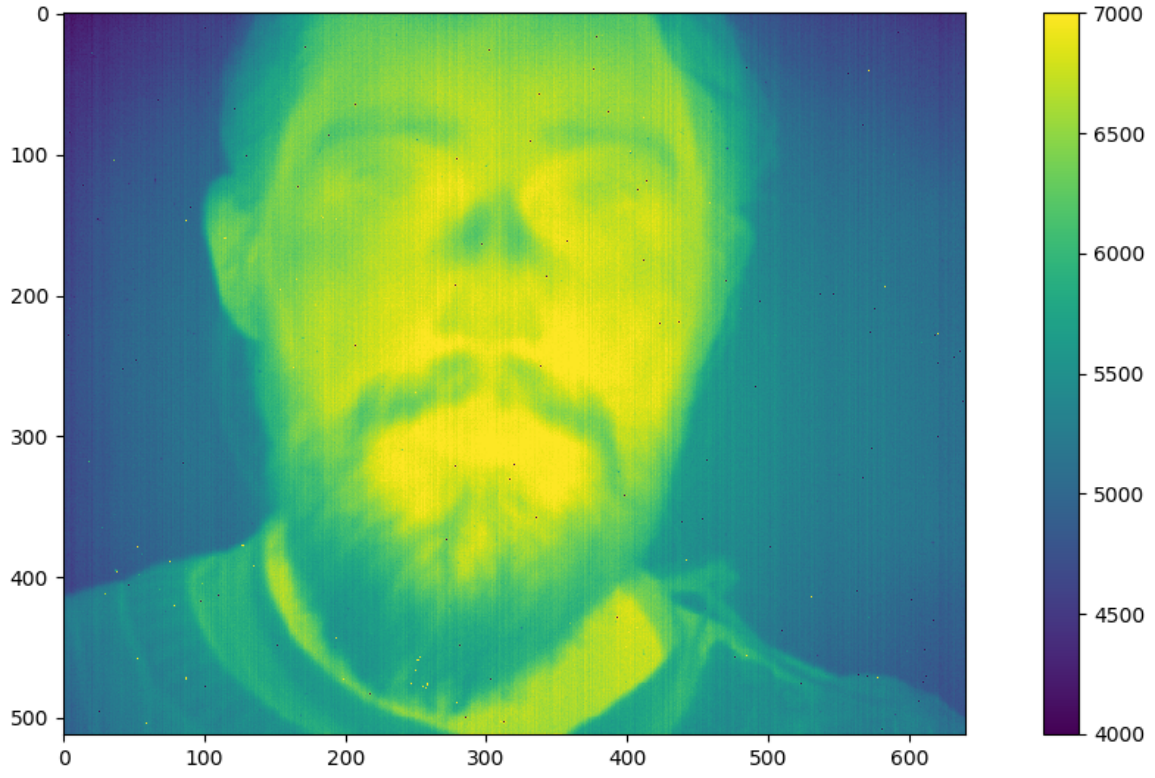
TALEC-BENARD Nicolas DENES Marie	Systemes de perception	FIPA_2020 31/10/2019

BE Camera infrarouge

Partie 1: affichage de l'image brute

```
import numpy as np
import matplotlib.pyplot as plot

if __name__ == "__main__" :

    data = np.loadtxt('row_image.dat')
    plot.figure(1)
    plot.subplot(2,2,1)
    plot.imshow(data, vmin = 4000, vmax = 7000)
    plot.title("Image brute")
    plot.colorbar()
```

