_diff_expected
= 1310720
lambda_mid = 0.092315673828125, norm_diff_u_v = 1310850.318280488, norm_diff_expected
= 1310720
lambda_mid = 0.0922393798828125, norm_diff_u_v = 1309248.585205126, norm_diff_expecte
d = 1310720
lambda_mid = 0.09227752685546875, norm_diff_u_v = 1310049.4086101819, norm_diff_expec
ted = 1310720
lambda_mid = 0.09229660034179688, norm_diff_u_v = 1310449.8526674826, norm_diff_expec
ted = 1310720
lambda_mid = 0.09230613708496094, norm_diff_u_v = 1310650.0827801842, norm_diff_expec
ted = 1310720
```

1. Écrire un algorithme pour trouver le paramètre λ tel que $\|\tilde{u} - u\|^2$ soit minimale. (dans le cadre de ce TP on connaît l'image parfaite u, en général on ne la connaît pas). Commentaires ?

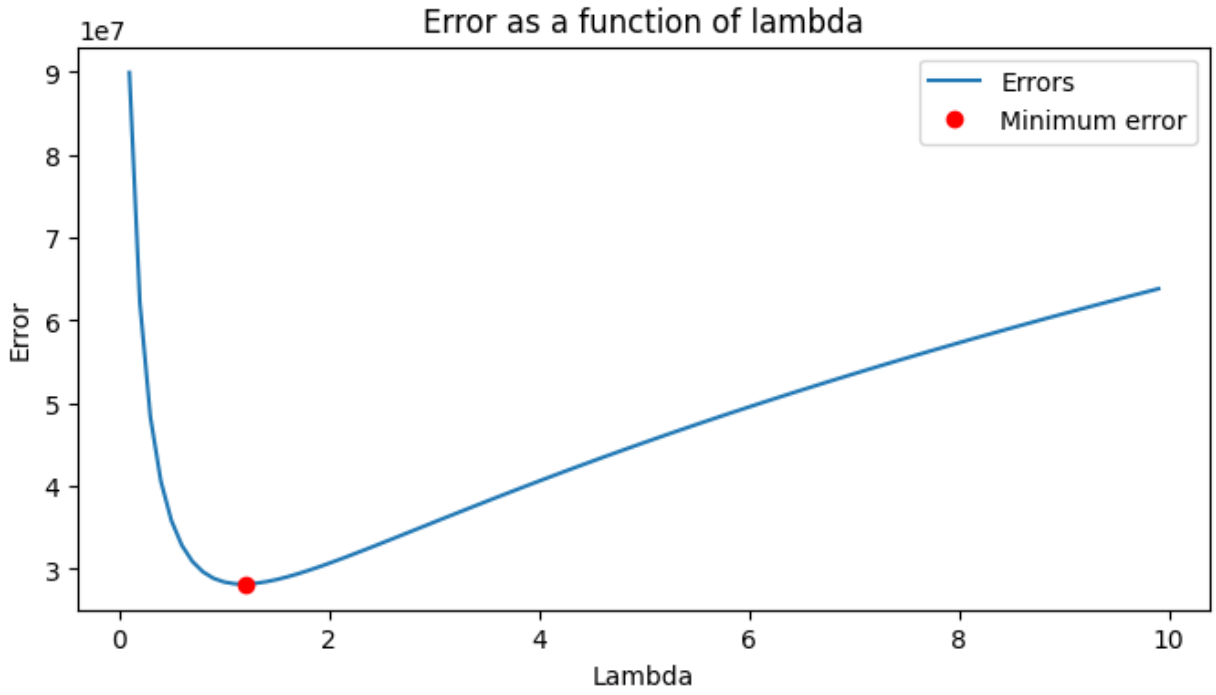
In [ ]:

```python
myim = imread('lena.tif')

# Minimization of E1 using minimisation_quadratique and resoud_quad_fourier ----------

# Setting up a vector of lambdas
lambdas = np.arange(0.1, 10, 0.1)
# Setting up a vector of errors
errors = np.zeros(len(lambdas))

# Applying minimisation_quadratique for each lambda
for i in range(len(lambdas)):
    u = minimisation_quadratique(imb, lambdas[i])
    errors[i] = norm2(u - myim)**2

