# BLANCHARD_Theau_TP_IMA203_Methodes_Variationnelles

January 16, 2022

## 1 Import

[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
    import bokeh
    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



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

    """
    print('using the wronf function')
    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 je ne sais comment faire. Si vous savez dites-moi.)
        prephrase='gimp -a '
        endphrase= ' &'
    elif platform.system()=='Windows':
        prephrase='start /B "D:/GIMP/bin/gimp-2.10.exe" -a '#Remplacer D:/...␣
 ↪par le chemin de votre GIMP
        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

    def viewimage(im,normalise=True,MINI=0.0, MAXI=255.0,titre=''):

        img=normalise_image_pour_bokeh(np.flipud(im),normalise,MINI,MAXI)# np.
 ↪flipud(np.fliplr(im)))
```

```
        p = figure(tooltips=[("x", "$x"), ("y", "$y"), ("value",␣
↪"@image")],title=titre)
        p.x_range.range_padding = p.y_range.range_padding = 0

        # must give a vector of images
        #p.image_rgba(image=[img], x=0,y=0, dw=im.shape[1], dh=im.shape[0])

        img_gray = img[:,:,0]
        p.image(image=[img_gray], x=0,y=0, dw=im.shape[1], dh=im.shape[0])
        showbokeh(p)


    def viewimage_plt(im,normalise=True,titre=''):
        if normalise :
            m = np.min(im)
            im - m
            M = np.max(m)
            if M > 0 :
                im = im/M

        plt.figure(figsize=(12,12))
        plt.imshow(im, cmap='gray')
        plt.title(titre)
        plt.show()
```

[4]:
```
## J'ai repris les fonctions du TP sur le graph cut qui elles fonctionnent avec␣
↪Bokeh

def affiche_pour_colab(im,MINI=None, MAXI=None,titre=''): #special colab, don't␣
↪look
  def normalise_image_pour_bokeh(X,MINI,MAXI):
    if MAXI==None:
      MAXI = np.max(X)
    if MINI==None:
      MINI = np.min(X)
    imt=np.copy(X.copy())
    imt=(np.clip(imt,MINI,MAXI)/(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
```

```
  img = im
  img=normalise_image_pour_bokeh(np.flipud(im),MINI,MAXI)
  p = figure(tooltips=[("x", "$x"), ("y", "$y"), ("value", "@image")],␣
↪y_range=[im.shape[0], 0], x_range=[0, im.shape[1]], title=titre)
  # p.x_range.range_padding = p.y_range.range_padding = 0
  # must give a vector of images
  p.image(image=[np.flipud(im)], x=0, y=im.shape[0], dw=im.shape[1], dh=im.
↪shape[0], palette="Greys9", level="image")
  p.xgrid.visible = False
  p.ygrid.visible = False
  showbokeh(p)

def affiche(im,MINI=0.0, MAXI=None,titre='',printname=False):
  affiche_pour_colab(im,MINI=MINI, MAXI=MAXI,titre=titre) # under google colab␣
↪many options disappear
```

## 2 Fonction utiles

[5]:
```
#%% foncttab_lambdautiles 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]) #Les deux premiers termes sont nuls.
        fK=fft2(K[k]) #Les deux premiers termes sont la TF des filtres␣
 ↪gradients fois sqrt(lambda) .Le derniers termes vaut 1 partout.
        #print('type de fV',fV.dtype,' type de fK',fK.dtype)
        numer+=np.conj(fK)*fV #Le dernier terme (le seul non nul) vaut la TF de␣
 ↪l'image bruitée
        denom+=abs(fK)**2 # La formule du poly
    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):
        #print(k)
        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 sur G_a"""
    (m,n)=v.shape
    P=projection(v,lamb/2,itmax)
    return v-P


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

## 3 Lecture et dégradation de l'image

[6]:
```
#%% lire une image

im=imread('lena.tif') #ATTENTION IL FAUT ETRE DANS LE BON REPERTOIRE (utiliser␣
 ↪os.chdir())
```

[7]:
```
#%%
im=imread('lena.tif')
imb=degrade_image(im,25)
```

```
affiche(im,titre="Originale")
affiche(imb,titre="Bruitée")
```

