tp-ima203-variatio

January 19, 2024

[1]:
```
# -*- 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.morphology import watershed
from skimage.feature import peak_local_max
```

[2]:
```
#%%
# VOUS DEVEZ FIXER LES DEUX VARIABLES SUIVANTES:
colaboratory=False #mettre True si vous utilisez google colab
notebook= True   # mettre Trus 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 local mais pas les deux en meme temps"

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 vous-meme
    from bokeh.plotting import figure, output_file, show
    from bokeh.plotting import show as showbokeh
    from bokeh.io import output_notebook
    output_notebook()
```

[3]:
```
#%% fonction pour voir une image
#FONCTION viewimage universelle
import IPython
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))

#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
```

[4]:
```
#%% 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)
    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 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 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)

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


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):
        Energ[k]=E2_nonperiodique(u,v,lamb)
        u=u-pas*gradient_TV_nonperiodique(v,u,lamb)
    return (u,Energ)


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 circulaire
    (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 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


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

[5]:
```
#%% lire une image
im=imread('lena.tif') #ATTENTION IL FAUT ETRE DANS LE BON REPERTOIRE (utiliser
 ↪os.chdir())
```

[6]:
```
# voir l'image
viewimage(im,titre='ORIGINALE')
#degrader une image

imb=degrade_image(im,25)

# voir l'image bruitée
viewimage(imb,titre='BRUITEE')
```

[7]:
```
#%% restauration quadratique : exemple
lamb= 10
restau=minimisation_quadratique(imb,lamb)
viewimage(restau,titre='RESTQUAD_LAMB='+str(lamb))
```

0.0.1 Partie 1 : Débruitage par régularisation quadratique

Question 1
Question 2
Lorsque λ est très grand, le terme de régularisation est très grand et on voit une image très floue.
Lorsque λ est très petit, le terme de régularisation est très petit et l'image reste bruitée.
Question 3
On trouve un lambda qui se rapproche de 0.3.

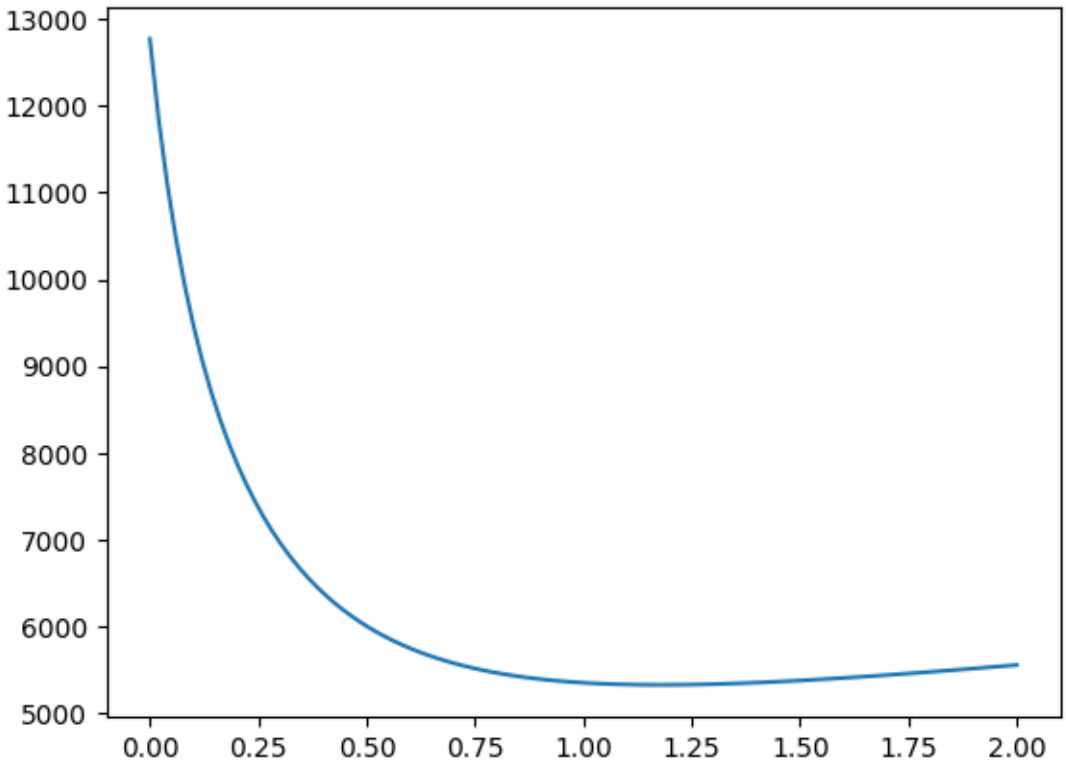
[8]:
```
# Question 3 : ajout d'un bruit de variance sigma à l'image lena
#trouver le paramètre lambda par dichotomie pour que l'image reconstruite soit␣
 ↪à égale distance de l'image bruitée et de l'image originale
def dichotomie(im,bruit,eps):
    a=0
    b=10
    while b-a>eps:
        lamb=(a+b)/2
```

