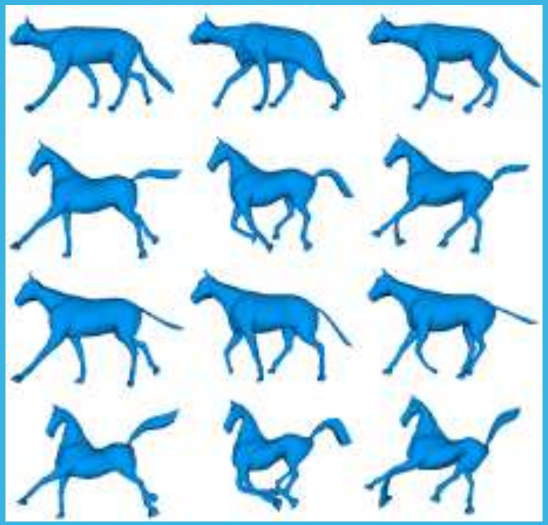
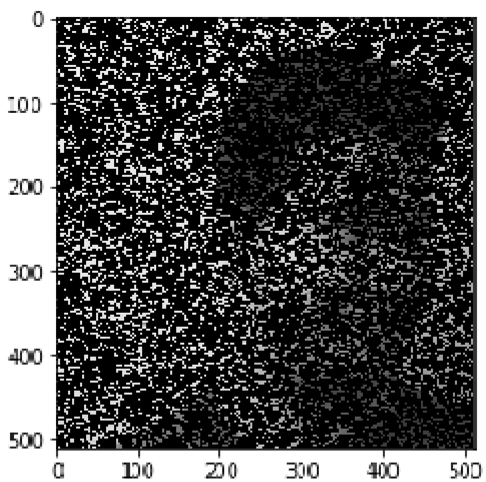
Thème: Transport
Année universitaire: 2018/2019
Numéro d'inscription: 14663

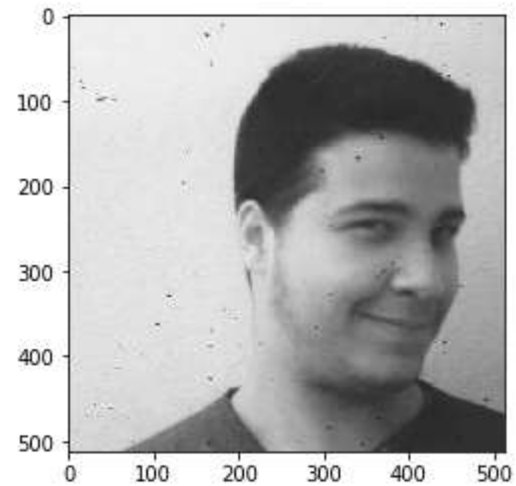
# TRANSPORT OPTIMAL ET INTERPOLATION D'IMAGES



# PROBLÉMATIQUE:

- I- Comment peut-on traiter les images en vertu de la méthode du gradient conjugué ?
- II- Comment peut-on modéliser le transport optimal d'images ?
- III- A quoi ça sert d'utiliser l'Inpainting ?



# CONTRIBUTIONS:

- Rencontre du mathématicien français Claude Deschamps (novembre 2018)
- Rencontre du physicien français Edouard Tantart (décembre 2018, février 2019)
- Implémentation d'un code python modélisant le transport optimal des pixels dans une image et mettant en valeur l'interpolation des images.

# PLAN DE TRAVAIL:

I- Introduction

II- Les dérivées d’une image

III-Projection sur la contrainte

IV- Résolution par méthode de descente de gradient à pas optimal

V- Minimisation différente pour l’inpainting

VI- Conclusion

# I- INTRODUCTION:

- Le transport optimal est une discipline qui s'interesse  à l'étude de transport optimal de matière en allouant des ressources minimales
- Plusieurs formulations: Monge et Kantorovitch
- Applications dans plusieurs domaines ( militaires, recherche opérat- ionnelles  et traitement d'images)

- Problème de transport optimal pour construire une interpolation entre deux images: une image détériorée et une autre corrigée


Mémoire sur la théorie des déblais et des remblais (1781)

MÉMOIRE
SUR LA
THÉORIE DES DÉBLAIS
ET DES REMBLAIS.
Par M. MONGE.

LORSQU'ON doit tranſporter des terres d'un lieu dans un autre, on a coutume de donner le nom de *Déblai* au volume des terres que l'on doit tranſporter, & le nom de *Remblai* à l'eſpace qu'elles doivent occuper après le tranſport.

Le prix du tranſport d'une molécule étant, toutes choſes d'ailleurs égales, proportionnel à ſon poids & à l'eſpace qu'on lui fait parcourir, & par conſéquent le prix du tranſport total


# POSITION DU PROBLÈME D'INPAINTING:

- On modélise l'image par une matrice H=($h_{i,j}$) ∈ $M_{n_{1},n_{2}}(\mathbb{R})$
- $h_{i,j}$ ∈[0,1] : la valeur de la luminosité
- M=($m_{i,j}$) la matrice du masque tel que
- $m_{i,j} = \begin{cases} 1, si\ le\ pixel\ est\ transmis \\ 0, si\ le\ pixel\ n'est\ pas\ transmis \end{cases}$

- On cherche une matrice U=($u_{i,j}$) tel que $u_{i,j}$=$h_{i,j}$ si le pixel est transmis afin de trouver une image complète qui corresponde le maximum l'image parfaite d'origine.