# Taking the minimum of the errors
min_error = errors.min()
lambda_min = lambdas[errors.argmin()]

# Printing the results
```

```
print('Lambda minimum: ', lambda_min)
print('Minimum error: ', min_error)
```

```
Lambda minimum:  1.2000000000000002
Minimum error:  28164753.300948977
```

In [ ]:
```
# Plotting the errors as a function of lambda
plt.figure(figsize=(8, 4))
plt.plot(lambdas, errors)
plt.plot(lambdas[errors.argmin()], min_error, 'ro')
plt.legend(["Errors", "Minimum error"])
plt.title("Error as a function of lambda")
plt.xlabel("Lambda")
plt.ylabel("Error")
plt.show()
```



Initially, as λ increases from zero, there is a significant decrease in error, suggesting that the introduction of regularization is effective at reducing overfitting to noisy data. The steep slope of the curve leading to the minimum error point illustrates the sensitivity of the model to regularization: even small increments in λ can lead to large improvements in performance.

Upon reaching the minimum error, the trend reverses; as λ continues to increase, the error begins to rise. This portion of the curve represents the regime where the regularization starts to dominate the error metric, leading to an over-smoothing of the image.

2. Débruitage par variation totale

Dans cette section on utilise la variation totale comme terme de régularisation. Cela donne l'énergie

$$E_2(u) = \|u - v\|^2 + \lambda \|\nabla u\|_1$$

Le second terme est

$$\iint \|\nabla u(x,y)\|\, dx\, dy$$

In [ ]:
```
def E2_nonperiodique(u, v, lamb): # renvoie l'énergie E2
    return lamb*norme_VT_nonperiodique(u)+norm2(u-v)**2

def gradient_TV_nonperiodique(v, u, lamb):
     """ calcule le gradient de la fonctionnelle E2 du TP"""
     gx = gradx(u)
     gy = grady(u)
     ng = ((gx**2)+(gy**2))**0.5+1e-5
     dive = div(gx/ng,gy/ng)
     return 2*(u-v) - lamb*dive

def minimise_TV_gradient(v, lamb, pas, nbpas):
    """ minimise E2 par descente de gradient a pas constant """
    u = np.zeros(v.shape)
    Energ = np.zeros(nbpas)
    for k in range(nbpas):
        # print(k)
        Energ[k] = E2_nonperiodique(u, v, lamb)
        u = u - pas*gradient_TV_nonperiodique(v, u, lamb)
    return (u, Energ)