```
        restau=minimisation_quadratique(bruit,lamb)
        if norm2(restau - im)>norm2(bruit - restau):
            a=lamb
        else:
            b=lamb
    return lamb,restau

lambda_dicho,restau=dichotomie(im,imb,0.001)
print(lambda_dicho)
viewimage(restau,titre='RESTQUAD_LAMB='+str(lambda_dicho))
print(norm2(restau - im))
print(norm2(imb - restau))
```

```
0.2960205078125
```

```
6991.7524099718485
6999.739004862471
```

[9]:
```
def best_lambda(im,bruit):
    lamb = np.linspace(0,2,100)
    norme = np.zeros(100)
    for i in range(100):
        restau=minimisation_quadratique(bruit,lamb[i])
        norme[i]=norm2(restau - im)
    plt.plot(lamb,norme)
    plt.show()
    return lamb[np.argmin(norme)], minimisation_quadratique(bruit,lamb[np.
 ↪argmin(norme)])

lambda_best,restau=best_lambda(im,imb)
print(lambda_best)
```



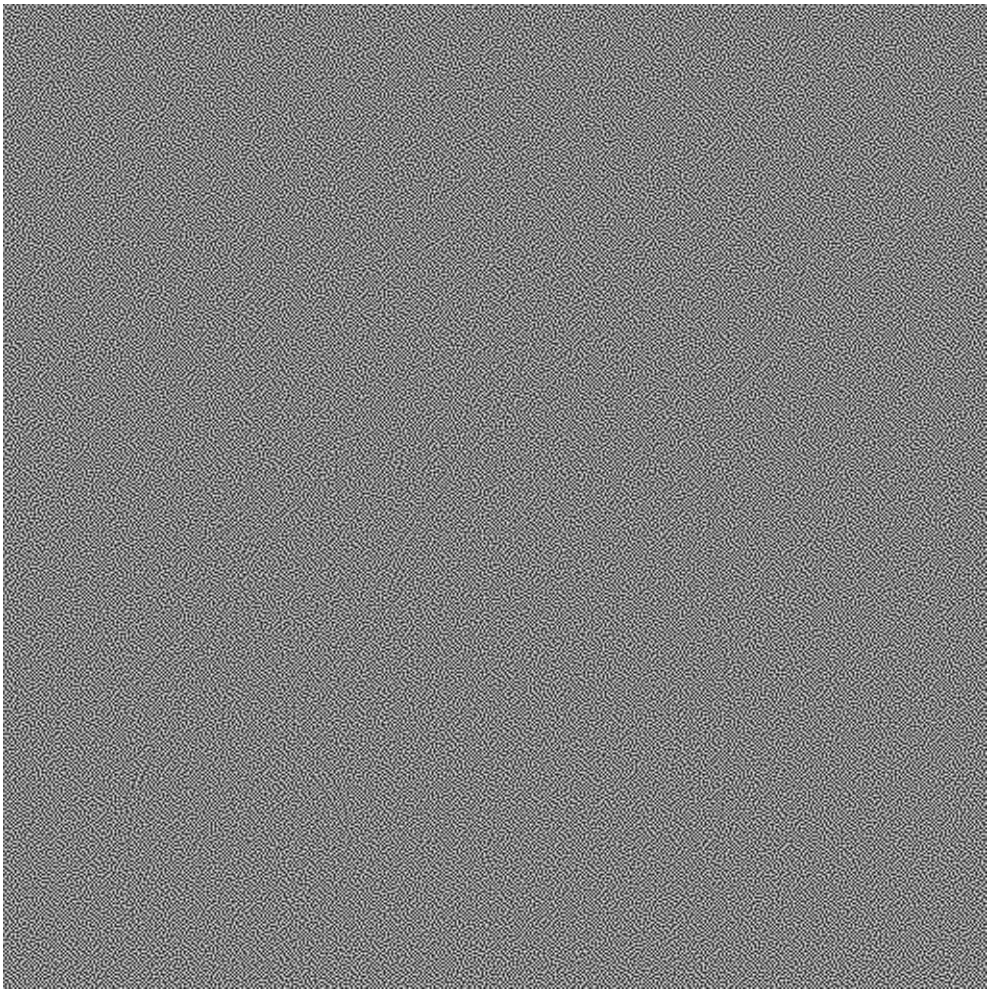