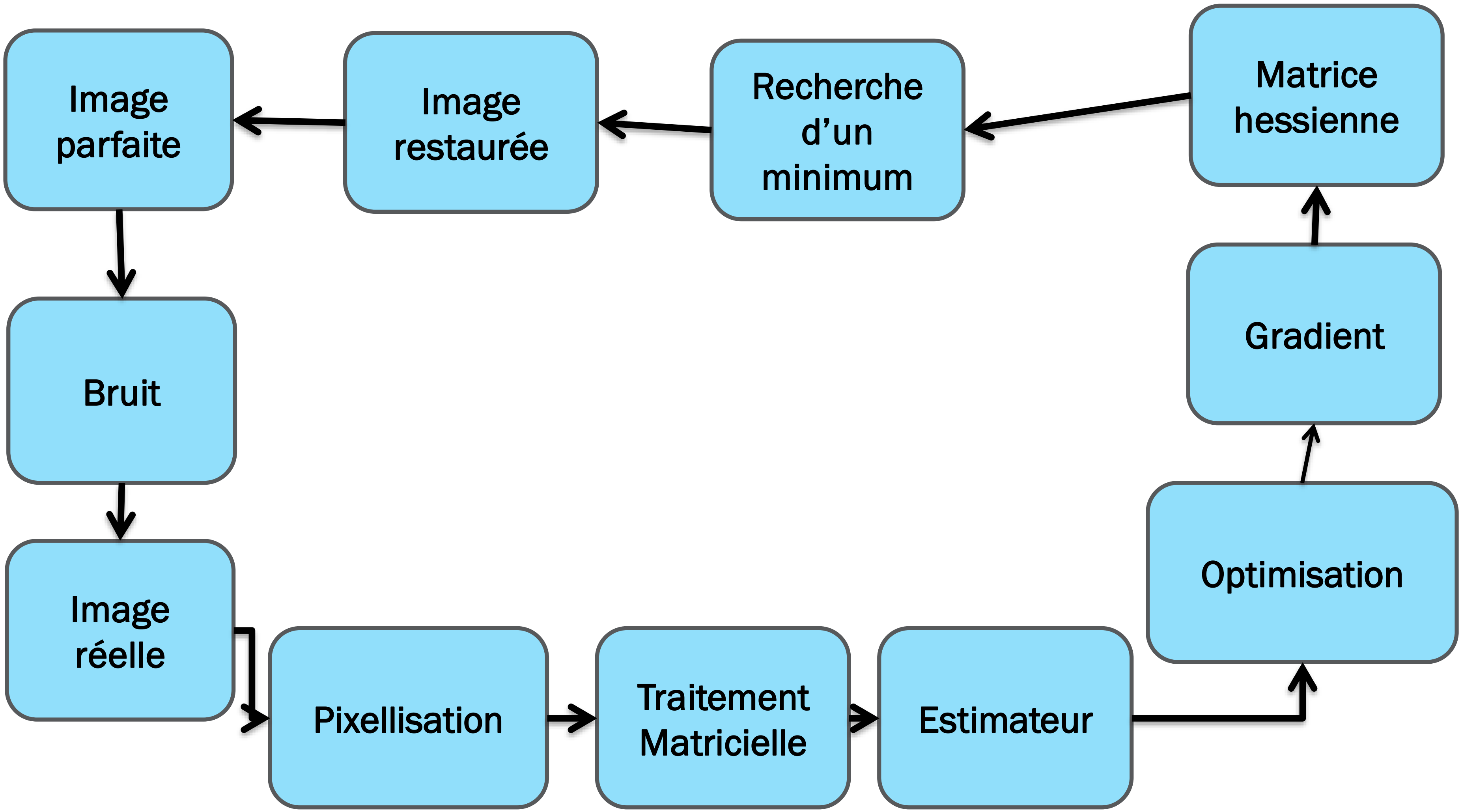
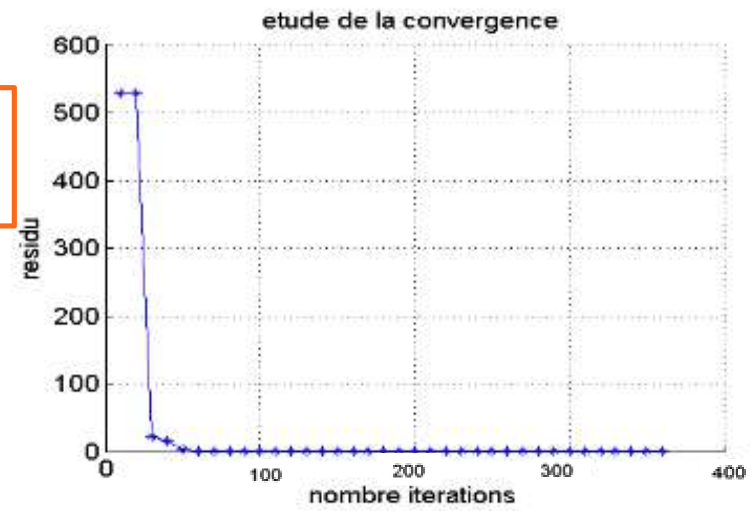