def vartotale_Chambolle(v, lamb, itmax=100):
    """ Trouve une image qui minimise lamb*TV(I)+||I-v||^2
    en utilisant la projection dur G_a"""
    (m, n) = v.shape
    P = projection(v, lamb/2, itmax)
    return v - P
```

2.1 Descente de gradient

La première idée pour minimiser cette fonctionnelle est d'utiliser une descente de gradient. Nous allons voir que cela peut entraîner des problèmes numériques.

Le gradient de la fonctionnelle E_2 est donné par

$$\nabla E_2(u) = 2(u-v) - \lambda \operatorname{div}\left(\frac{\nabla u}{\|\nabla u\|}\right)$$

Il est calculé par la fonction `gradient_TV`. Utiliser le programme `minimise_TV_gradient` pour différentes valeurs du pas de descente. Atteignez-vous toujours le même minimum d'énergie? (le programme renvoie l'évolution de l'énergie).

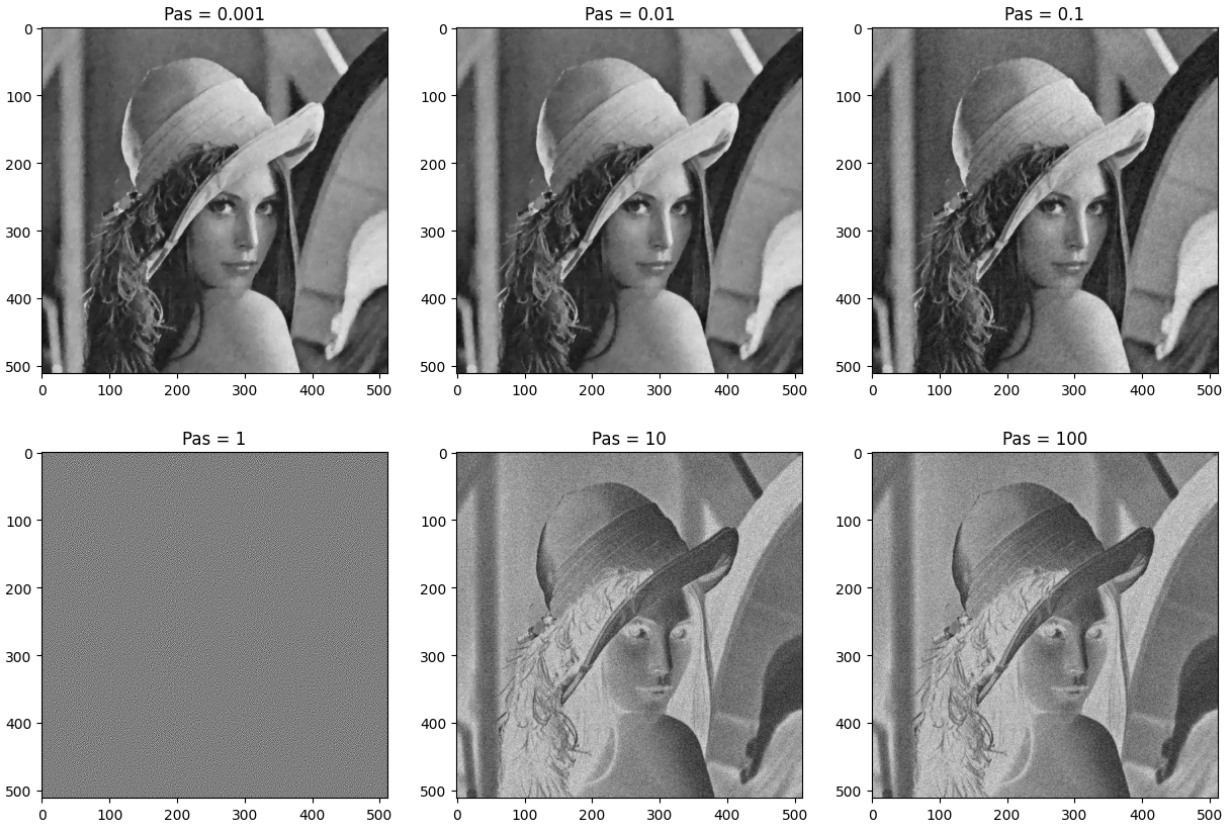
In [ ]:
```
# Minimization of E2 using minimise_TV_gradient
myim = imread('lena.tif')
imb = degrade_image(myim, 25)
lambda_ = 40
num_steps = 50

# Computing the result for differents step sizes
(u0001, en0001) = minimise_TV_gradient(imb, lambda_, 0.001, num_steps)
```

```
(u001, en001) = minimise_TV_gradient(imb, lambda_, 0.01, num_steps)
(u01, en01) = minimise_TV_gradient(imb, lambda_, 0.1, num_steps)
(u1, en1) = minimise_TV_gradient(imb, lambda_, 1, num_steps)
(u10, en10) = minimise_TV_gradient(imb, lambda_, 10, num_steps)
(u100, en100) = minimise_TV_gradient(imb, lambda_, 100, num_steps)
```

```
Mean of the noise:  0.02857484860578544
Standard deviation of the noise:  25.051936343455985
```

In [ ]:

```
# Plotting the results in a 2x3 grid
plt.figure(figsize=(15,10))
plt.subplot(231)
plt.imshow(u0001, cmap='gray')
plt.title('Pas = 0.001')
plt.subplot(232)
plt.imshow(u001, cmap='gray')
plt.title('Pas = 0.01')
plt.subplot(233)
plt.imshow(u01, cmap='gray')
plt.title('Pas = 0.1')
plt.subplot(234)
plt.imshow(u1, cmap='gray')
plt.title('Pas = 1')
plt.subplot(235)
plt.imshow(u10, cmap='gray')
plt.title('Pas = 10')
plt.subplot(236)
plt.imshow(u100, cmap='gray')
plt.title('Pas = 100')
plt.show()
```



In the ideal case, given the convexity of the energy function $E_2(u) = \|u - v\|^2 + \lambda\|\nabla u\|_1$, the optimization process should converge to the same global minimum regardless of the

regularization parameter λ. Convexity ensures that any local minimum is also a global minimum, and as such, the solution space does not contain the traps of local minima that are suboptimal.

However, practical considerations can complicate this ideal scenario. The discrete approximation of the total variation term, the handling of image boundaries, the choice and tuning of the regularization parameter λ, and the non-differentiability of the L1 norm at zero introduces numerical challenges.

Furthermore, factors such as initialization, step size in iterative methods, and computational precision can all lead to variations in the results. As a consequence, even though the underlying problem is convex, the practical application of the algorithm might not always converge to the same global minimum due to these real-world complexities.

2.2 Projection Chambolle

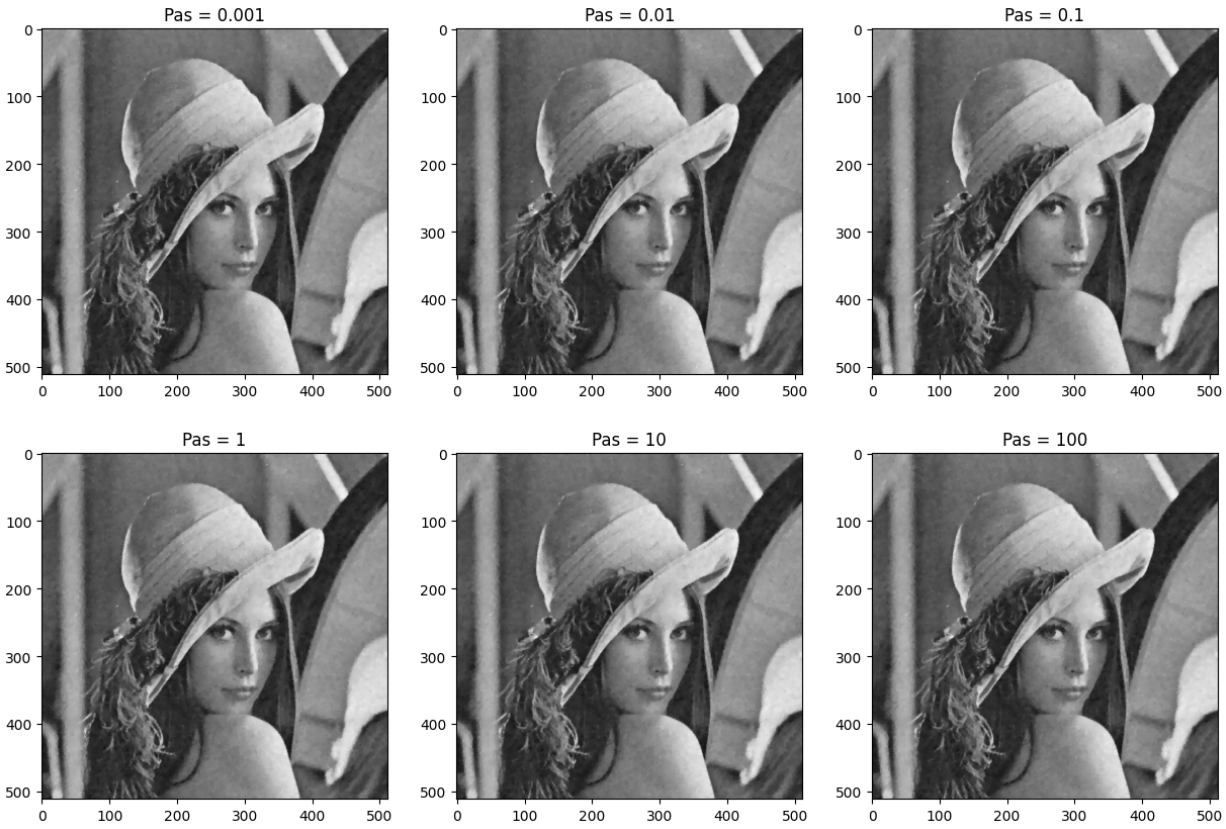
Le programme `vartotale_Chambolle` applique la méthode de Chambolle (expliquée dans le polycopié) au même problème posé par E_2. Utilisez ce programme et que constatez-vous quant à la vitesse de cette algorithme et sa précision (minimisation effective de E_2) par rapport à la descente de gradient. Dans la suite vous n'utiliserez plus que cette technique pour minimiser la fonctionnelle E_2.

In [ ]:
```
# Chambole projection using var_totale_Chambolle
myim = imread('lena.tif')
imb = degrade_image(myim, 25)
lambda_ = 40
num_steps = 50