```
1.1717171717171717
```

[10]:
```
viewimage(restau,titre='RESTQUAD_LAMB='+str(lambda_best))
```

0.0.2 2 Débruitage par variation totale

2.1 Descente de gradient

[11]:
```
#%%
# u1 = minimise_TV_gradient(imb,lamb,pas,nbpas)
u1,en1=minimise_TV_gradient(imb, 40, 1, 20)
u05,en05=minimise_TV_gradient(imb, 40, 0.5, 20)
u01,en01=minimise_TV_gradient(imb, 40, 0.1, 20)
#u05inf,en05inf=minimise_TV_gradient(imb, 10, 0.5, 200)
viewimage(u1,titre='RESTTV_LAMB='+str(40)+'_pas='+str(1))
viewimage(u05,titre='RESTTV_LAMB='+str(40)+'_pas='+str(0.5))
viewimage(u01,titre='RESTTV_LAMB='+str(40)+'_pas='+str(0.1))
print(min(en1))
print(min(en05))
print(min(en01))
```



```
3467736496.6853724
477140846.41234016
264380486.9568906
```

On atteint pas les mêmes minimums quand le pas change

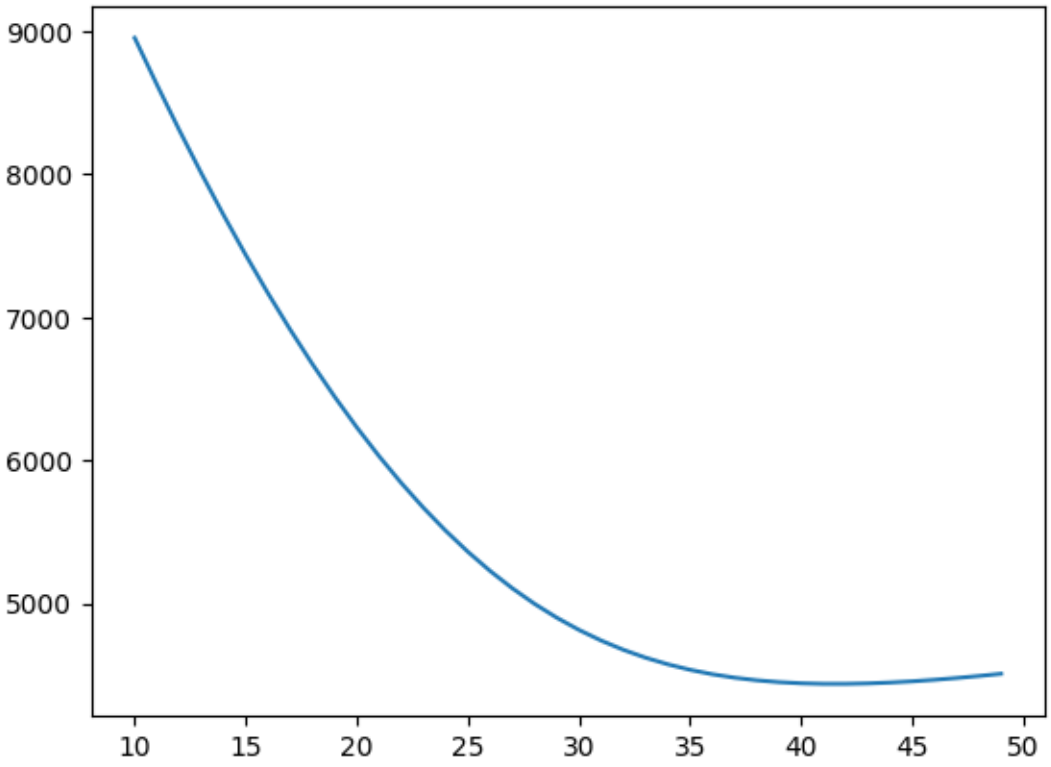
2.2 Projection Chambolle

[12]:
```
restau = vartotale_Chambolle(imb,50)
viewimage(restau,titre='RESTTV_LAMB='+str(50))
print(norm2(restau - im))
```

4524.561322627299

[13]:
```
def best_lamb_chambolle(im,bruit):
    lamb = np.linspace(10,49,40)
    norme = np.zeros(40)
    for i in range(40):
        restau = vartotale_Chambolle(bruit, int(lamb[i]))
        norme[i] = norm2(restau - im)