## 4 Débruitage par régularisation quadratique

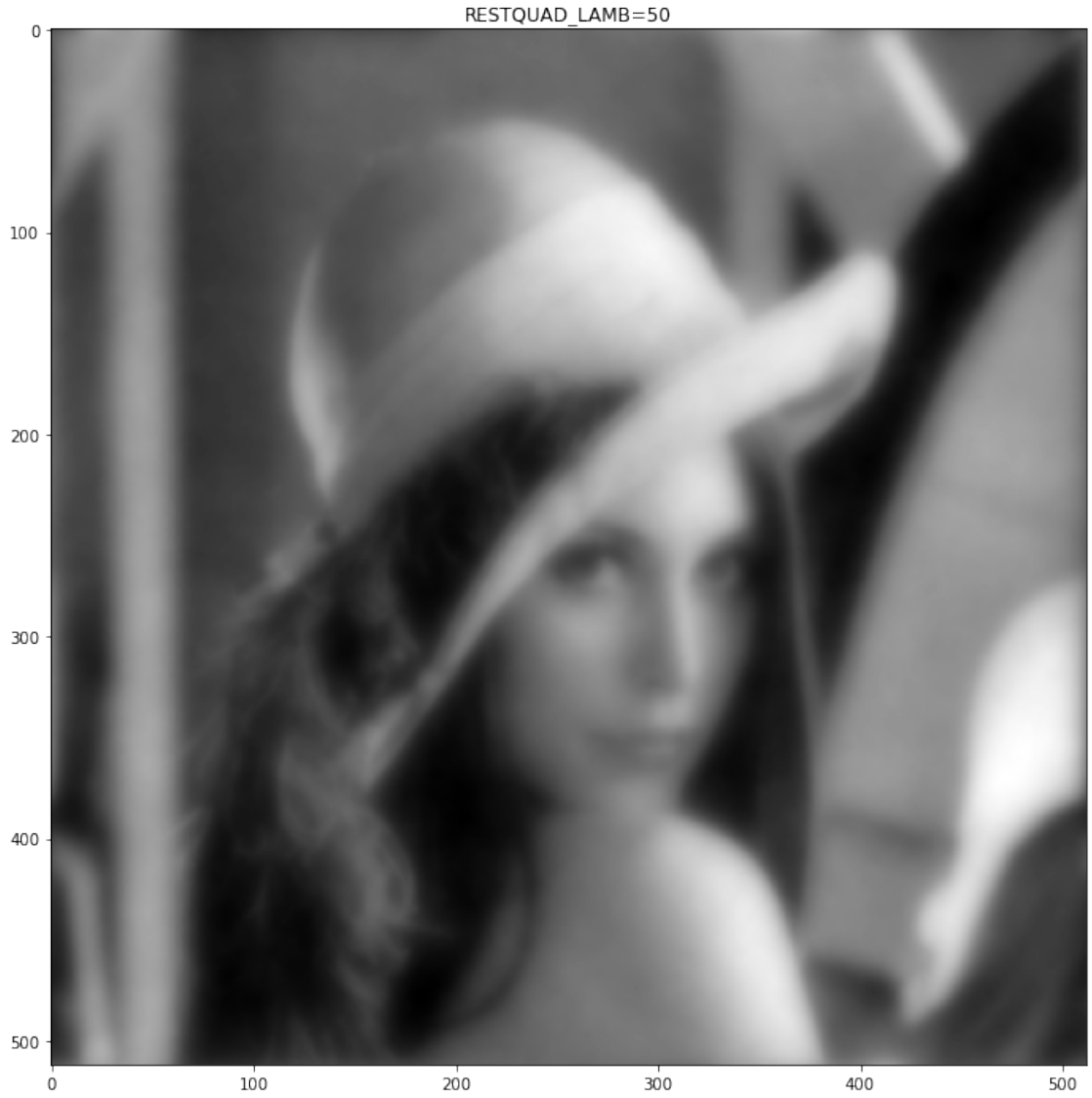
**1/Explication de la méthode** L'algorithme de restauration quadratique implémente le calcul issu du polycopié qui met en relation ce que l'on veut minimiser avec des transformées de Fourier. Plus précisement dans resoud_quad_fourier, on va avoir : - Pour k = 0 et k = 1, fV = 0 donc on aura num = 0 - Pour k = 2, fV est la TF de l'image bruitée et fK vaut 1 partout, donc au final on aura num qui sera la TF de l'image bruitée.

On reconnait dans la définition de Kx et Ky, la définition de la dérivée en x et y dans les coin supérieur gauche. Par linéarité on aura que la TF de s*K_i pour i = x et i = y sera la TF du filtre correspondant multipliée par la racine de lambda. Enfin la définition de delta donne que sa TF vaut un partout.

Finalement on a reconstruit la formule du poly qui minimise cette fonctionnelle pour chaque $\omega$. On applique ensuite la TF inverse pour retrouver l'image reconstruite.

**2/Discussion sur $\lambda$ :** On voit que pour des lambdas faibles (<2 mettons), l'image reconstruite reste très bruitée. C'est normal car l'influence du paramètre de régularisation est négligeable et que donc $E_1(u) \approx ||u - v||^2$ qui est minimale pour $ u = v $. Plus on augmente lambda plus l'image est débruitée mais plus elle est aussi floue. Pour des lambdas très grands, le terme d'attache aux données devient très faible devant le terme de régularisation. Or l'intégrale du gradient est minimisée lorsque celui-ci est nulle, c'est-à-dire lorsque l'image est uniforme.