# Computing the result for differents step sizes
u0001 = vartotale_Chambolle(imb, lambda_, num_steps)
u001 = vartotale_Chambolle(imb, lambda_, num_steps)
u01 = vartotale_Chambolle(imb, lambda_, num_steps)
u1 = vartotale_Chambolle(imb, lambda_, num_steps)
u10 = vartotale_Chambolle(imb, lambda_, num_steps)
u100 = vartotale_Chambolle(imb, lambda_, num_steps)
```

```
Mean of the noise:  -0.010694351906084141
Standard deviation of the noise:  24.984666562347766
```

In [ ]:
```
# Plotting the results in a 2x3 grid
plt.figure(figsize=(15,10))
plt.subplot(231)
plt.imshow(u0001, cmap='gray')
plt.title('Pas = 0.001')

plt.subplot(232)
plt.imshow(u001, cmap='gray')
plt.title('Pas = 0.01')

plt.subplot(233)
plt.imshow(u01, cmap='gray')
plt.title('Pas = 0.1')

plt.subplot(234)
```

```
plt.imshow(u1, cmap='gray')
plt.title('Pas = 1')

plt.subplot(235)
plt.imshow(u10, cmap='gray')
plt.title('Pas = 10')

plt.subplot(236)
plt.imshow(u100, cmap='gray')
plt.title('Pas = 100')

plt.show()
```



1. **Speed of the Algorithms**:
    - Chambolle's method is usually quicker than regular gradient descent because it moves more smoothly towards the solution.
    - Ordinary gradient descent can be slow in general cases.
    - Chambolle's algorithm is smart about handling the unique parts of the TV norm, so it can take bigger steps without missing the target, leading to quicker results.
2. **How Well They Minimize**:
    - Chambolle's algorithm tends to do a better job at reducing the value of E_2 because it cleverly deals with parts of the TV norm that are tricky for standard methods.
    - Normal gradient descent might not do as well with the rough spots of the TV norm, which can make it less precise unless it's adjusted just right.
    - Chambolle's method keeps the solution in the right zone with each step, making sure it's always heading towards a more exact answer.

3 Comparaison

Après avoir fixé une image bruitée par un bruit de 25. Trouver pour chacune des deux méthodes (TV et quadratique) le meilleur paramètre λ et comparez qualitativement le résultat obtenu par les deux méthodes pour le débruitage.

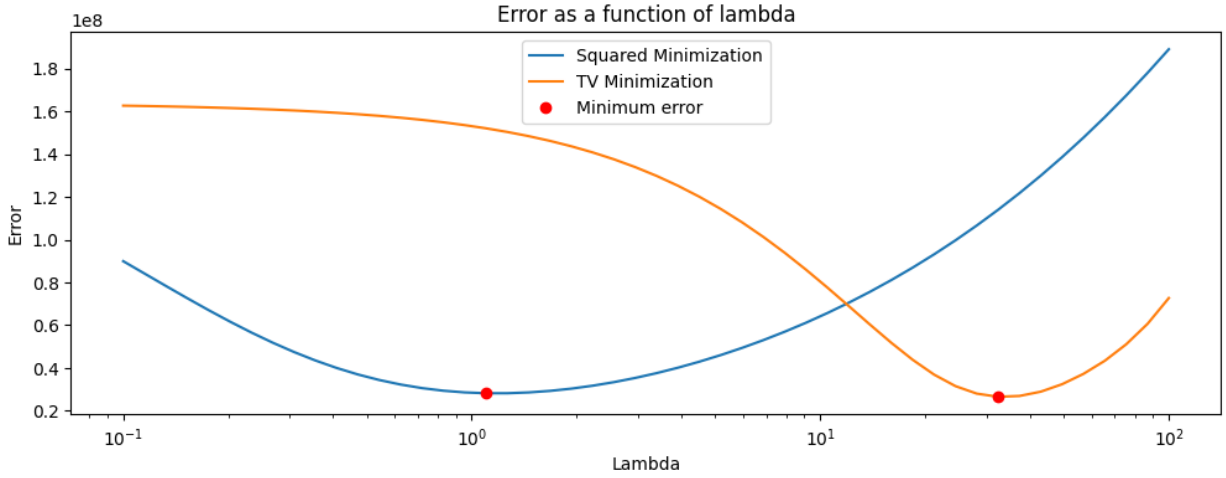
In [ ]:
```
myim = imread('lena.tif')