Image parfaite
Image restaurée
Recherche d'un minimum
Matrice hessienne
Gradient
Optimisation
Bruit
Image réelle
Pixellisation
Traitement Matricielle
Estimateur

# RÉSOLUTION PAR LA MÉTHODE DU GRADIENT CONJUGUÉ:

Chaque ingénieur en Machine Learning cherche toujours à améliorer les performances de son modèle. C'est là que l'optimisation, l'un des domaines les plus importants dans le monde moderne, qui entre en jeu. Il nous permet de sélectionner les meilleurs paramètres, associés à la méthode du gradient conjugué que nous utilisons, pour notre problématique.
Il s'agit d'un algorithme d'optimisation pour résoudre un système d'équations linéaire itérative qui converge vers un nombre fini d'itérations.

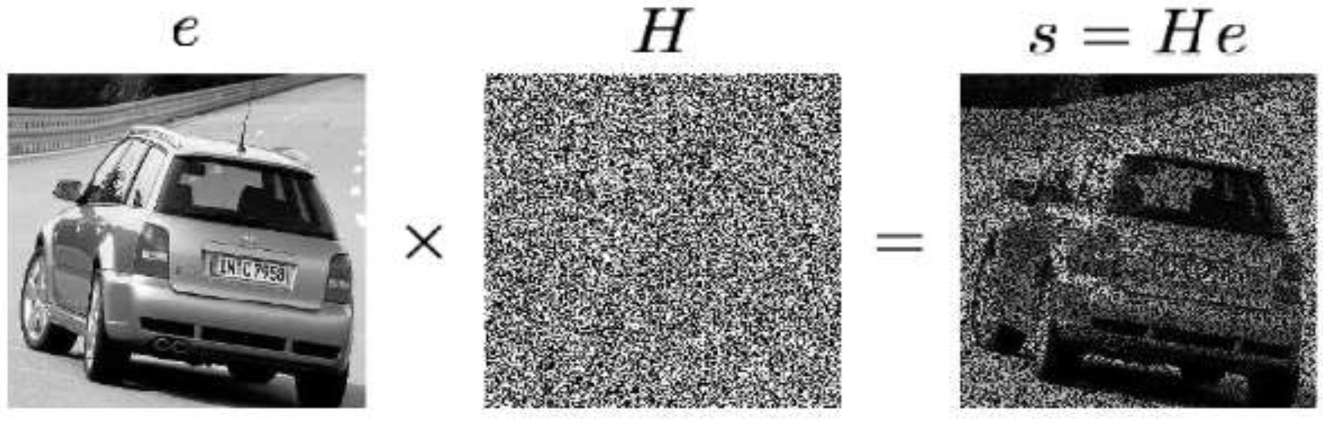
Soit $r_k$ le résidu de la $k^{éme}$ itération tel que
$r_k$ = s-H$e_k$




Ça converge après 350 itérations

## Algorithme itératif en pseudo-code :

L'algorithme ci-dessous résout **H***e* = *s*, où **H** est une matrice réelle, symétrique, et définie positive. Le vecteur d'entrée $e_0$ peut être une approximation de la solution initiale ou 0.



$$r_0 := s - He_0$$

$$p_0 := r_0$$

$$répeter$$

$$\alpha_k := \frac{r_k^T r_k}{p_k^T H_{p_k}}$$

$$e_{k+1} := e_k + \alpha_k p_k$$

Résidu

$$r_{k+1} := r_k - \alpha_k H_{p_k}$$

$si$ $\mathrm{r}_{k+1}$ est suffisamment petit, alors on sort de la boucle

$$\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

Contrainte de conjugaison: c'est une contrainte d'orthonormalité par le procédé de Gram-Schmidt

$$p_{k+1} := r_{k+1} + \beta_k p_k$$

$$k := k + 1$$

fin de répéter

le résultat est $\mathrm{e}_{k+1}$

Comme la matrice H est symétrique définie positive, on peut définir le produit scalaire suivant sur $\mathbb{R}^n$ : $< u, v > = u^T$ H $v$

Ainsi, la méthode du gradient conjugué consiste à construire une suite $(p_k)_{k \in \mathbb{N}^*}$

De n vecteurs conjugués . Ainsi, la suite $p_1$ ..................$p_n$ forme une base de $\mathbb{R}^n$

Remarque: on dit que deux éléments conjugués si pour tout u,v réels on a

<u,v> = 0

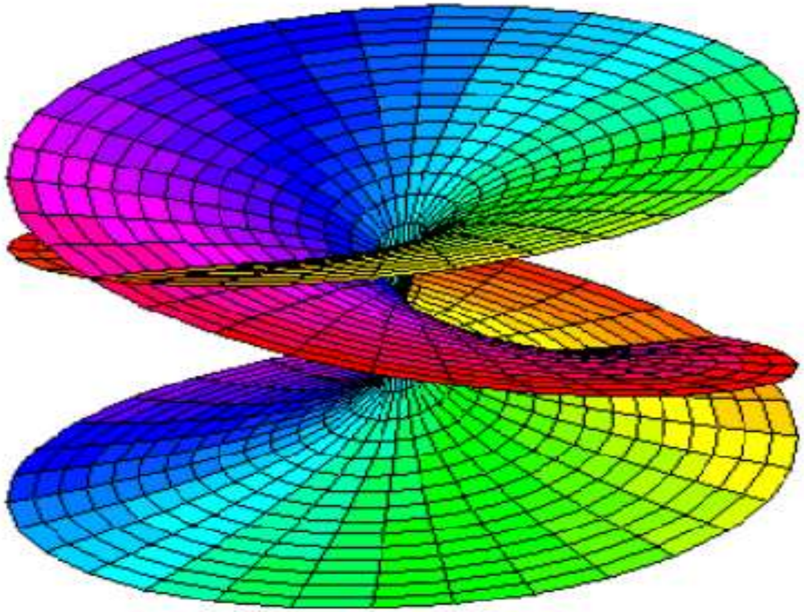
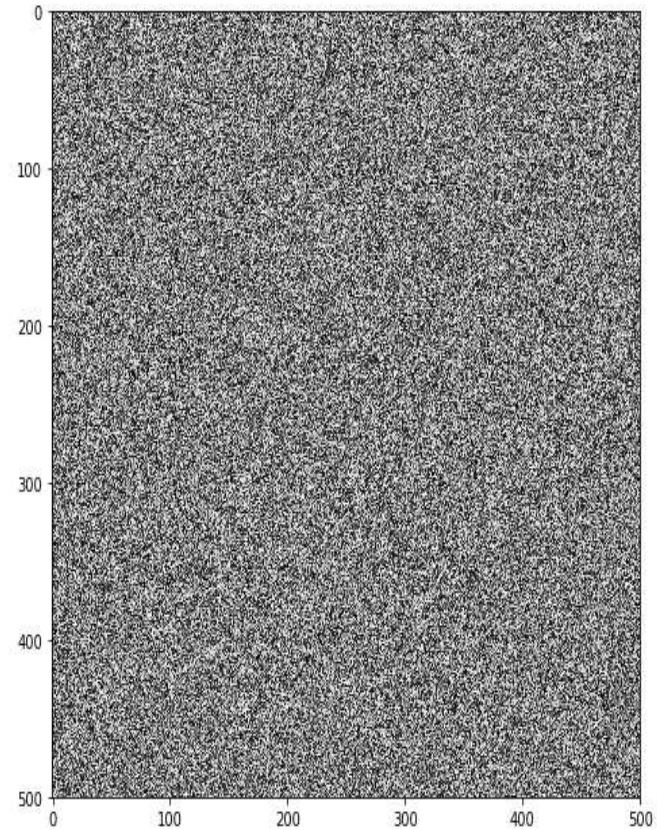



Image réelle




Bruit

# II- LES DÉRIVÉES D'UNE IMAGE:

- Soit I de taille h$\times$ $\boldsymbol{w}$

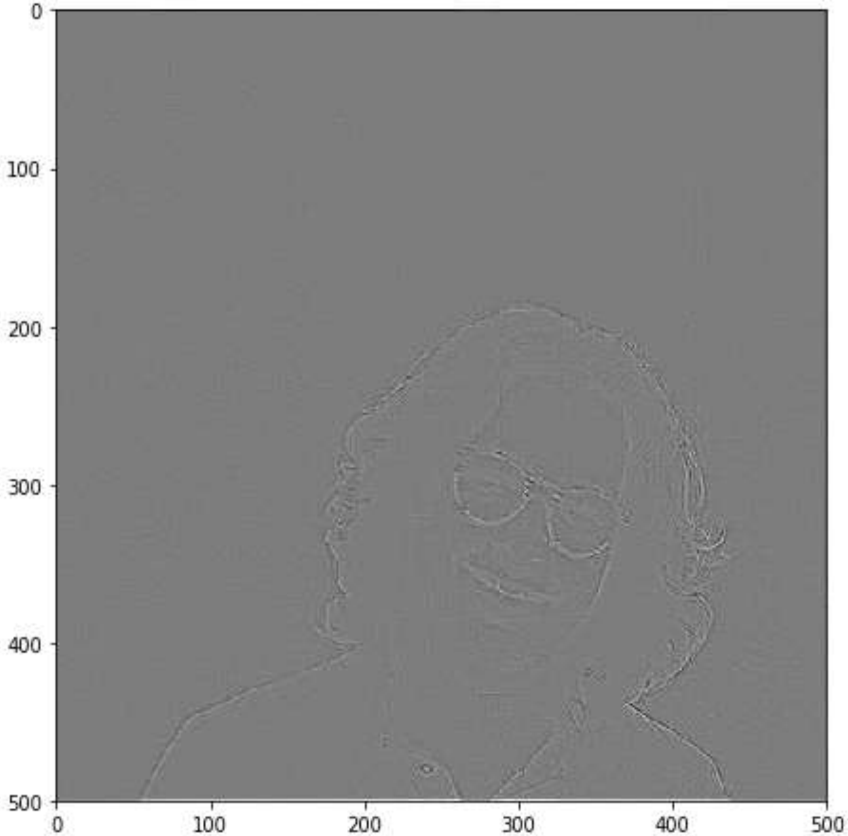
- On appelle le laplacien de I l'image $\Delta_{\boldsymbol{I}}$ de taille h$\times$ $\boldsymbol{w}$

$$(\Delta_{\boldsymbol{I}})[i,j] := I[i+1,j] + I[i-1,j] + I[i,j+1] + I[i,j-1] - 4I[i,j]$$

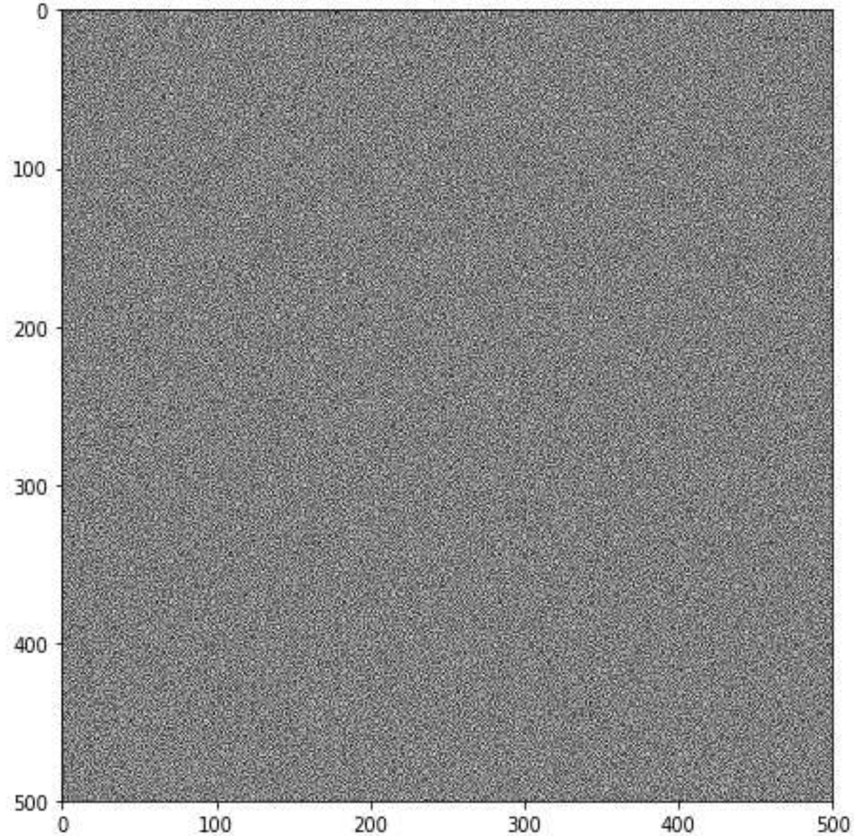
- Le laplacien d'une image initiale est presque uniforme

On applique le gradient sur l'image

Laplacien de l’image réelle




Laplacien de l’image aléatoire



On appelle énergie $H_1$ de l'image $I$ le nombre réel

$$H_1(I) := \left[\sum_{i,j}\left(\left|I[i+1,j]-I[i,j]\right|^2+\left|I[i,j+1]-I[i,j]\right|^2\right)\right]^{1/2}.$$

$$\left[H_1(I)\right]^2 = -\left\langle I, \Delta_I\right\rangle_{\mathbb{R}^{h\times w}} = -\sum_{i,j} I[i,j]\times\Delta_I[i,j]$$

# III- PROJECTION SUR LA CONTRAINTE:

- Dans la méthode inpainting, on sait exactement quelle zone on va restaurer

- On sait quels pixels doivent être modifiés et quels pixels doivent rester constant

**On a accès à un masque M**


0
100
200
300
400
500
0
100
200
300
400
500
Ceci est un texte
qu'on aimerait bien
retirer, non seulement
parce qu'il cache
l'image, mais aussi
parce qu'il est
complètement illisible !
Comment faire ??

# IV-RÉSOLUTION PAR MÉTHODE DE DESCENTE À PAS OPTIMAL:

- On veut minimiser $H_1$(I) sous la contrainte I = P(I)

$$\Updownarrow$$

- I est dans l'image de P ou encore les bons pixels de $I_m$
- L'idée est de faire une descente de gradient  à pas constant  $\tau$ et de projeter les itérations  avec P

$$\boxed{I_{n+1} = P\left(I_n - \tau \nabla H_1\left(I_n\right)\right).}$$

*) Pour faire cela on commence par:

- On commence par calculer le gradient $\boldsymbol{H_1}$

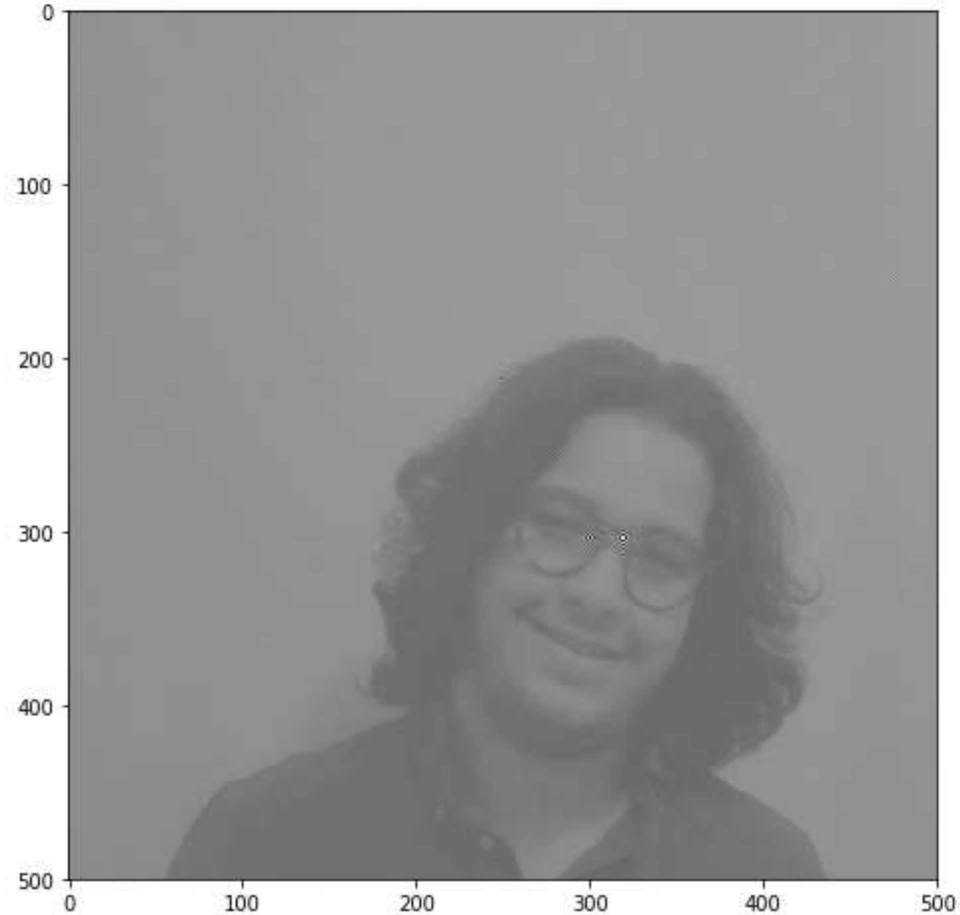
- On fera attention qu'en parlant de gradient

- On voit ici les images comme des vecteurs $\mathbb{R}^{\boldsymbol{h \times w}}$ où on a juste trié les

éléments sous forme d'une image.

- On obtiendra

$$\boxed{\nabla H_1(I) = \frac{-\Delta_I}{\sqrt{-\langle I, \Delta_I \rangle}}.}$$


0
100
200
300
400
500
0
100
200
300
400
500

# V- MINIMISATION DIFFÉRENTE POUR L'INPAINTING :

- La minimisation de $H_1$ donne déjà des résultats relativement convaincants.
- Cependant on arrive encore à lire le texte
- On cherche une nouvelle fonction à minimiser
- On peut regarder par exemple la fonctionnelle suivante appelée

Variation totale

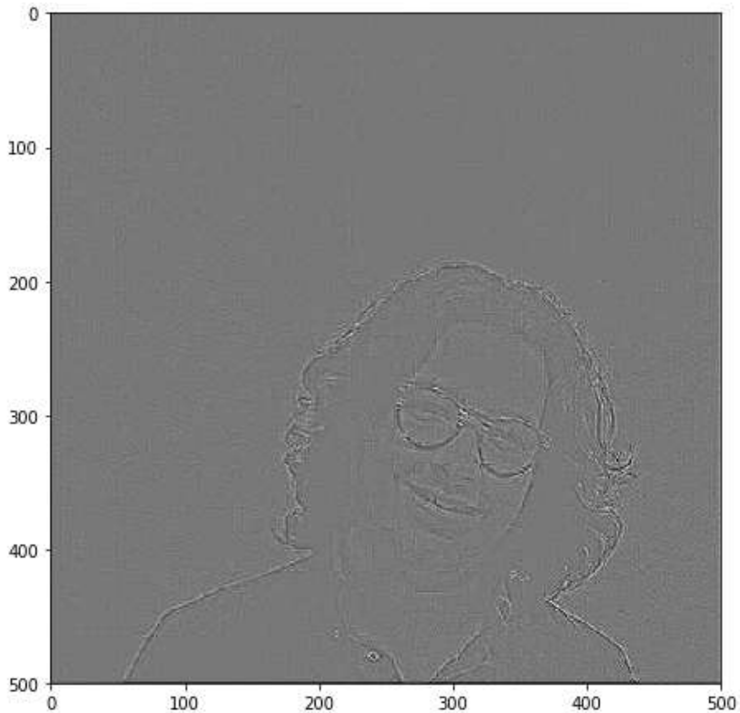
$$TV(I) := \sum_{i,j} \sqrt{\delta + \left|I[i+1,j] - I[i,j]\right|^2 + \left|I[i,j+1] - I[i,j]\right|^2}.$$

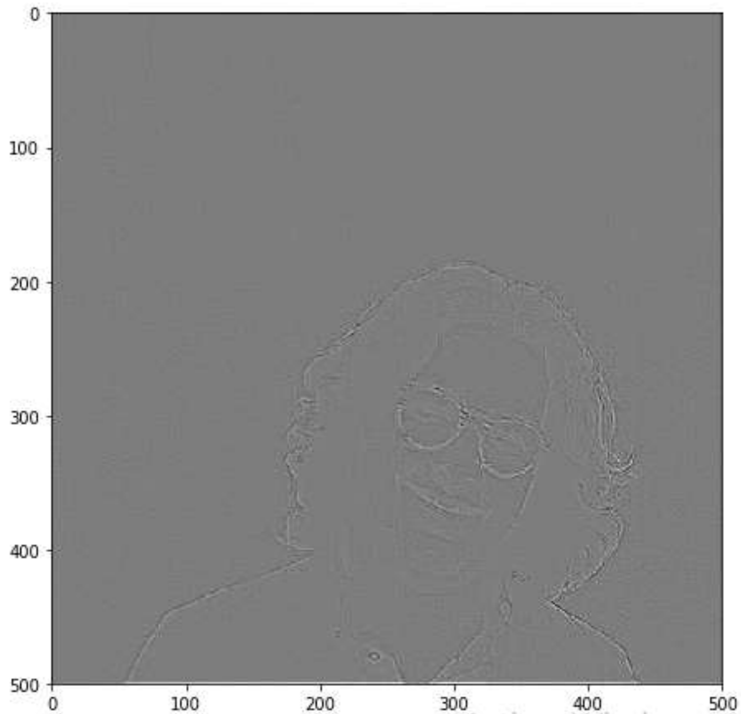
# LE GRADIENT DE TV EN I:

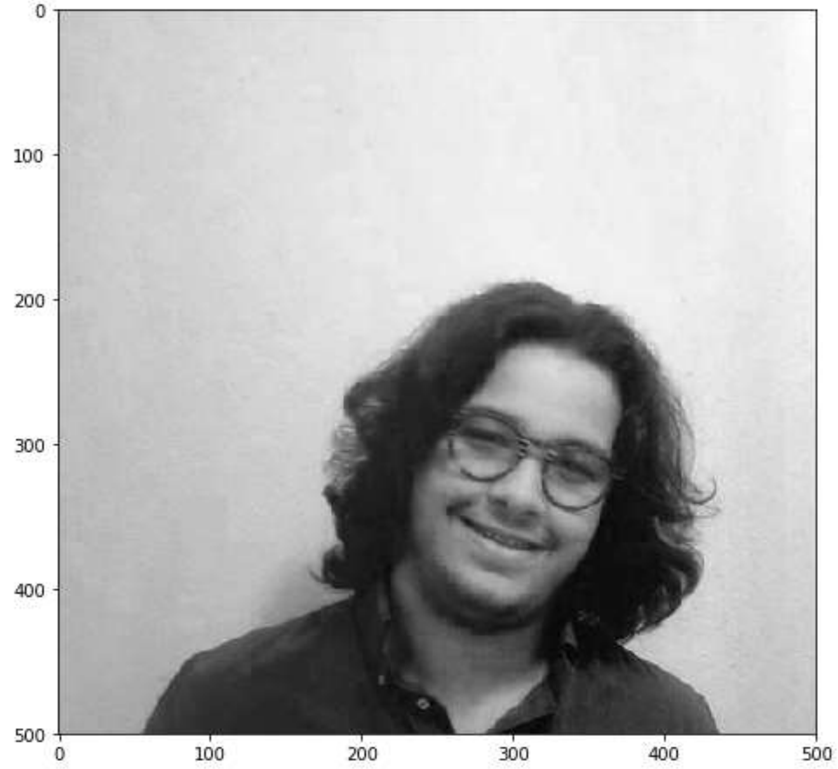
$$\frac{\partial TV(I)}{\partial I[i,j]} = \frac{2I[i,j] - I[i+1,j] - I[i,j+1]}{\sqrt{\delta + \left|I[i+1,j] - I[i,j]\right|^2 + \left|I[i,j+1] - I[i,j]\right|^2}}$$
$$+ \frac{I[i,j] - I[i-1,j]}{\sqrt{\delta + \left|I[i,j] - I[i-1,j]\right|^2 + \left|I[i-1,j+1] - I[i-1,j]\right|^2}}$$
$$+ \frac{I[i,j] - I[i,j-1]}{\sqrt{\delta + \left|I[i,j-1] - I[i+1,j-1]\right|^2 + \left|I[i,j] - I[i,j-1]\right|^2}}.$$

# VI- CONCLUSION:

Le gradient à pas optimal avec l'énergie TV






Et voilà on a bien récupéré notre image originale !!

# ANNEXES:

```
from scipy.signal import convolve2d
from math import exp,pi
import random as rd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img


def miroirH(t):
    n,p=np.shape(t)
    tt=np.zeros((n,p))

    for j in range(p):
        tt[:,j]=t[:,p-1-j]

    return tt


def histogramme(image):
    h = np.zeros(256)
    s = image.shape
    for j in range(s[0]):
        for i in range(s[1]):
            valeur = image[j,i]
            h[valeur] += 1
    return h


def negatif(t):
    return np.vectorize(lambda x: 1.0-x)(t)

def carr_lum(t,delta):
    n,p=np.shape(t)
    tt=np.zeros((n,p))
    if delta>0:
        for i in range(n):
            for j in range(p):
                tt[i,j]=min(1,t[i,j]+delta)
    else:
        for i in range(n):
            for j in range(p):
                tt[i,j]=max(0,t[i,j]+delta)
    return tt

def carr_lum2(t,delta):
    n,p=np.shape(t)

    tt=np.zeros((n,p))
    if delta>0:
        tt=np.vectorize(lambda x: min(1.0,x+delta))(t)
    else:
        tt=np.vectorize(lambda x: max(0.0,x+delta))(t)
    return tt
```

1

```
def carr_gamma(t,g):
    return np.vectorize(lambda x: x**g)(t)


def f(x,k):
    y=x+k*(x-0.5)
    if y>1:
        return 1.0
    elif y<0:
        return 0.0
    else:
        return y

def carr_lin(t,k):
    return np.vectorize(lambda x: f(x,k))(t)

def recodage(t,b):
    return np.vectorize(lambda x: int(2**b*int(255*x)/256)/(2**b-1))(t)

def reduction_taille1(t,k):
    n,p=np.shape(t)
    nr,pr=n//k,p//k
    tr=np.zeros((nr,pr))
    i=0
    while i<nr:
        j=0
        while j<pr:
            tr[i,j]=t[k*i,k*j]
            j+=1
        i+=1
    return tr

def reduction_taille2(t,k):
    n,p=np.shape(t)
    nr,pr=n//k,p//k
    tr=np.zeros((nr,pr))
    i=0
    while i<nr:
        j=0
        while j<pr:
            tr[i,j]=np.mean(t[k*i:k*(i+1),k*j:k*(j+1)])
            j+=1
        i+=1
    return tr


def agrandissement(t,k):
    n,p=np.shape(t)
    na,pa=n*k,p*k
    ta=np.zeros((na,pa))
    bloc=np.ones((k,k))
    for i in range(n):
        for j in range(p):
            ta[k*i:k*(i+1),k*j:k*(j+1)]=t[i,j]*bloc
    return ta
```

2

```
%pylab inline
gray() # pour afficher toutes les images en noir et blanc
figsize(16,8) #pour afficher les images en grand format
```


```
%pylab inline
gray() # pour afficher toutes les images en noir et blanc
figsize(16,8) #pour afficher les images en grand format
```


```
h,w= shape(Im)
Im_alea = rand(h,w)
imshow(Im_alea)
```

```
def Delta(I):
    Ix = roll(I,1,1) + roll(I,-1,1) - 2*I
    Iy = roll(I,1,0) + roll(I,-1,0) - 2*I
    return Ix + Iy
```

```
subplot(121)
imshow(Delta(Im))
subplot(122)
imshow(Delta(Im_alea))
```

```
def H1(I) :
    return sqrt(-sum(I*Delta(I)))
def H1bis(I) :
    return sqrt(sum( (roll(I,1,1) - I)**2 + (roll(I,1,0) - I)**2))

print("H1(Im) = ", H1(Im))
print("H1(Im_alea) = ", H1(Im_alea))
print('''On observe un facteur 3 entre l'énergie H1 d'une image "normale", et celle d'une image aléatoire.''')
```

```python
#Vérification, ne pas modifier
assert( abs(H1(Im) - 80.5412) < 1e-3) , "problème dans la fonction H1"
```

```python
def P(I):
    return M*Im + (1-M)*I
imshow(P(Im_alea))
```

```python
def dH1(I):
    DeltaI = Delta(I)
    return 1/sqrt(-sum(I*DeltaI))*(-DeltaI)
```

```python
def gradientProjete(dH, I0, tau, tol=1e-4, Niter=200):
    In = I0
    for n in range(Niter):
        Inp1 = P( In-tau*dH(In) )
        if norm(Inp1 - In) < tol:
            return Inp1
        In = Inp1
    print("Problème, l'algorithme n'a pas convergé après", Niter, "itérations")
    return Inp1
```

```python
Istar = gradientProjete(dH1, Im,10)
imshow(Istar)
```

```python
delta = 1e-1
def TV(I):
    return sum(sqrt(delta + (roll(I,-1,0) - I)**2 + (roll(I,-1,1) - I)**2))
print("TV(Im) = ", TV(Im))
print("TV(Im_alea) = ", TV(Im_alea))
```

```python
def dTV(I):
    J1 = (2*I - roll(I,-1,0) - roll(I,-1,1))/sqrt(delta + (roll(I,-1,0) - I)**2 + (roll(I,-1,1) - I)**2)
    Iim1 = roll(I,1, 0)
    J2 = (I - Iim1)/sqrt(delta + (I - Iim1)**2 + (roll(Iim1, -1, 1) - Iim1)**2)
    Ijm1 = roll(I,1, 1)
    J3 = (I - Ijm1)/sqrt(delta + (Ijm1 - roll(Ijm1, -1, 0))**2 + (I - Ijm1)**2)
    return J1 + J2 + J3
```

```python
subplot(121)
imshow(dTV(Im))
subplot(122)
imshow(dTV(Im_alea))
```

```python
IstarTV = gradientProjete(dTV, Istar, 0.05, tol=1e-10, Niter=1000)
imshow(IstarTV)
```

```python
imshow(dTV(IstarTV))
```