    return lamb[np.argmin(norme)], restau,norme,lamb

lamb,restau,norm,list_lamb=best_lamb_chambolle(im,imb)


plt.plot(list_lamb,norm)
plt.show()
viewimage(restau,titre='RESTTV_LAMB='+str(lambda_best))
```

```
print(norm2(restau - im))
print(lamb)
```

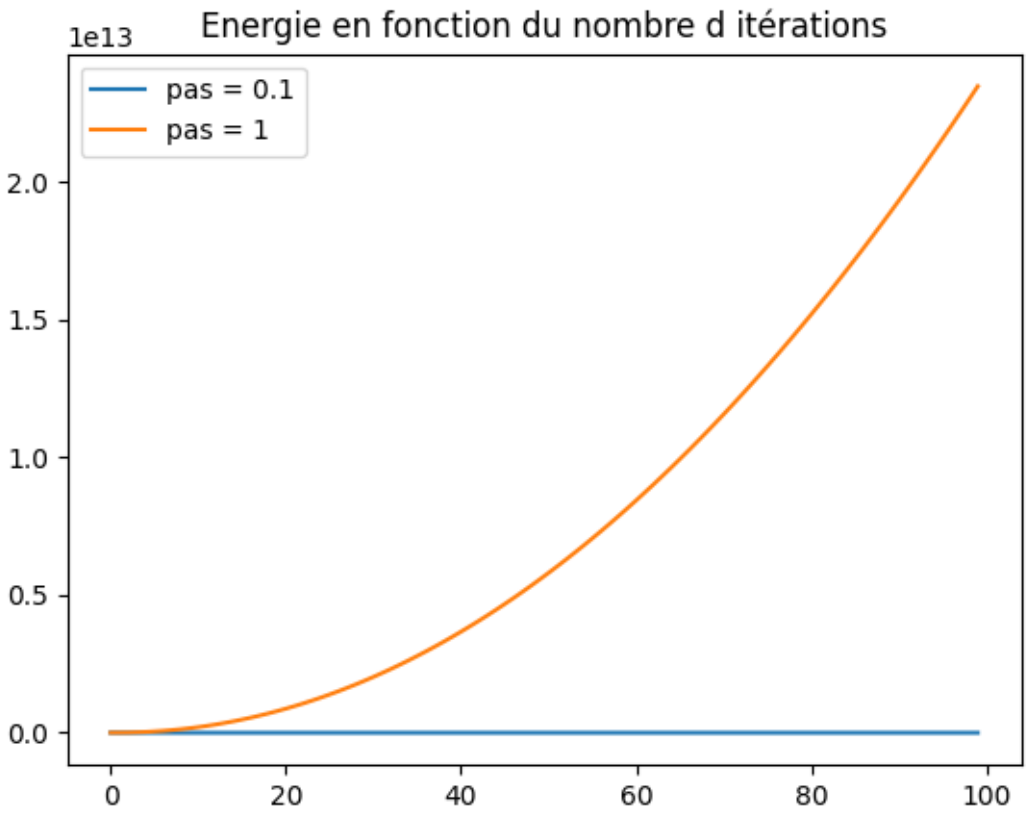