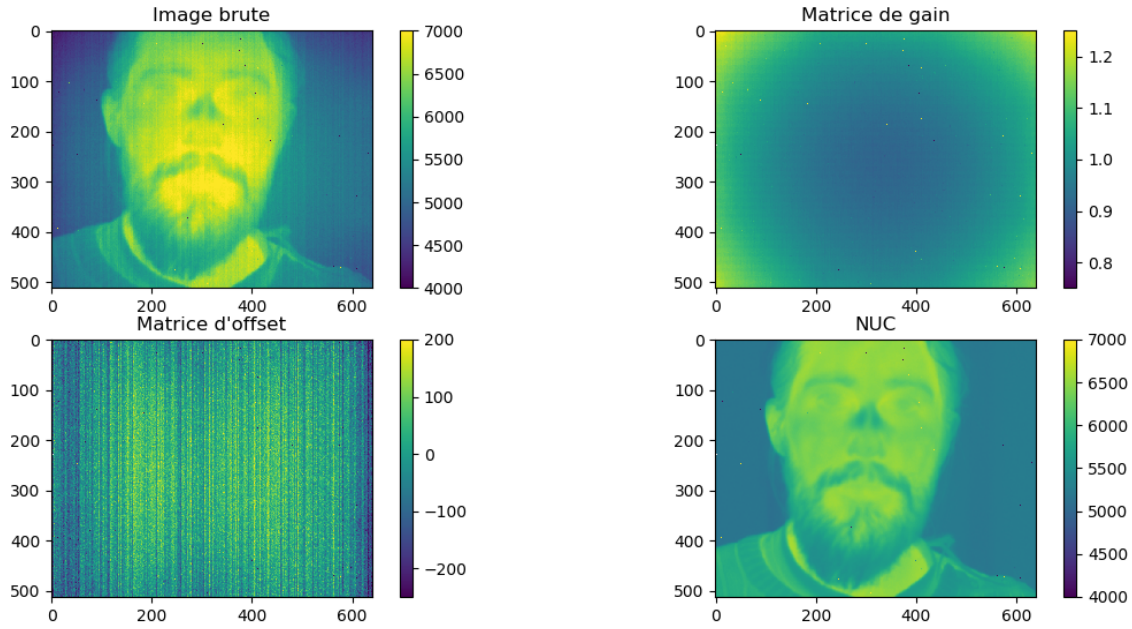


Nous obtenons l'image suivante. Nous remarquons que l'image a des pixels avec des valeurs défaillantes. Ils sont jaunes dans des zones vertes.

Partie 2: choisir deux températures & calculer NUC à deux points

Par choix, nous prenons des images proches de la température ambiante à l'intérieur (entre 18°c et 26°c).

```
T1 = np.loadtxt('temp_bb_18C.dat')
T2 = np.loadtxt('temp_bb_26C.dat')

alpha =  (np.mean(T1)-np.mean(T2))/(T1-T2)

plot.subplot(2,2,2)
plot.imshow(alpha,vmin = 0.75,vmax = 1.25) # valeurs de pixels
plot.title("Matrice de gain")
plot.colorbar()

beta =  np.mean(T1)-alpha*T1

plot.subplot(2,2,3)
plot.imshow(beta, vmin = -300,vmax = 200)   # valeurs de la colorbar
plot.title("Matrice d'offset")
plot.colorbar()

NUC = alpha * data + beta
plot.subplot(224)
plot.imshow(NUC, vmin=4000, vmax=7000)
plot.title("NUC")
plot.colorbar()
```



Partie 3: mettre en place un algorithme de détection de pixels défectueux

```
calib = np.where((alpha < 0.75) | (alpha > 1.25) | (beta < -5000) | (beta > 5000),0,1)

alpha[np.where((alpha < 0.75) | (alpha > 1.25) | (beta < -5000) | (beta > 5000))] = 1.

beta[np.where((alpha < 0.75) | (alpha > 1.25) | (beta < -5000) | (beta > 5000))] = 0.

print("Pixels defectueux")
print(calib)
```

Nous obtenons une matrice de pixels défaillants grâce à la fonction np.where() en les mettant à 0 quand ils sont défaillants. Nous avons mis les conditions de pixels défaillants du cours qui sont :

- les pixels de la matrice de gain doivent être compris entre 0.75 et 1.25
- les pixels de la matrice d'offset doivent être compris entre -5000 et 5000

Ensuite, pour éviter d'avoir des retours de NaN (not a number) dans nos matrices, nous forçons le pixel défaillant dans la matrice de gain à prendre la valeur 1. Cette valeur correspond à la valeur moyenne théorique d'une matrice de gain.

Avec le même raisonnement, nous mettons la valeur du pixel défaillant à 0 pour la matrice d'offset.