# Setting up a vector of lambdas in a logarithmic scale
lambdas = np.logspace(-1, 2, 50)

# For squared minimization ----------------------------------------------------
# Setting up a vector of errors
errors_quad = np.zeros(len(lambdas))

# Applying minimization_quadratique for each lambda
for i in range(len(lambdas)):
    u = minimisation_quadratique(imb, lambdas[i])
    errors_quad[i] = norm2(u - myim)**2

# Taking the minimum of the errors
min_error_quad = errors_quad.min()
lambda_min_quad = lambdas[errors_quad.argmin()]

# For TV Minimization ---------------------------------------------------------
# Setting up a vector of errors
errors_TV = np.zeros(len(lambdas))

# Applying minimise_TV_gradient for each lambda
for i in range(len(lambdas)):
    u = minimise_TV_gradient(imb, lambdas[i], 0.1, 100)[0]
    errors_TV[i] = norm2(u - myim)**2

# Taking the minimum of the errors
```

```
min_error_TV = errors_TV.min()
lambda_min_TV = lambdas[errors_TV.argmin()]
```

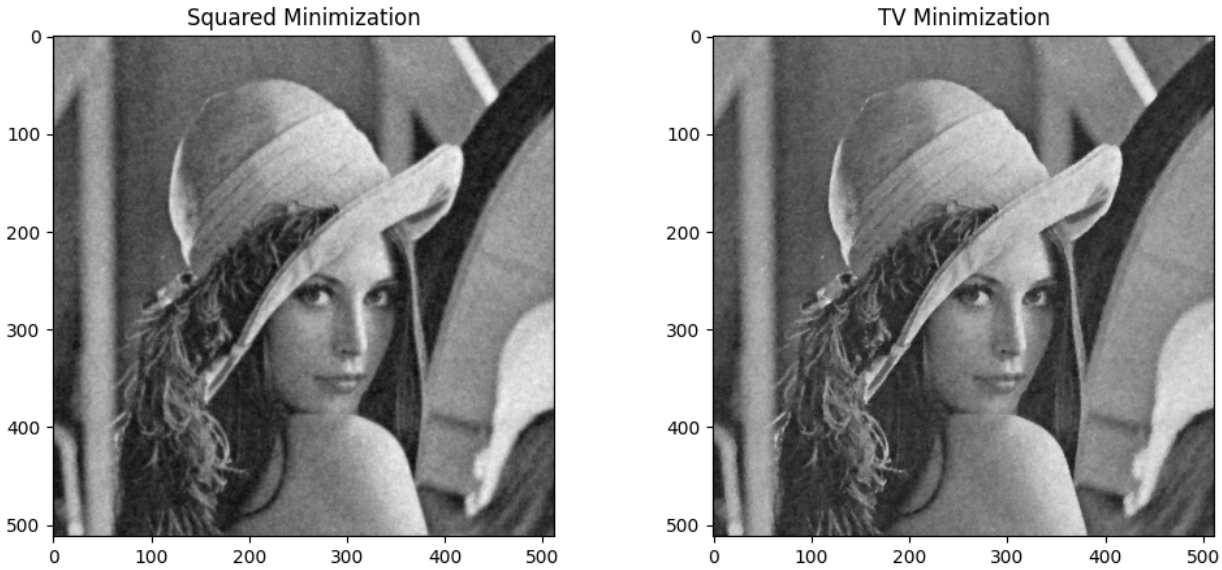
In [ ]:
```
# Printing the results
print('Lambda minimum (Squared Minimization): ', lambda_min_quad)
print('Minimum error (Squared Minimization): ', min_error_quad)
print('Lambda minimum (TV Minimization): ', lambda_min_TV)
print('Minimum error (TV Minimization): ', min_error_TV)
```

```
Lambda minimum (Squared Minimization):  1.0985411419875584
Minimum error (Squared Minimization):  28207119.51734165
Lambda minimum (TV Minimization):  32.374575428176435
Minimum error (TV Minimization):  26546418.062729347
```

In [ ]:
```
# Plotting the two errors as a function of lambda in the same logarithmic plot
plt.figure(figsize=(12, 4))
plt.semilogx(lambdas, errors_quad)
plt.semilogx(lambdas, errors_TV)
plt.semilogx(lambdas[errors_quad.argmin()], min_error_quad, 'ro')
plt.semilogx(lambdas[errors_TV.argmin()], min_error_TV, 'ro')
plt.legend(["Squared Minimization", "TV Minimization", "Minimum error"])
plt.title("Error as a function of lambda")
plt.xlabel("Lambda")
plt.ylabel("Error")
plt.show()
```



In [ ]:
```
# Plotting the two images for the minimum lambda in each case in a 1x2 grid
plt.figure(figsize=(12,5))
plt.subplot(121)
plt.imshow(minimisation_quadratique(imb, lambda_min_quad), cmap='gray')
plt.title('Squared Minimization')
plt.subplot(122)
plt.imshow(minimise_TV_gradient(imb, lambda_min_TV, 0.1, 100)[0], cmap='gray')
plt.title('TV Minimization')
plt.show()
```



Squared Minimization Image:

- Likely exhibits smoother regions, as squared minimization tends to average out the pixel values, which result in loss of fine details.
- Edges may appear less sharp because the L2 norm does not specifically preserve edges.
- Noise reduction is generally good, but at the cost of making the image blurrier.

TV Minimization Image:

- Edges and fine details are better preserved due to the nature of TV minimization, which is designed to reduce noise while keeping edges crisp.
- Within certain regions the image is smooth, but the boundaries of these regions (edges) are maintained.

4 Déconvolution avec variation totale

Cette partie est optionnelle et ne peut pas être terminée pendant le temps du TP. Ici nous comparons la déconvolution par variation totale avec la déconvolution quadratique. D'abord on donne la formule de l'observation :

$$v = Au_0 + b$$

où u_0 est l'image parfaite, A est une convolution contre un noyau K et b un bruit blanc de puissance σ^2.

On considère les deux énergies de restauration. La première est quadratique, elle utilise comme régularisation l'énergie quadratique du champ de gradients.

$$E_q(u) = \|Au - v\|^2 + \lambda \|\nabla u\|_2^2$$

La seconde E_{TV} utilise la variation totale comme régularisation

$$E_{TV}(u) = \|Au - v\|^2 + \lambda \|\nabla u\|_1$$

La première se résout avec la fonction `resoud_quad_fourier`. Pour la seconde on utilise un algorithme de split en écrivant une nouvelle énergie qui dépend de deux variables. La première est l'image u recherchée. La seconde est d, un champ de vecteur qui représente le gradient de u. La manière la plus simple d'utiliser cette idée serait de minimiser l'énergie

$$E_{\text{split}}(u,d) = \|Au - v\|^2 + \lambda\|d\|_1 + \gamma\|\nabla u - d\|^2$$

Lorsque le paramètre γ devient grand le couple qui minimise cette énergie va vérifier $\nabla u = d$ et alors u minimise E_{TV}. L'avantage de cette énergie est que la minimisation par rapport à u (avec d fixe) l'énergie est quadratique et se résout facilement (Fourier). Si u est fixée la minimisation par rapport à d est facile aussi car

$$E_{\text{split}}(u = \text{cst}, d) = \text{cst} + \sum_x (\gamma\|\nabla u(x) - d(x)\|^2 + \lambda\|d(x)\|)$$

Pour chaque x (x est une position) on a une seule variable, le vecteur $d(x)$. Or le minimiseur t (est un vecteur inconnu et w un vecteur connu constant) de

$$\|t\| + \alpha\|t - w\|^2$$

est

$$t = \beta w \text{ et } \beta = \begin{cases} 0 & \text{si } \|w\| \leq \frac{1}{2\alpha} \\ 1 - \frac{1}{2\alpha\|w\|} & \text{sinon} \end{cases} \quad (1)$$

En prenant $\alpha = \gamma/\lambda$ et $w = \nabla u(x)$ et $t = d(x)$ on trouve une manière de minimise E_{split} par rapport à d pour u fixé et c'est un algorithme rapide (quelques opérations pour chaque point x).

Malheureusement, l'énergie E_{split} implique d'augmenter le paramètre γ sans vraiment savoir si γ est devenu assez grand pour forcer l'égalité $\nabla u = d$.

Une énergie alternative est la suivante, elle fait intervenir un autre champ de vecteurs b_n

$$E^n_{\text{split}}(u,d) = \gamma\|d - \nabla u - b_n\|^2 + \lambda\|d(i,j)\|_1 + \|Au - v\|^2$$

Ici γ est fixée et ce que l'on fait est de modifier b_n à chaque étape pour donner b_{n+1} ainsi au lieu de faire tendre γ vers l'infini on laisse le champ b_n évoluer de manière à forcer $\nabla u = d$.

Voici l'algorithme :

1. $n = 0, b_0 = 0$ et $d = 0$ Boucler sur les étapes (a) Minimiser E^n_{split} par rapport à u. (b) Garder le u obtenu et minimiser E^n_{split} par rapport à d (c) Faire $b_{n+1} = b_n + \nabla u - d$ et incrémenter n. Boucler

La démonstration de convergence de cet algorithme est difficile mais remarquons que s'il atteint une limite u, d, alors d ne doit plus évoluer mais cela implique que b_n n'évolue plus (car d est proportionnel en tout point à $b_n + \nabla u$, si b_n change alors d aussi). Mais pour que b_n n'évolue plus il faut que $\nabla u - d$ soit nul.

4.1 Mise en application

Pour tester cette méthode il faut implémenter une fonction qui prend un champ de vecteur w et renvoie un champ de vecteur t comme dans l'équation (1). On appelle cette fonction `softthresh(w, \alpha)`.

Une autre chose à laquelle il faut faire attention est que la convolution doit être appliquée par une multiplication en Fourier ce qui revient à supposer que l'image est périodique. On appelle `convol` la fonction convolution on fait ceci :

Partie préparation: Prendre une image parfaite et lui appliquer `convol` et ajouter du bruit. Typiquement on prend un flou de noyau une gaussienne de largeur 1,5 pixels et un bruit de 2 ou 3 (pour une image de 255 niveaux de gris).

algorithme Commencer par $b_0 = 0$ et $d = 0$. Utiliser `resoud_quad_fourier` pour minimiser E^n_{split} par rapport à u. Utiliser `softthresh(w, \alpha)` pour minimiser E_{split} par rapport à d. Modifier b_n, comme en (c) plus haut. Recommencer.

Dans cet article en ligne http://www.ipol.im/pub/art/2012/g-tvdc/, les auteurs expliquent qu'une bonne valeur à laquelle fixer γ est 5. Mais dans cet article ils manipulent des images à valeurs réelles entre 0 et 1. Expliquer pourquoi un bon paramètre γ pour les images entre 0 et 255 est $5/(255^2)$.

Plus généralement, si on suppose que pour v et σ (le bruit) données les bons paramètres sont λ_0 et γ_0, quels sont les bons paramètres pour $v' = 255v$ et $\sigma' = 255\sigma$.

In [ ]:
```python
import numpy as np
import skimage.io as skio

# Assuming you have the necessary functions like appfiltre, degrade_image, etc., alrea

def total_variation_deconvolution(im, br, lambda_tv, itmax, gamma):
    # Preparing the degraded image
    degraded_im = degrade_image(im, br)

    # Gaussian blur kernel (for example)
    kernel = np.array([[1, 4, 6, 4, 1], [4, 16, 24, 16, 4], [6, 24, 36, 24, 6], [4, 16

    # Start the algorithm
    u = np.copy(degraded_im)  # Initialize u with the degraded image
    for _ in range(itmax):
        # Step 1: Minimize E_split^n with respect to u using minimisation_quadratique
        K = [kernel]
        V = [degraded_im]
        u = minimisation_quadratique(u, gamma)

        # Step 2: Apply the projection for total variation
        u = projection(u, lambda_tv, itmax)

    return u
```

```
# Example usage
im = imread('lena.tif')  # Original image
br = 5  # Noise level
lambda_tv = 1  # Lambda for TV
itmax = 100  # Number of iterations
gamma = 5 / (255 ** 2)  # Gamma value

# Perform the total variation deconvolution
result_im = total_variation_deconvolution(im, br, lambda_tv, itmax, gamma)
```

```
Mean of the noise:  0.010300397460105028
Standard deviation of the noise:  4.996418002174816
```

In [ ]:

```
# Plot the results
plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.imshow(im, cmap='gray')
plt.title('Original image')
plt.subplot(122)
plt.imshow(result_im, cmap='gray')
plt.title('Deconvolved image')
plt.show()
```