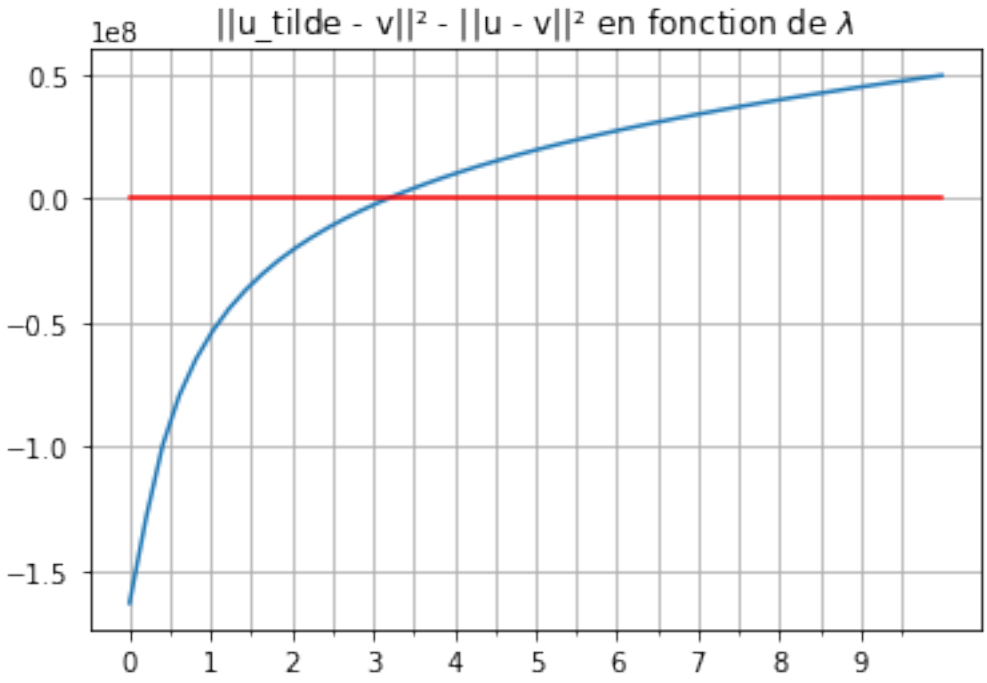
[8]:
```
#%% restauration quadratique : exemple
lamb=50
restau=minimisation_quadratique(imb,lamb)
viewimage_plt(restau,titre='RESTQUAD_LAMB='+str(lamb))
```




[9]:
```
# Graphe de l'erreur entre l'image restaurée et le bruit moins l'erreur entre␣
 ↪l'image parfaite et l'image bruitée
tab_lambda = np.linspace(0,10,50)
tab_erreur = np.array([norm2(minimisation_quadratique(imb,k) - imb)**2 for k in␣
 ↪tab_lambda])
ref_value = norm2(im-imb)**2
tab_erreur = tab_erreur - ref_value
ligne_zero = np.array([0 for k in range(len(tab_lambda))])
```

[10]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
major_ticks = np.arange(0, 10, 1)
```

```
minor_ticks = np.arange(0, 10, 0.5)
ax.set_xticks(major_ticks)
ax.set_xticks(minor_ticks, minor=True)
ax.grid(which='both')
#ax.scatter(lambda_optim1,error_optim1-ref_value)
ax.plot(tab_lambda,tab_erreur)
ax.plot(tab_lambda,ligne_zero, c='r')
ax.set_title(r'||u_tilde - v||² - ||u - v||² en fonction de $\lambda$')
plt.show()
```




[11]:
```
def dichotomie(ref_value,v, lambda_min = 0 ,lambda_max = 1000, eps = 0.1):
    a, b = lambda_min, lambda_max
    # La fonction à annuler est ||u_tilde-v||² - ||u - v ||²
    def f(lamb) :
        u_tilde = minimisation_quadratique(v,lamb)
        found_value = norm2(u_tilde - v)**2
        return found_value - ref_value

    while b - a > eps :
        m = (a+b)/2
        if f(a)*f(m) <= 0 :
            b = m
        else :
            a = m
```

```
    return m
```

[12]:
```
ref_value = norm2(im-imb)**2
lambda_optim1 = dichotomie(ref_value, imb)
u_tilde1 = minimisation_quadratique(imb,lambda_optim1)
error_optim1 = norm2(u_tilde1-imb)**2
print("Valeur de l'erreur de ||u - v||²", ref_value)
print("Valeur de l'erreur ||u_tilde - imb ||² dans ce cas", error_optim1)
print("Erreur relative des deux erreurs ", np.abs(error_optim1-ref_value)/
 ↪ref_value *100)
print("Valeur de lambda qui minimise ||u_tilde - imb ||² - ||u -␣
 ↪v||²",lambda_optim1)
```