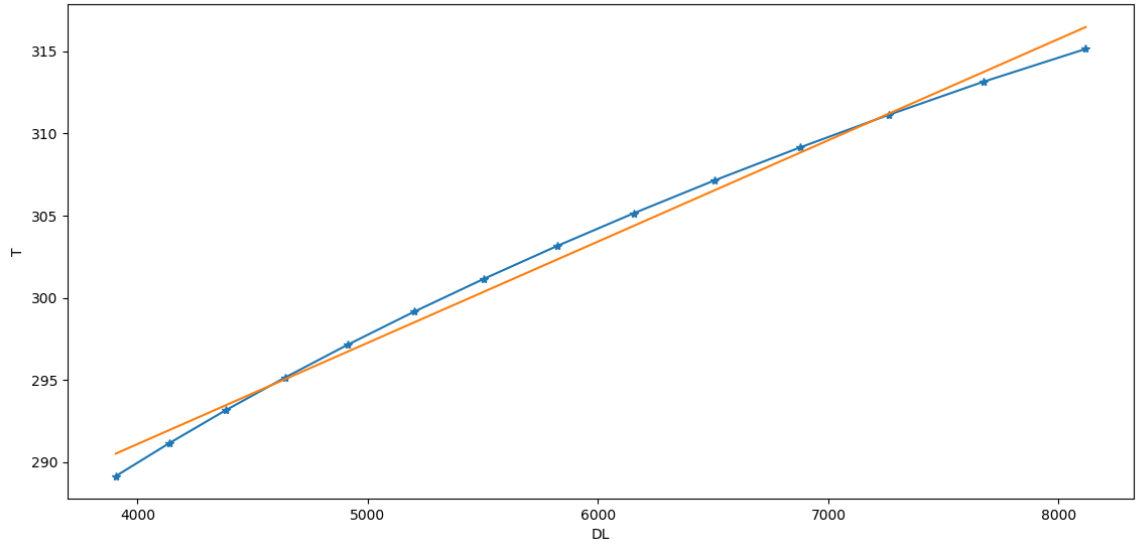
Partie 4: évaluer le loi de type Planck (avec offset) permettant de convertir une image brute (après application de la NUC) en une cartographie de température

```
DL = []
for temp in range (16, 42, 2):
    file = 'temp_bb_{}C.dat'.format(temp)
    data = np.loadtxt(file)
    #print(data)
    #print(alpha.min(), alpha.max(), beta.min(), beta.max())
    nuc_data = alpha*data + beta
    #print(nuc_data)
    gamma = np.sum(nuc_data*calib)
    gamma /= np.sum(calib)

    DL.append(gamma)

print(DL)
```

Nous effectuons une boucle pour récupérer l'ensemble des mesures en effectuant une moyenne en tenant compte d'un masque qui ne prendra pas en compte les pixels défaillants dans le calcul.

Nous affichons ensuite la courbe que DL en fonctions de nos températures. Puis, nous utilisons la fonction

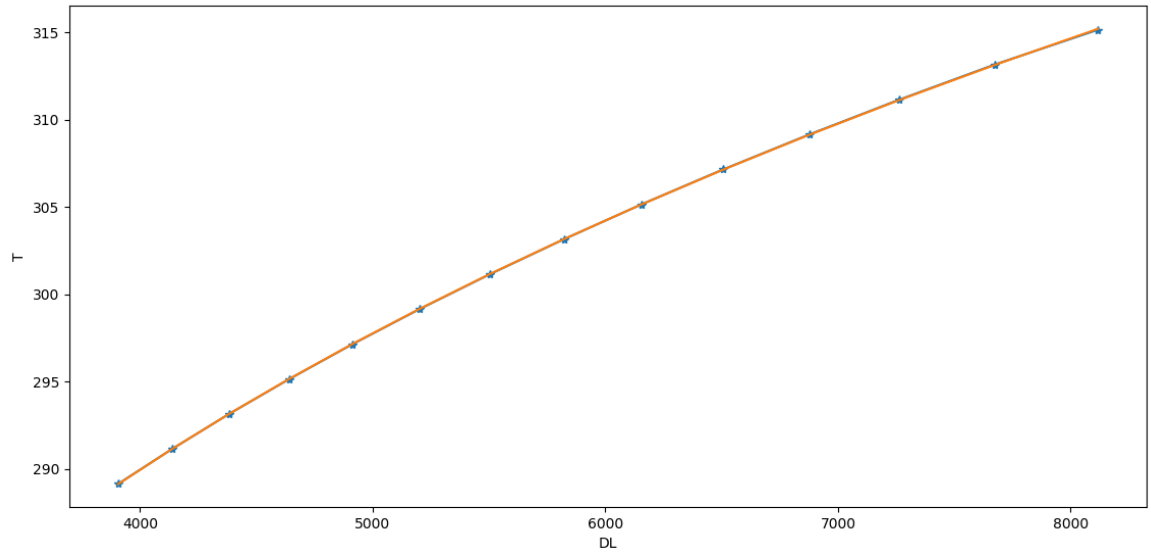
```
ydata = np.arange(16,44,2)+ 273.15
xdata = DL
plot.figure(3)

plot.plot(xdata,ydata,'-*')
plot.xlabel("DL")
plot.ylabel("T en °K")

R, B = optimize.curve_fit(planck, xdata, ydata)
plot.plot(xdata, planck(xdata, *R))
print(R)
```



On obtient une mauvaise optimisation de la courbe. En effet, la matrice obtenue retourne une valeur négative. On va donc rajouter dans le code des limites pour inciter la fonction optimize à choisir des valeurs non aberrantes.

```
ydata = np.arange(16,44,2)+ 273.15
xdata = DL

plot.figure(3)

plot.plot(xdata,ydata,'-*')
plot.xlabel("DL")
plot.ylabel("T en °K")

p0 = np.array([5e7, 1e3, 1e3])
bounds = (0, np.inf)

R, B = optimize.curve_fit(planck, xdata, ydata, p0=p0, bounds=bounds)
plot.plot(xdata, planck(xdata, *R))
print(R)
```

Comme le montre l'image ci-dessous, la courbe est parfaitement suivie.