```
4507.847092894286
42.0
```

Ce programme est lent, un peu moins de 5s par itération mais il est plus précis que la descente de gradient. On peut voir que l'image est plus nette et que le bruit est mieux supprimé.

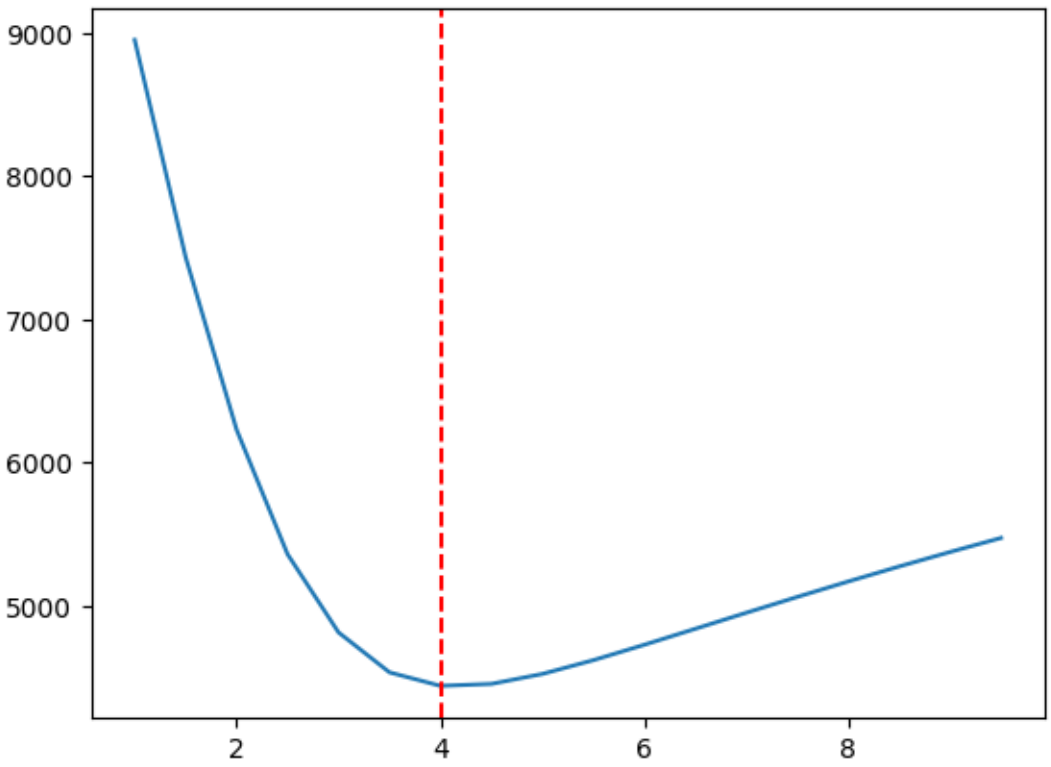
[14]:
```
(u,energ)=minimise_TV_gradient(imb,40,0.1,100)    # pas = 0.1
(u,energ2)=minimise_TV_gradient(imb,40,1,100)       # pas = 1
plt.plot(energ, label='pas = 0.1')
plt.plot(energ2, label='pas = 1')
plt.legend()
plt.title('Energie en fonction du nombre d itérations')
```

[14]:
```
Text(0.5, 1.0, 'Energie en fonction du nombre d itérations')
```



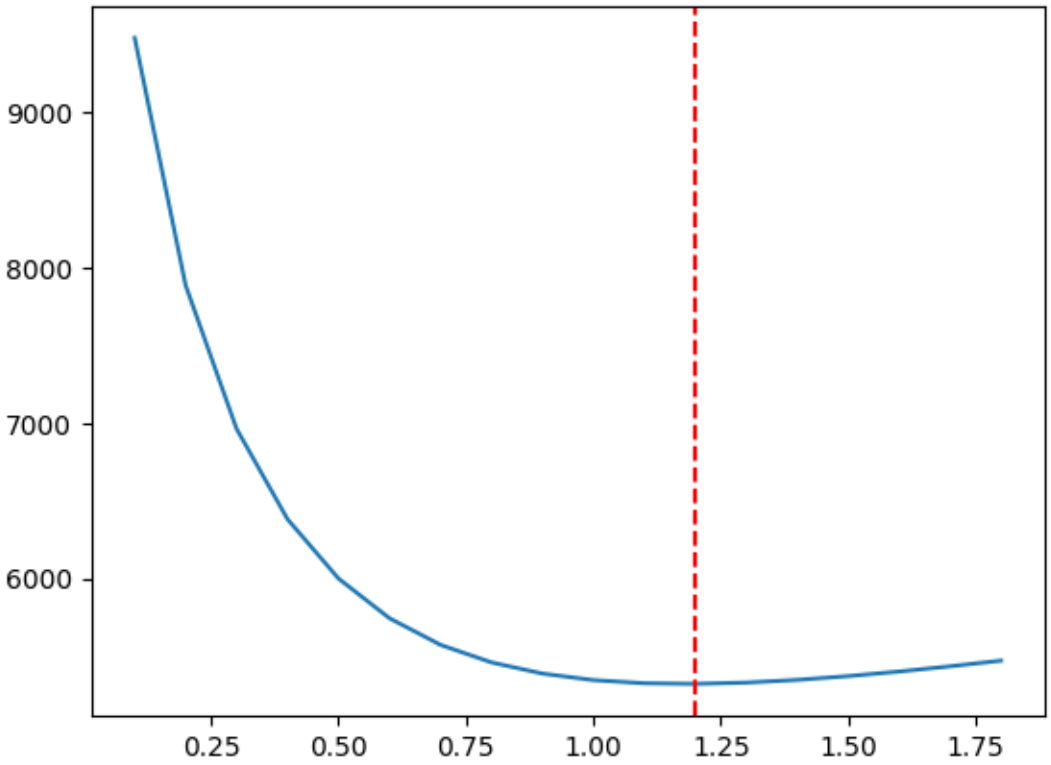
[15]:
```
#%% COMPARAISON des methodes
# vous pouvez vous inspirer de ce qui suit pour trouver les meilleurs
# parametres de regularisation