```
Valeur de l'erreur de ||u - v||² 162811068.87018296
Valeur de l'erreur ||u_tilde - imb ||² dans ce cas 163549985.33963355
Erreur relative des deux erreurs  0.4538490377701324
Valeur de lambda qui minimise ||u_tilde - imb ||² - ||u - v||² 3.23486328125
```

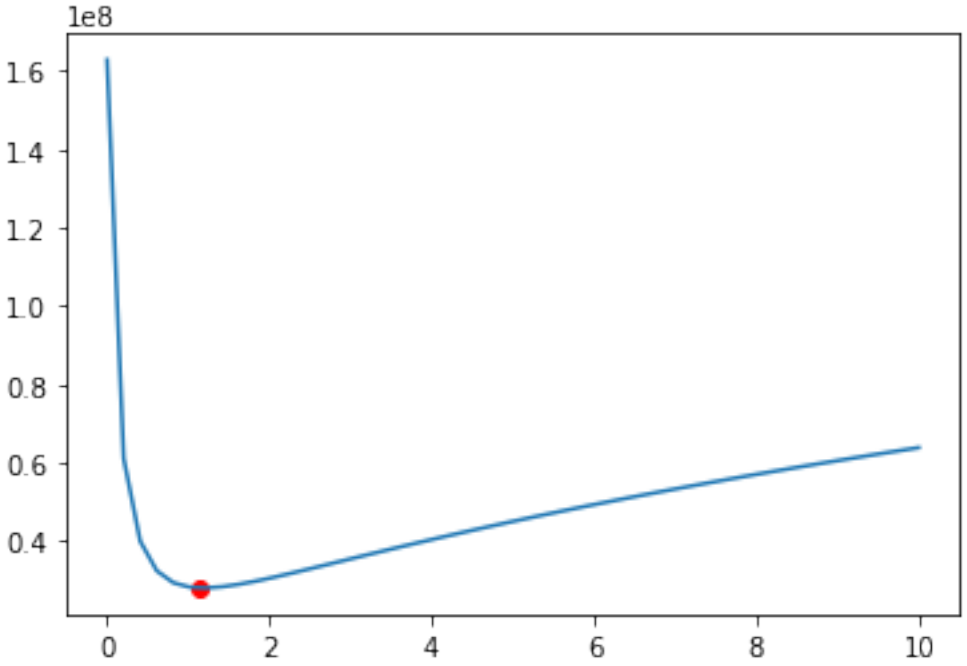
[13]:
```
affiche(im,titre="Originale")
affiche(u_tilde1,titre="Restauration par minimisation quadratique")
```

[14]:
```
def minisation_erreur(imb,u, lambda_min = 0, lambda_max = 5, eps=0.1):

    def f(m,u) :
        u_tilde = minimisation_quadratique(imb,m)
        found_error = norm2(u_tilde - u)**2
        return found_error


    a, b = lambda_min, lambda_max
    while b-a > eps :
        m = (b+a)/2
        if f(m,u) <= f(b,u):
            b = m
        else :
            a = m

    return (a+b)/2
```

[15]:
```
lambda_optim2 = minisation_erreur(imb,im)
u_tilde2 = minimisation_quadratique(imb,lambda_optim2)
error_optim2 = norm2(u_tilde2 - im)**2
tab_lambda = np.linspace(0,10,50)
tab_erreur = np.array([norm2(minimisation_quadratique(imb,k) - im)**2 for k in␣
 ↪tab_lambda])
```

[16]:
```
print("Valeur de l'erreur ||u_tilde - u ||² dans ce cas", error_optim2)
print(r"Valeur de lambda qui minimise ||u - v||²",lambda_optim2)
plt.plot(tab_lambda,tab_erreur)
plt.scatter(lambda_optim2, error_optim2, c='r')
plt.show()
affiche(u_tilde2,titre='Minimisation quadratique')
```

```
Valeur de l'erreur ||u_tilde - u ||² dans ce cas 28055252.94465143
Valeur de lambda qui minimise ||u - v||² 1.1328125
```

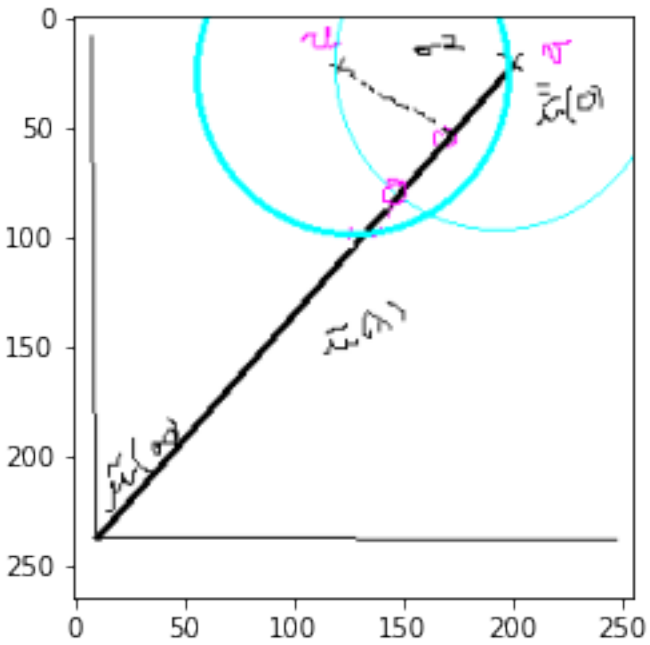



[17]:
```
print("Finalement avec la première méthode on obtient une erreur de␣
 ↪",error_optim1, "contre ",error_optim2," avec la 2ème méthode. Soit un␣
 ↪rapport entre les deux de", error_optim1/error_optim2)
print("La 2ème méthode est donc plus efficace. On le comprend bien avec un␣
 ↪dessin car la 2ème réalise le projeté orthogonal de u sur les␣
 ↪u_tilde(lambda)")
dessin = imread("explication.png")
plt.imshow(dessin)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
Finalement avec la première méthode on obtient une erreur de  163549985.33963355
contre  28055252.94465143  avec la 2ème méthode. Soit un rapport entre les deux
de 5.829567306425352
```

La 2ème méthode est donc plus efficace. On le comprend bien avec un dessin car la 2ème réalise le projeté orthogonal de u sur les u_tilde(lambda)



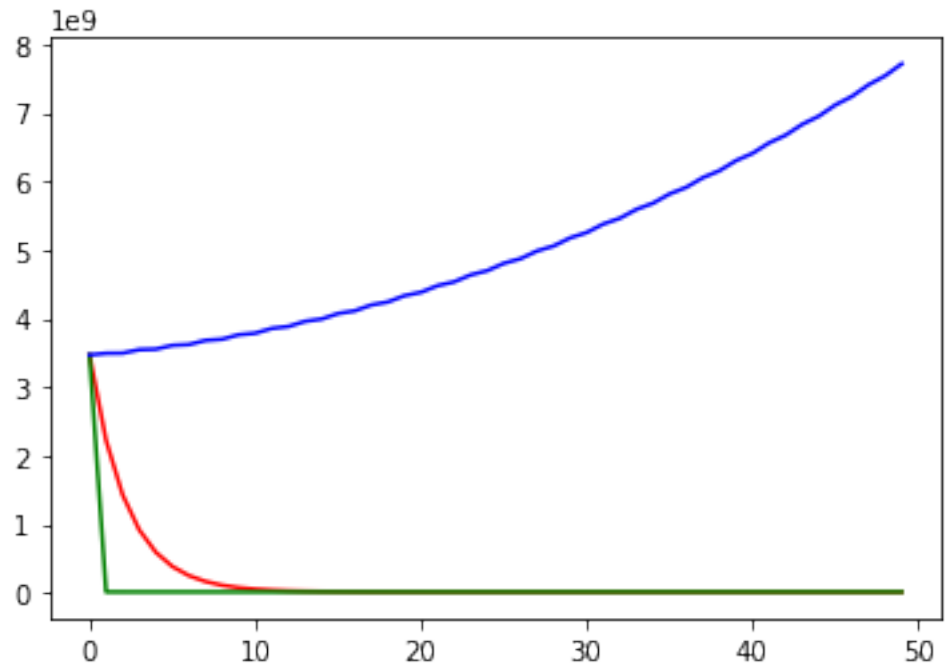

# 5 Débruitage par variation totale

## 5.1 Descente de gradient

On constate tout d'abord que cette méthode peut être assez longue si le nombre d'itérations devient trop important. De plus on n'a pas de convergence assuré si le pas n'est pas choisi correctement.

[18]:
```
#%%
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)
```

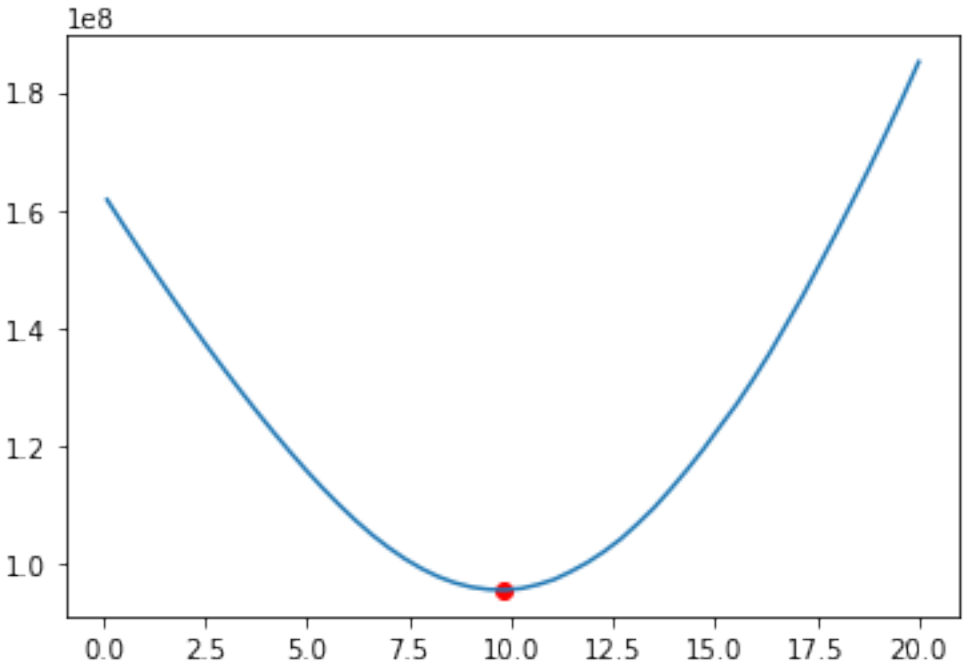
[19]:
```
#%%
#myim=imread('lena.tif')
#imb=degrade_image(myim,25)
(u,energ)=minimise_TV_gradient(imb,1,0.1,50)    # pas = 0.1
(u,energ1)=minimise_TV_gradient(imb,1,0.5,50)       # pas = 0.5
(u,energ2)=minimise_TV_gradient(imb,1,1,50)       # pas = 1
plt.plot(energ,c='r')
plt.plot(energ1,c='g')
plt.plot(energ2,c='b')
plt.show()
```




[20]:
```
# Graphe de l'erreur entre l'image restaurée et le l'image parfaite
tab_lambda = np.linspace(0.1,20,50)
tab_erreur = np.array([norm2(minimise_TV_gradient(imb,k,0.5,100)[0] - im)**2␣
 ↪for k in tab_lambda])
```

[21]:
```
lambda_gradient = tab_lambda[np.argmin(tab_erreur)]
erreur_gradient = np.min(tab_erreur)
u_gradient = minimise_TV_gradient(imb,lambda_gradient,0.5,100)[0]

print("Meilleur lambda ",lambda_gradient)
print("Erreur initiale ", norm2(imb-im)**2)
print("Erreur finale   ", norm2(u_gradient-im)**2)
print("Rapport des erreurs",norm2(imb-im)**2/norm2(u_gradient-im)**2)
plt.plot(tab_lambda, tab_erreur)
plt.scatter(lambda_gradient,erreur_gradient,c='r')
plt.show()
affiche(u_gradient,titre="Restauration par variation totale, descente de␣
 ↪gradient")
```

```
Meilleur lambda  9.846938775510203
Erreur initiale  162811068.87018296
Erreur finale    95394745.74919014
Rapport des erreurs 1.7067089763857897
```

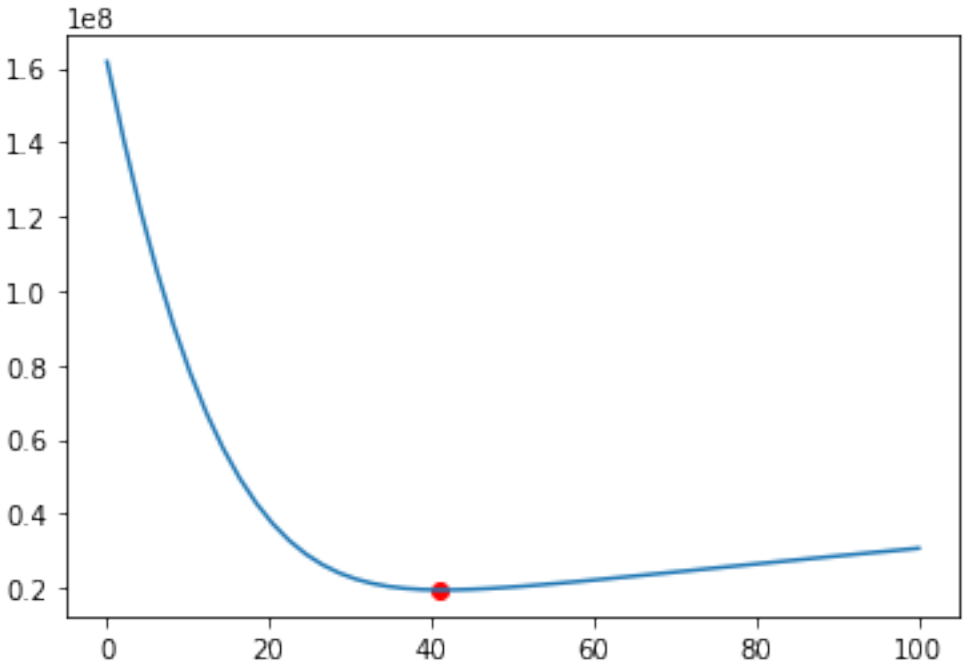



### 5.2 Projection de Chambolle

[22]:
```
# Graphe de l'erreur entre l'image restaurée et le l'image parfaite
tab_lambda2 = np.linspace(0.1,100,50)
tab_erreur2 = np.array([norm2(vartotale_Chambolle(imb,k) - im)**2 for k in␣
↪tab_lambda2])
```

[23]:
```
lambda_chambolle = tab_lambda2[np.argmin(tab_erreur2)]
erreur_chambolle = np.min(tab_erreur2)
u_chambolle = vartotale_Chambolle(imb,lambda_chambolle)
print("Meilleur lambda ",lambda_chambolle)
print("Erreur initiale ", norm2(imb-im)**2)
print("Erreur finale   ", norm2(u_chambolle-im)**2)
print("Rapport des erreurs",norm2(imb-im)**2/norm2(u_chambolle-im)**2)

plt.scatter(lambda_chambolle,erreur_chambolle,c='r')
plt.plot(tab_lambda2, tab_erreur2)
plt.show()
affiche(u_chambolle,titre="Restauration par variation totale, projection de␣
↪Chambolle")
```

```
Meilleur lambda  40.87551020408164
Erreur initiale  162811068.87018296
Erreur finale    19503149.686981417
Rapport des erreurs 8.347937204156377
```



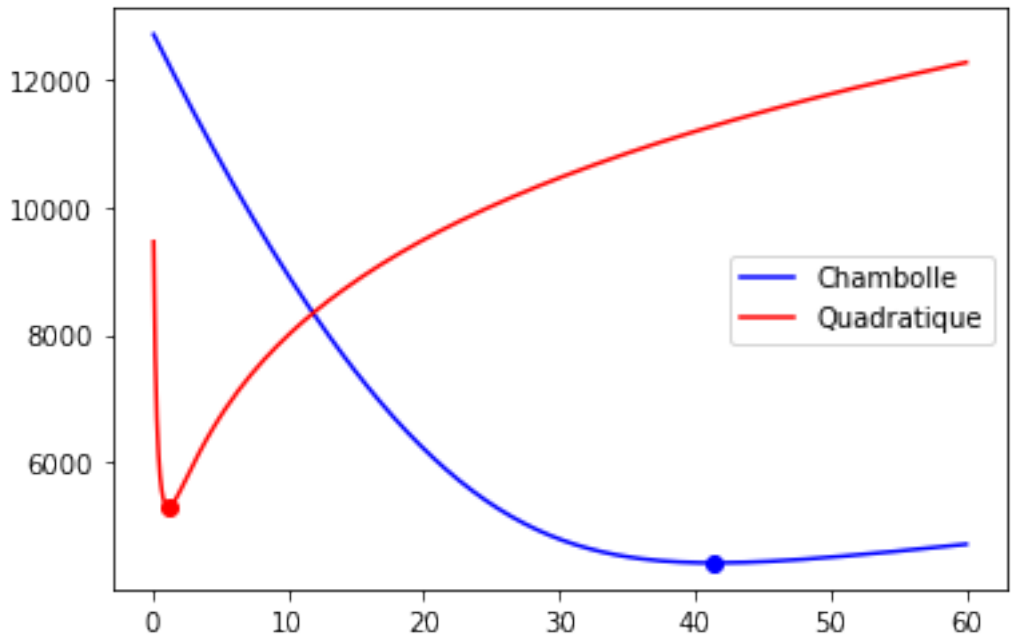

## 6 Comparaison des méthodes

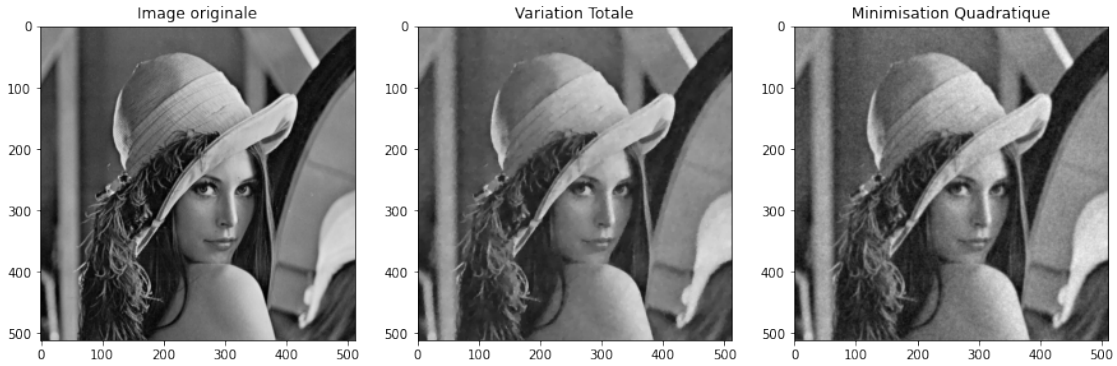
[24]:
```
errvt=[]
erreur=[]

vk=np.concatenate((np.linspace(0.1,3,50),np.linspace(3,60,120)))
loading_bar = "."*(50+120)
print('Loading bar')
print(loading_bar)
for k in vk:
    print ('.',end='' )
    restva=vartotale_Chambolle(imb,k)
    restq=minimisation_quadratique(imb,k)
    erreur.append(norm2(im-restq))
    errvt.append(norm2(im-restva))
```

```
Loading bar
…
…
…
…
…
…
```

[25]:
```
best_quadratique = vk[np.argmin(erreur)]
best_chambolle = vk[np.argmin(errvt)]
err_init = norm2(imb-im)**2
err_qt = np.min(erreur)
err_cham = np.min(errvt)
u_qt = minimisation_quadratique(imb,best_quadratique)
u_cham = vartotale_Chambolle(imb,best_chambolle)


print("Meilleur lambda pour la variation totale",best_chambolle)
print("Meilleur lambda pour la minimisation quadratique",best_quadratique)

print("Erreur initiale ", err_init)
print("Erreur initiale / minimisation quadratique ", err_init/err_qt)
print("Erreur initiale / variation totale ", err_init/err_cham)
print("Erreur minimisation quadratique / variation totale ",err_qt/err_cham)
#print("Rapport des erreurs",norm2(imb-im)**2/norm2(u_chambolle-im)**2)


plt.scatter(best_quadratique,err_qt,c='r')
plt.scatter(best_chambolle,err_cham,c='b')
plt.plot(vk,errvt, label = 'Chambolle',c='b')
plt.plot(vk,erreur, label = 'Quadratique',c='r')
plt.legend()
plt.show()

fig, (ax0,ax1, ax2) = plt.subplots(1, 3, figsize=(15,15))
ax0.imshow(im,cmap='gray')
ax0.set_title("Image originale")
ax1.imshow(u_chambolle,cmap='gray')
ax1.set_title("Variation Totale")
ax2.imshow(u_qt,cmap='gray')
ax2.set_title("Minimisation Quadratique")
plt.show()
```

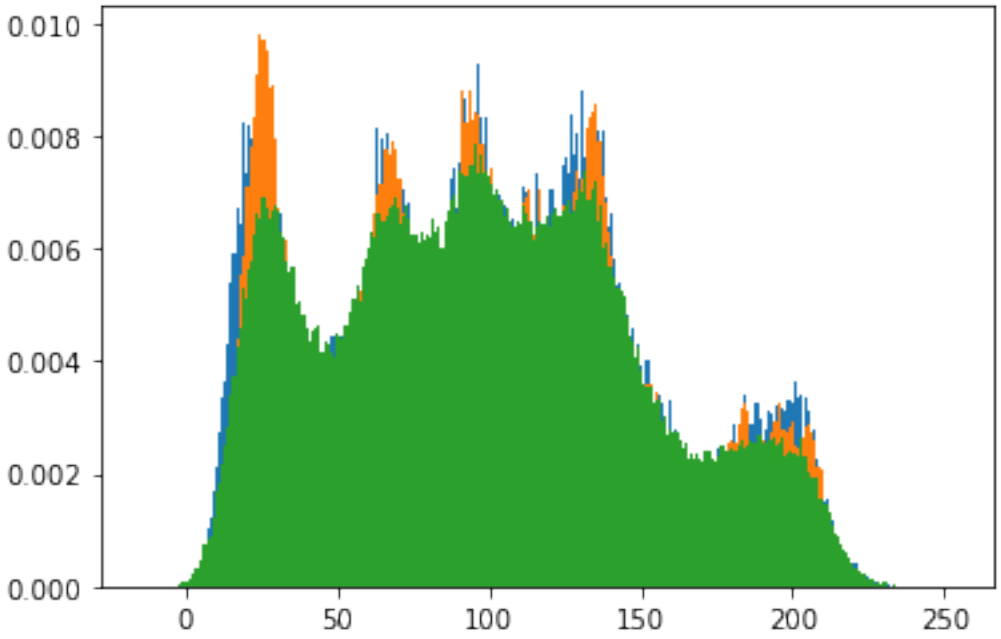
```
Meilleur lambda pour la variation totale 41.319327731092436
Meilleur lambda pour la minimisation quadratique 1.1653061224489796
Erreur initiale  162811068.87018296
Erreur initiale / minimisation quadratique  30745.59276178834
Erreur initiale / variation totale  36870.96111614313
Erreur minimisation quadratique / variation totale  1.1992275251224822
```

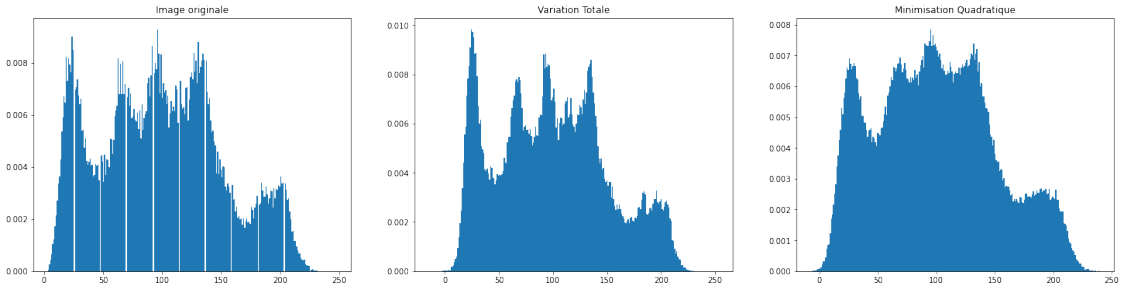



[26]:
```
plt.hist(im.flatten(),bins=256,density=True)
plt.hist(u_cham.flatten(),bins=256,density=True)
plt.hist(u_qt.flatten(),bins=256,density=True)
plt.show()

fig, (ax0,ax1, ax2) = plt.subplots(1, 3, figsize=(25,6))
ax0.hist(im.flatten(),bins=256,density=True)
ax0.set_title("Image originale")
ax1.hist(u_cham.flatten(),bins=256,density=True)
ax1.set_title("Variation Totale")
ax2.hist(u_qt.flatten(),bins=256,density=True)
ax2.set_title("Minimisation Quadratique")
```

```
plt.show()
```




Finalement on trouve que quantitativement la méthode de variation totale est plus efficace. Tout d'abord, elle est au moins aussi rapide à calculer par la méthode de Chambolle que la méthode de minimisation quadratique car on s'évite le calcul des TF. Ensuite l'erreur obtenue est 1.2 fois plus petite que l'erreur par l'autre méthode.

Qualitativement le résultat est plus subtil. S'il apparait bien que par la variation totale on a moins de bruit sur l'image, il semble que l'on ait perdu en contraste (même si cela ne se voit pas sur les histogrammes).