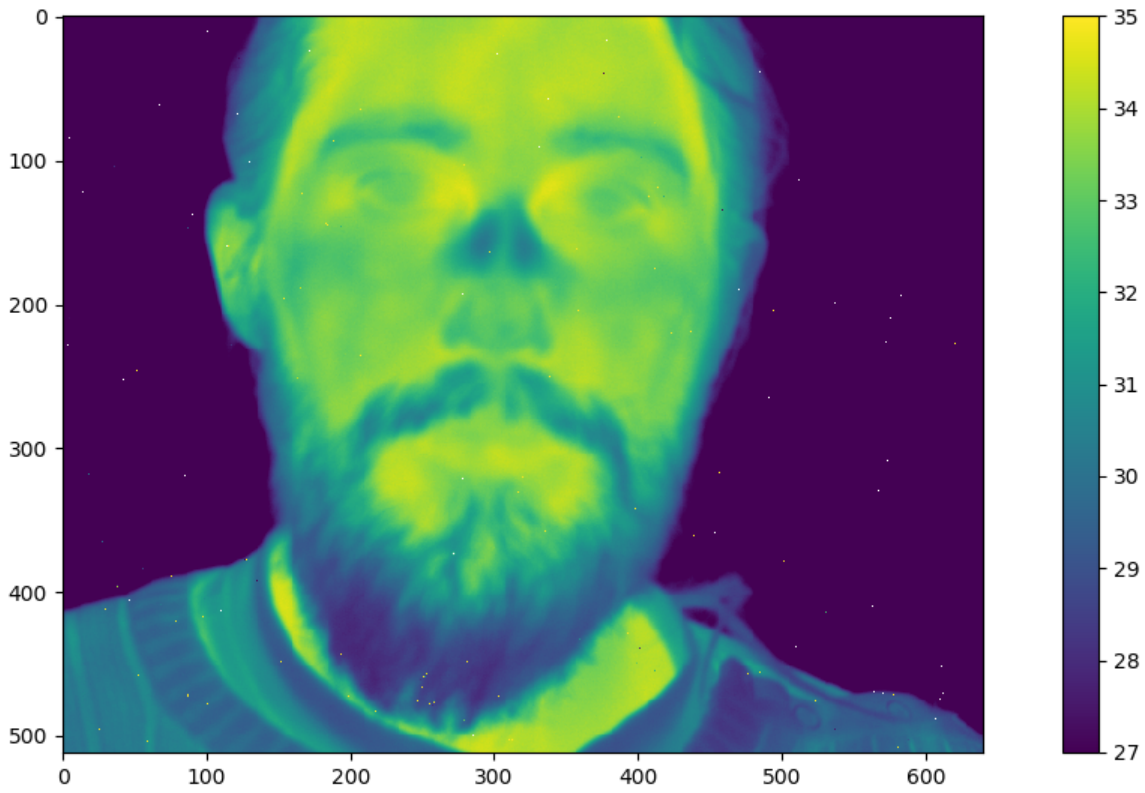
Partie 5: appliquer la chaîne numérique complète sur l'image fournie, i.e. calculer une jolie image partant de l'image brute, et identifier ce qu'elle représente

Afin d'obtenir une image avec la température, nous utilisons le code suivant :

```
image = planck(NUC, *R)
image -= 273.15
plot.figure(5)
plot.imshow(image, vmin=27, vmax=35)
plot.colorbar()
```

Nous reprenons les paramètres calculés avec l'optimisation de la courbe par notre fonction Planck.

Après plusieurs tests, l'image est la plus nette avec une limite mise entre 27°c et 35°c. Il y a deux zones de température.



Sur l'image, il y a des pixels défectueux car l'image n'a pas été corrigée. Par ailleurs, les données vues sur cette représentation ne sont pas totalement vraies. En effet, nous avons fait les calculs en admettant que le visage était un corps noir avec :

$$\epsilon = 1$$

Le visage s'approche d'un corps noir mais n'en est pas un. Donc, il faudrait corriger cela en partant sur les caractéristiques d'émissivité du visage et de température ambiante. Ainsi, il n'y aurait pas plus de réflexion.