errvt=[]
vk=np.arange(1,10,0.5)
for k in vk:
    #restq=minimisation_quadratique(imb,10^(k));
    #errq.append[]=norm2(restq-myim);
    restvt=vartotale_Chambolle(imb,10*k)
    #restq=minimisation_quadratique(imb,k)
    #erreur.append(norm2(im-restq))
    errvt.append(norm2(restvt-im))
plt.plot(vk,errvt)
vkminvt = vk[np.argmin(errvt)]
plt.axvline(x=vkminvt, color='r', linestyle='--', label = 'lambda optimal')
```

[15]:
```
<matplotlib.lines.Line2D at 0x2483babadf0>
```



[16]:
```
#%% COMPARAISON des methodes
# vous pouvez vous inspirer de ce qui suit pour trouver les meilleurs
# parametres de regularisation
erreur=[]
vk=np.arange(0.1,1.9,0.1)
for k in vk:
    restq=minimisation_quadratique(imb,k)
    erreur.append(norm2(im-restq))
plt.plot(vk,erreur)
vkminq = vk[np.argmin(erreur)]
plt.axvline(x=vkminq, color='r', linestyle='--', label = 'lambda optimal')
```

[16]:
```
<matplotlib.lines.Line2D at 0x2483ca10400>
```



[17]:
```
viewimage(restq,titre='RESTQUAD_LAMB='+str(vkminq))
viewimage(restvt,titre='RESTTV_LAMB='+str(vkminvt))
```

[ ]: