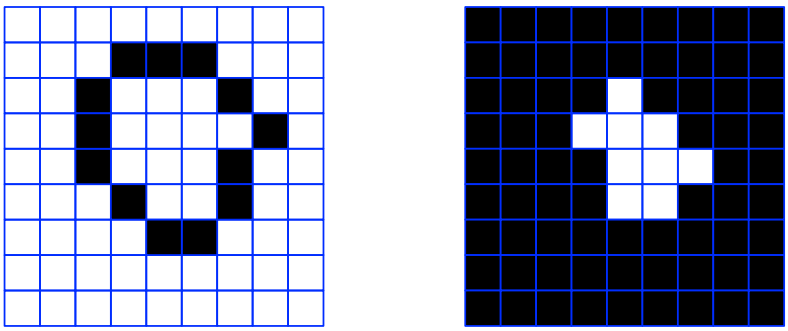
Extraction de régions

1. Introduction

Une région est un ensemble connexe de pixels ayant le même niveau, délimités par un contour ou par une région de pixels de niveaux différents. La figure suivante montre une région de pixels clairs délimitée par un contour plus sombre, et une région constituée d'une tache claire sur un fond sombre.

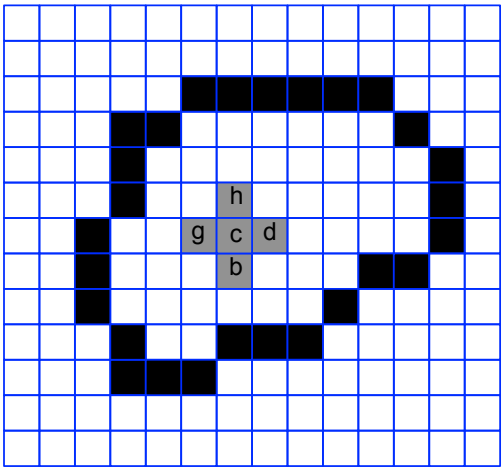


L'extraction d'une région consiste à répertiorer les pixels d'une région afin d'étudier ses caractéristiques géométriques (nombre de pixels, barycentre, etc).

2. Remplissage par diffusion

2.a. Principe

Pour parcourir les pixels d'une région, il faut faire un remplissage de la région, qui consiste à modifier les valeurs de ses pixels au fûr et à mesure de leur rencontre. Sur la figure suivante, les pixels sont coloriés en gris. Pour chaque pixel (c) déjà colorié, on doit colorier les quatre pixels voisins situés à gauche (g), à droite (d), en haut (h) et en bas (b).



2.b. Implémentation récursive

```
import imageio
import numpy
from matplotlib.pyplot import *
import sys
```

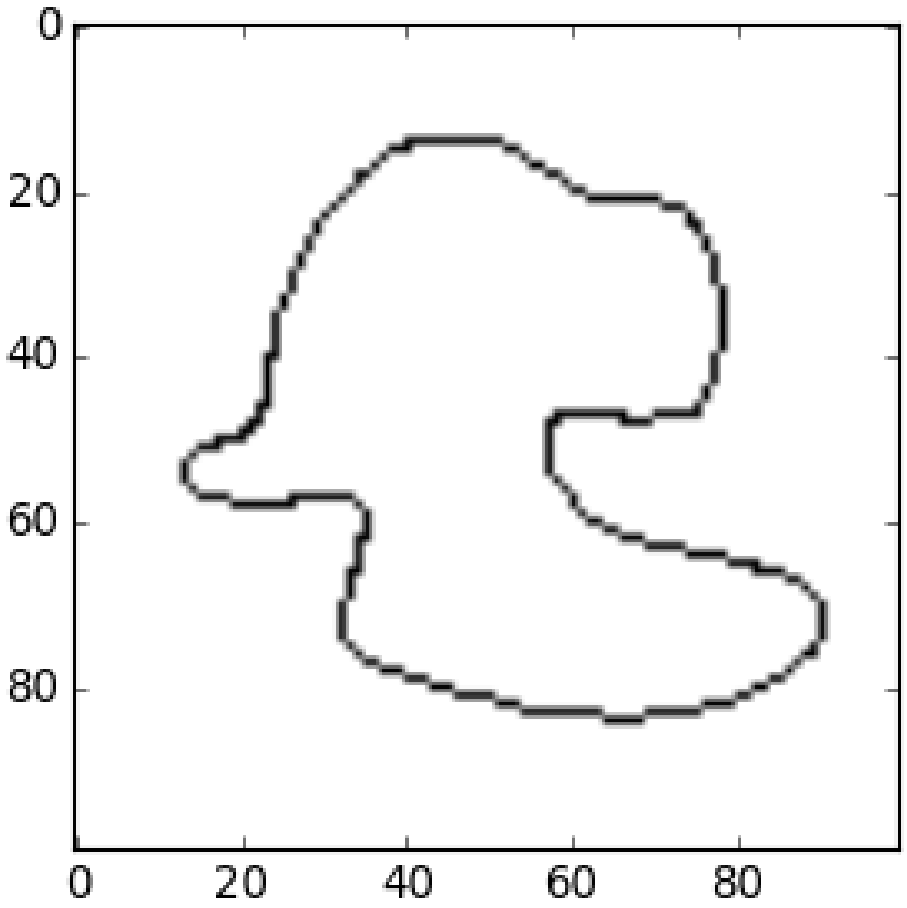
L'algorithme de remplissage par diffusion se prête à une implémentation récursive. On définit pour cela une fonction de coloriage d'un pixel, qui explore les quatre voisins et appelle récursivement la fonction de coloriage sur les voisins qui ne sont pas déjà coloriés. La fonction suivante effectue le coloriage d'un pixel si sa valeur dépasse un seuil fourni.

```
def coloriage_pixel(image,shape,seuil,valeur,i,j):
    image[j][i] = valeur
    voisins = [(i+1,j),(i-1,j),(i,j-1),(i,j+1)]
    for pixel in voisins:
        (k,l) = pixel
        if k>=0 and k<shape[1] and l>=0 and l<shape[0]:
            if image[l][k]>seuil:
                coloriage_pixel(image,shape,seuil,valeur,k,l)
```

Voici un exemple. On charge l'image et on extrait sa couche rouge

```
img = imageio.imread("../../../../figures/image/extraction/regions/contours3.png")
rouge = img[:,:,0]
vert = img[:,:,1]
bleu = img[:,:,2]
figure(figsize=(4,4))
```

```
imshow(rouge,cmap='gray',vmin=0,vmax=255)
```

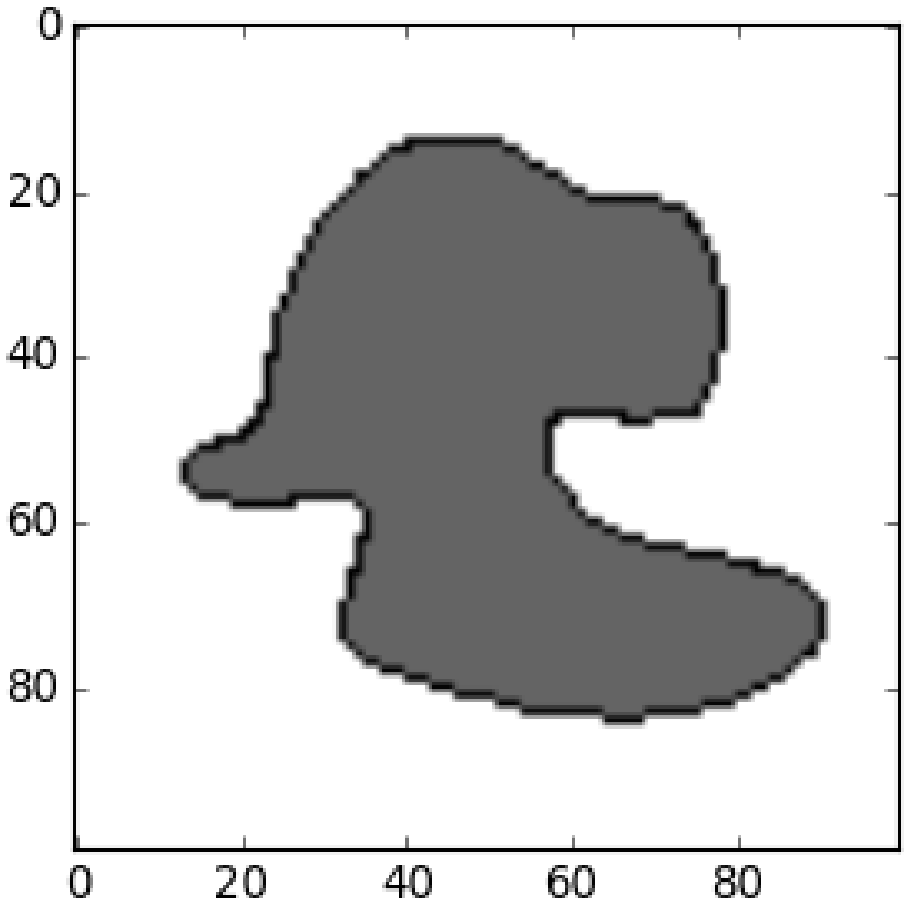


Par défaut, le nombre de récursions permis est insuffisant. Il faut donc l'augmenter, sachant que le nombre de récursions nécessaire est au maximum égal au nombre de pixels de l'image.

```
sys.setrecursionlimit(rouge.size)
```

Voici le remplissage :

```
image = rouge.copy()
coloriage_pixel(image,image.shape,110,100,40,40)
figure(figsize=(4,4))
imshow(image,cmap='gray',vmin=0,vmax=255)
```



2.c. Implémentation avec une pile de voisins

La récursion n'est pas toujours disponible. D'ailleurs, la fonction ci-dessus ne fonctionne pas pour les grandes images. De plus, l'implémentation récursive rend plus difficile les opérations d'analyse de la région de pixels. On préfère donc remplacer la récursion par une implémentation itérative au moyen d'une pile de voisins. Pour chaque pixel colorié, on explore les quatre voisins et on place ceux qui doivent être coloriés dans une pile. Les pixels de la pile sont coloriés et le calcul s'arrête lorsque la pile est vide. Voici tout d'abord la fonction qui colorie un pixel et complète la pile :

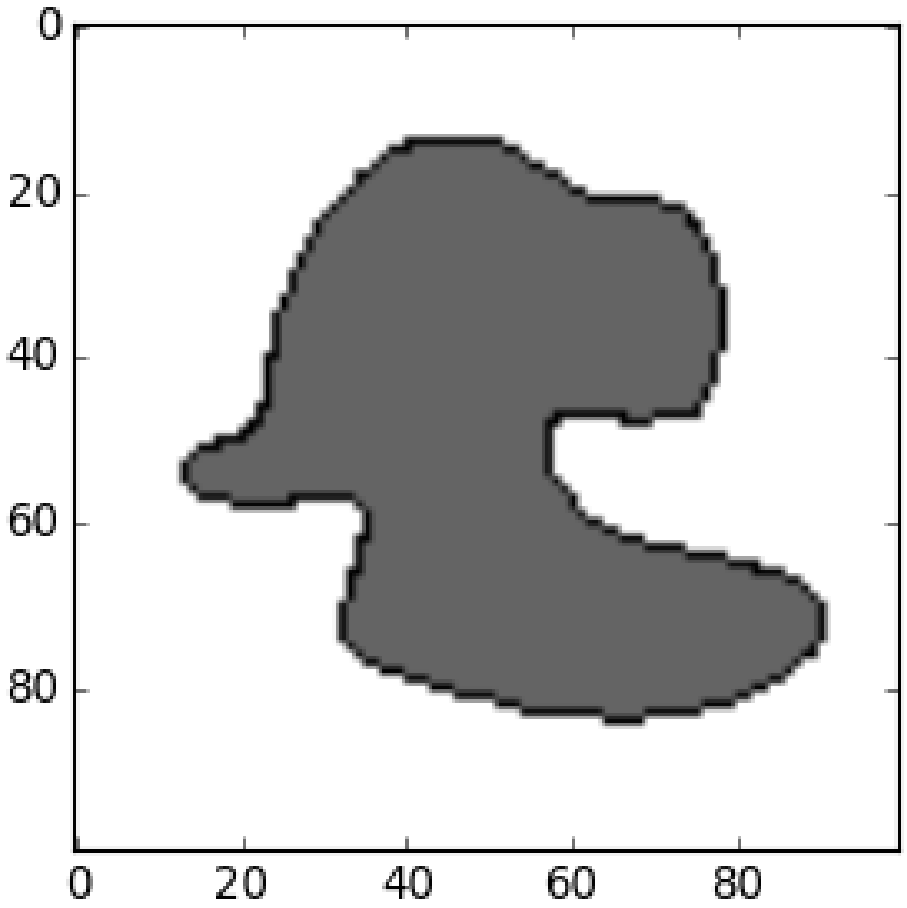
```
def coloriage_pixel_pile(image,shape,seuil,valeur,pile,i,j):
    image[j][i] = valeur
    voisins = [(i+1,j),(i-1,j),(i,j-1),(i,j+1)]
    for pixel in voisins:
         (k,l) = pixel
         if k>=0 and k<shape[1] and l>=0 and l<shape[0]:
             if image[l][k]>seuil:
                 image[l][k] = valeur
                 pile.append(pixel)
```

La fonction de coloriage de la région initialise la pile avec le premier pixel à colorier puis appelle la fonction précédente de manière itérative jusqu'à épuisement de la pile :

```
def coloriage_region(image,shape,seuil,valeur,i0,j0):
    if valeur>seuil:
        seuil = valeur
    pile = [(i0,j0)]
    while len(pile)>0:
        (i,j) = pile.pop()
        coloriage_pixel_pile(image,shape,seuil,valeur,pile,i,j)
```

Voici un exemple :

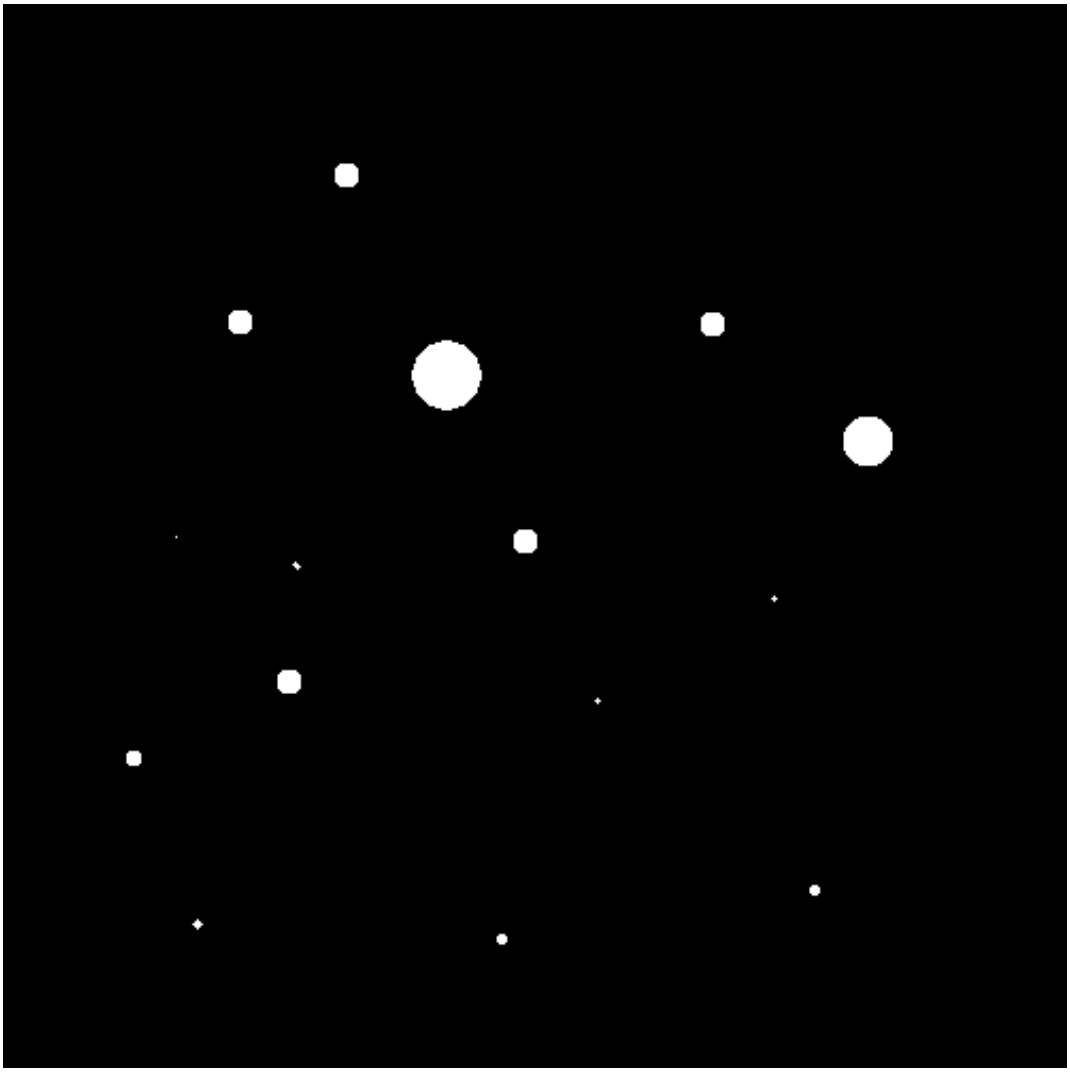
```
image = rouge.copy()
coloriage_region(image,image.shape,110,100,40,40)
figure(figsize=(4,4))
imshow(image,cmap='gray',vmin=0,vmax=255)
```



3. Extraction de taches

3.a. Algorithme et fonctions

On considère comme exemple l'image binaire suivante :



Elle comporte des taches plus ou moins grosses, dont la forme est d'ailleurs quelconque. Il s'agit d'énumérer ces taches et de déterminer le barycentre de chacune d'elle.

Pour cela, on parcourt l'image par lignes successives. Dès que l'on rencontre un pixel blanc, on entre dans une boucle qui effectue le coloriage de la région liée à ce pixel et calcule le barycentre.

Voici la fonction de coloriage d'un pixel qui utilise une pile. Elle est identique à celle définie plus haut mais on a ajouté le coloriage d'une image secondaire (noire au départ), qui permettra de visualiser le résultat du parcours.

```
def coloriage_pixel_pile(image,result,shape,seuil,valeur,pile,i,j):
    image[j][i] = valeur
    result[j][i] = 255
    voisins = [(i+1,j),(i-1,j),(i,j-1),(i,j+1)]
    for pixel in voisins:
        (k,l) = pixel
```

```
        if k>=0 and k<shape[1] and l>=0 and l<shape[0]:
            if image[l][k]>seuil:
                image[l][k] = valeur
                pile.append(pixel)
```

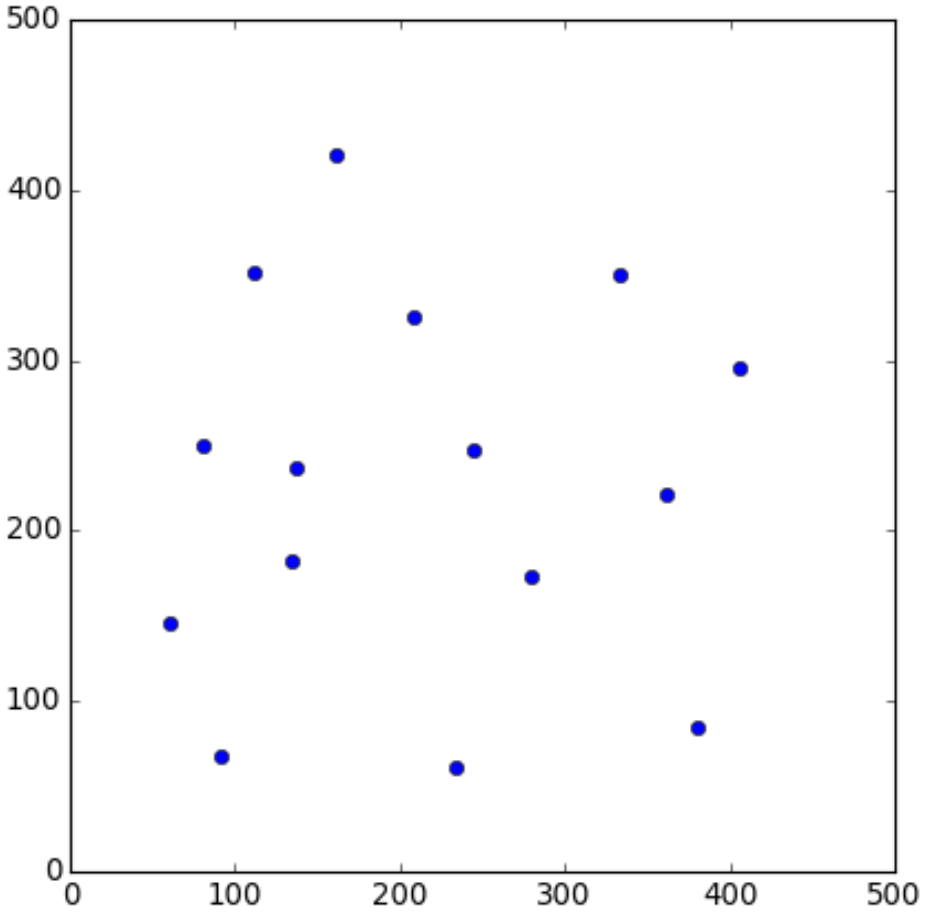
La fonction suivante effectue l'analyse complète de l'image. Les pixels dont la valeur est supérieure à un seuil sont pris en compte dans l'extraction. La fonction renvoit une liste des positions des barycentres avec le nombre de pixels de chaque tache.

```
def analyse(image,seuil):
    shape = image.shape
    Nx = shape[1]
    Ny = shape[0]
    compteur = 0
    positions = []
    result = numpy.zeros((Ny,Nx),dtype=numpy.uint8)
    for y in range(Ny):
        for x in range(Nx):
            if image[y][x] > seuil:
                compteur += 1
                pile = [(x,y)]
                si = 0
                sj = 0
                npix = 0
                while len(pile)>0:
                    (i,j) = pile.pop()
                    si += i
                    sj += j
                    npix += 1
                    coloriage_pixel_pile(image,result,shape,seuil,0,pile,i,j)
                xb = si*1.0/npix
                yb = sj*1.0/npix
                positions.append((xb,yb,npix))
    print("Nombre de taches : %d"%compteur)
    for p in positions:
        print("x=%f, y=%f, npix=%d"%(p[0],p[1],p[2]))
    return (positions,result)
```

Voici le traitement de l'image montrée plus haut :

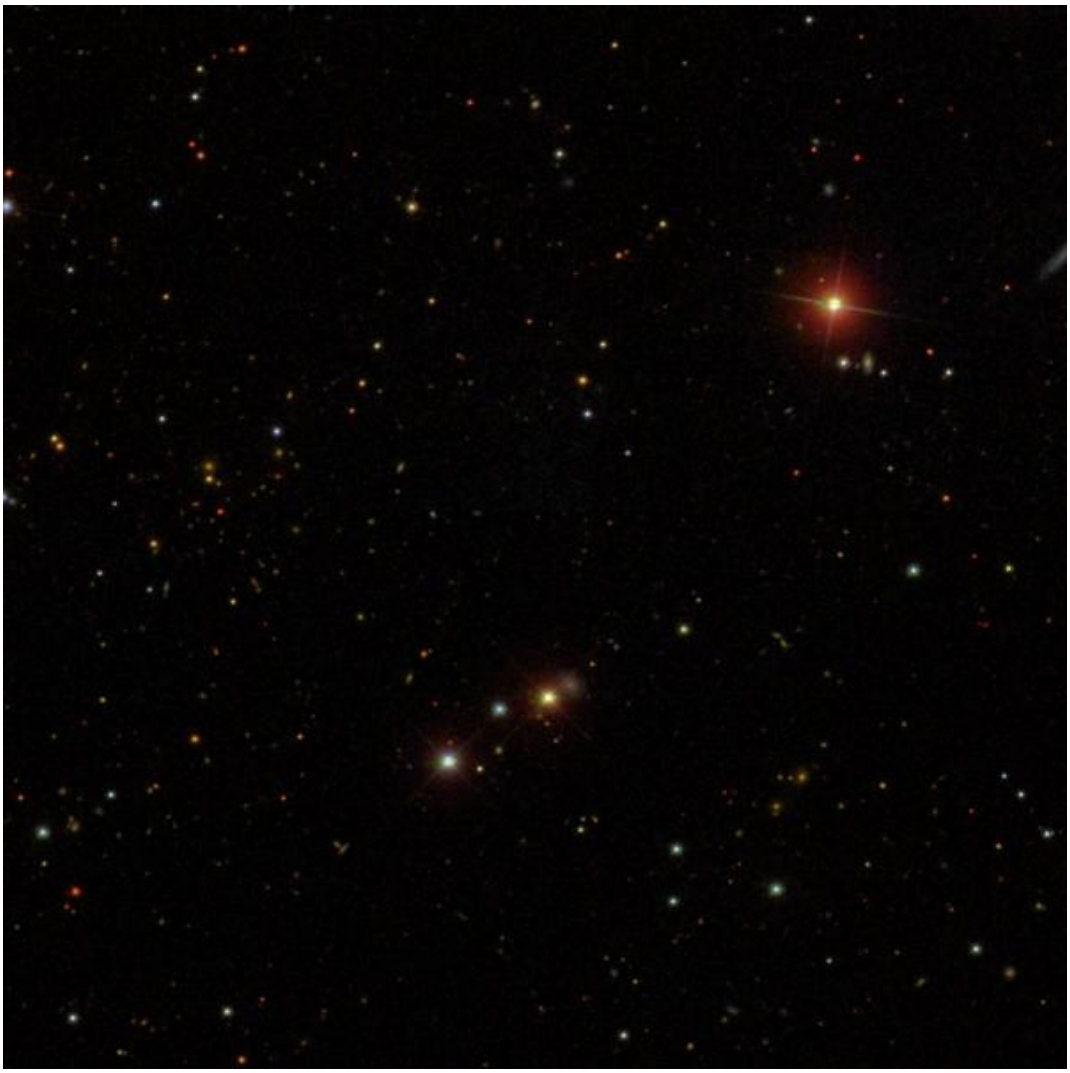
```
img = imageio.imread("../../../../figures/image/extraction/regions/points.png")
rouge = img[:,:,0]
vert = img[:,:,1]
bleu = img[:,:,2]
image = rouge.copy()
(positions,result) = analyse(image,130)
```

```
shape = image.shape
figure(figsize=(6,6))
for p in positions:
    plot([p[0]],[shape[0]-p[1]],'ob')
axis([0,shape[1],0,shape[0]])
```



3.b. Application : analyse d'un champ stellaire

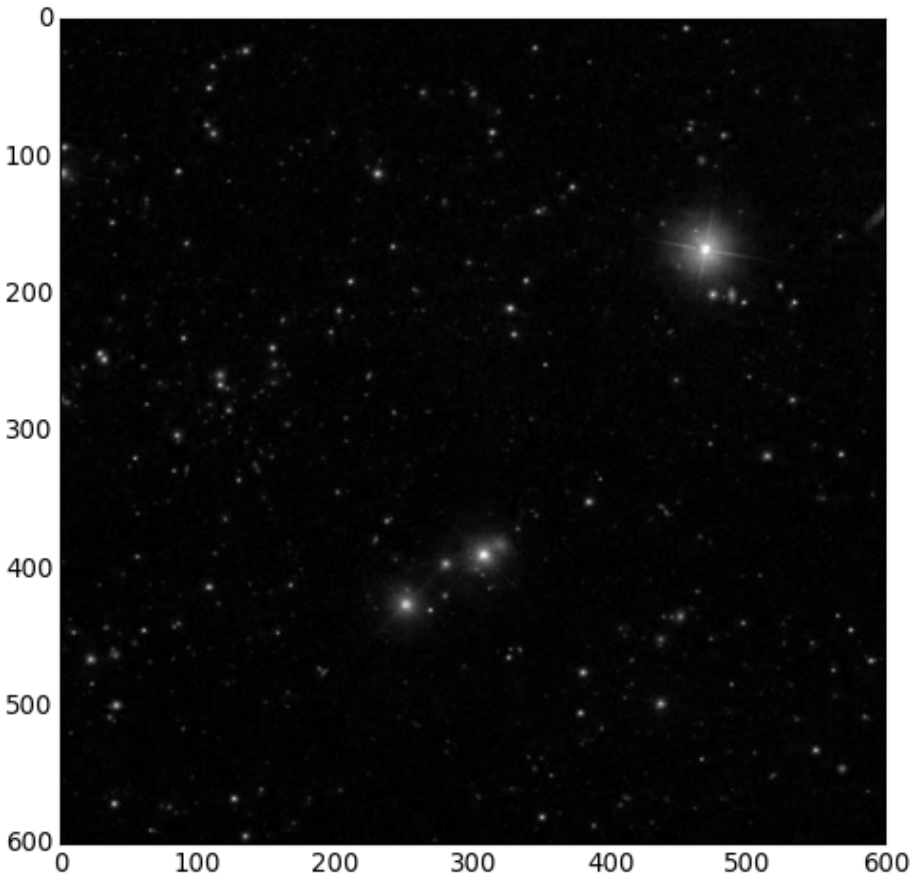
L'image suivante a été obtenue sur la base de données astronomiques Sloan Digital Sky Survey. Il s'agit d'un champ carré de 10 minutes d'arc de large, centré sur le point d'ascension droite 150 degrés et de déclinaison 0 degrés.



L'image contient principalement des étoiles et quelques galaxies. Elle a 600 par 600 pixels. Les taches sont à peu près circulaires, même si la diffraction due à la fixation du miroir secondaire est visible sur les étoiles les plus lumineuses. La tache d'une étoile est d'autant plus grande qu'elle est lumineuse.

On commence par extraire la couche rouge :

```
img = imageio.imread("../../../../figures/image/extraction/regions/sky3.jpeg")
rouge = img[:,:,0]
vert = img[:,:,1]
bleu = img[:,:,2]
figure(figsize=(7,7))
imshow(rouge,cmap='gray',vmin=0,vmax=255)
```



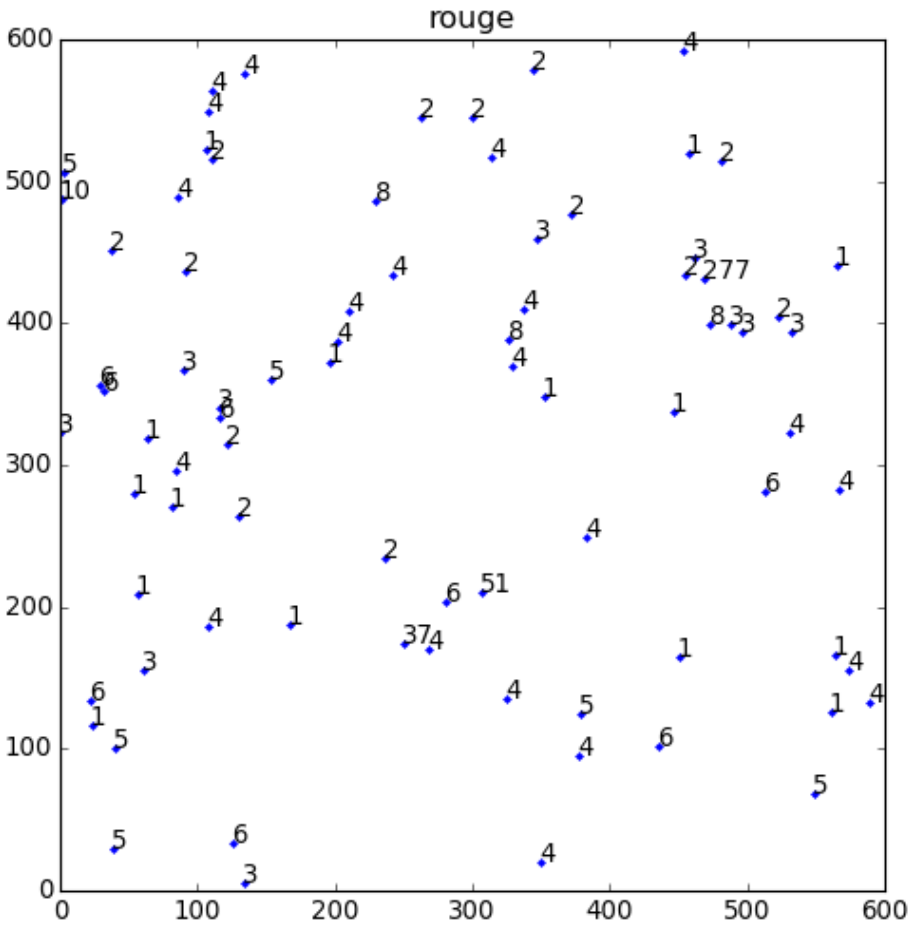
Les étoiles sont toutes visibles sur cette couche, même les étoiles de couleur bleue.

On analyse l'image et on trace les positions des étoiles avec le nombre de pixels de la tache correspondante, qui donne une indication de la luminosité sur la couche rouge :

```
image = rouge.copy()
(positions,result) = analyse(image,130)

shape = image.shape
figure(figsize=(7,7))
for p in positions:
    x = p[0]
    y = shape[0]-p[1]
    plot([x],[y],'.b')
    text(x,y,"%d"%p[2])

axis([0,shape[1],0,shape[0]])
title("rouge")
```

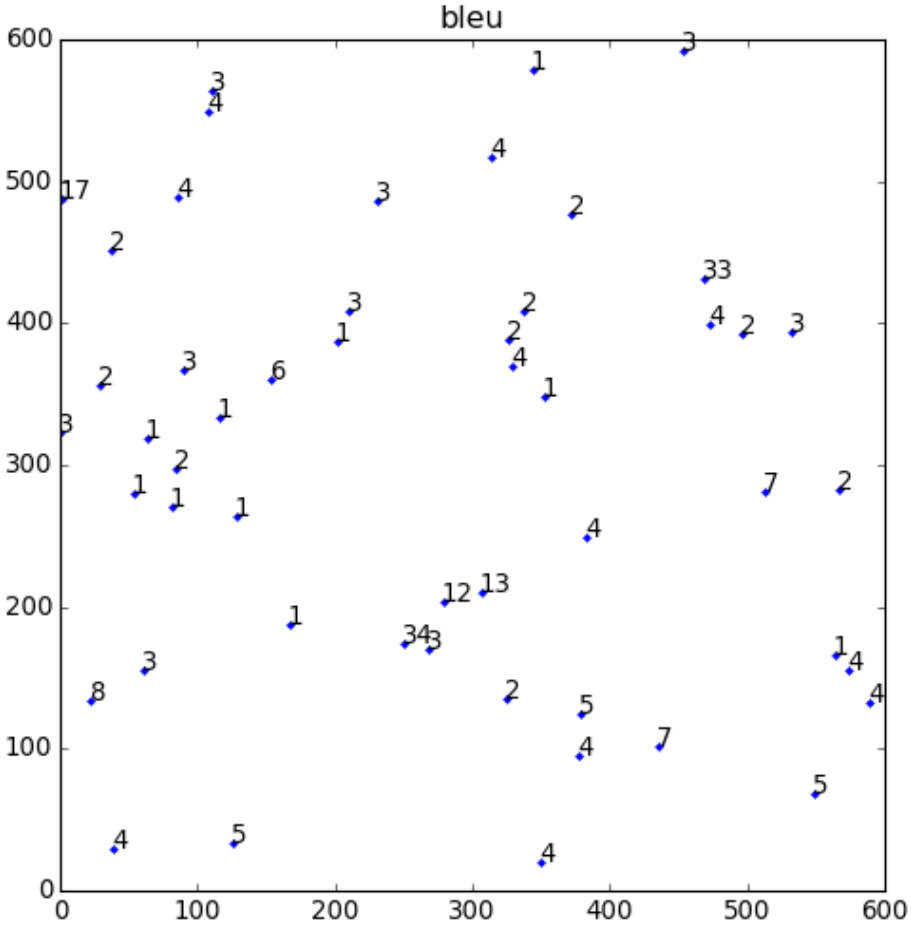


Il est intéressant de faire le même traitement sur la couche bleue :

```
image = bleu.copy()
(positions,result) = analyse(image,130)

shape = image.shape
figure(figsize=(7,7))
for p in positions:
    x = p[0]
    y = shape[0]-p[1]
    plot([x],[y],'.b')
    text(x,y,"%d"%p[2])

axis([0,shape[1],0,shape[0]])
title("bleu")
```



On voit par exemple que la tache de l'étoile la plus lumineuse comporte 506 pixels sur la couche rouge et seulement 54 pixels sur la couche bleue, ce qui est bien sûr une conséquence de sa forte couleur rouge visible sur l'image RVB. À l'inverse, on repère facilement des étoiles bleues dont la tâche a plus de pixel sur la couche bleue.

La base de données Sloan Digital Sky Survey peut être interrogée directement par des requêtes SQL. L'utilitaire `sqlcl.py` permet de faire ces requêtes depuis un code python ou en ligne de commande.

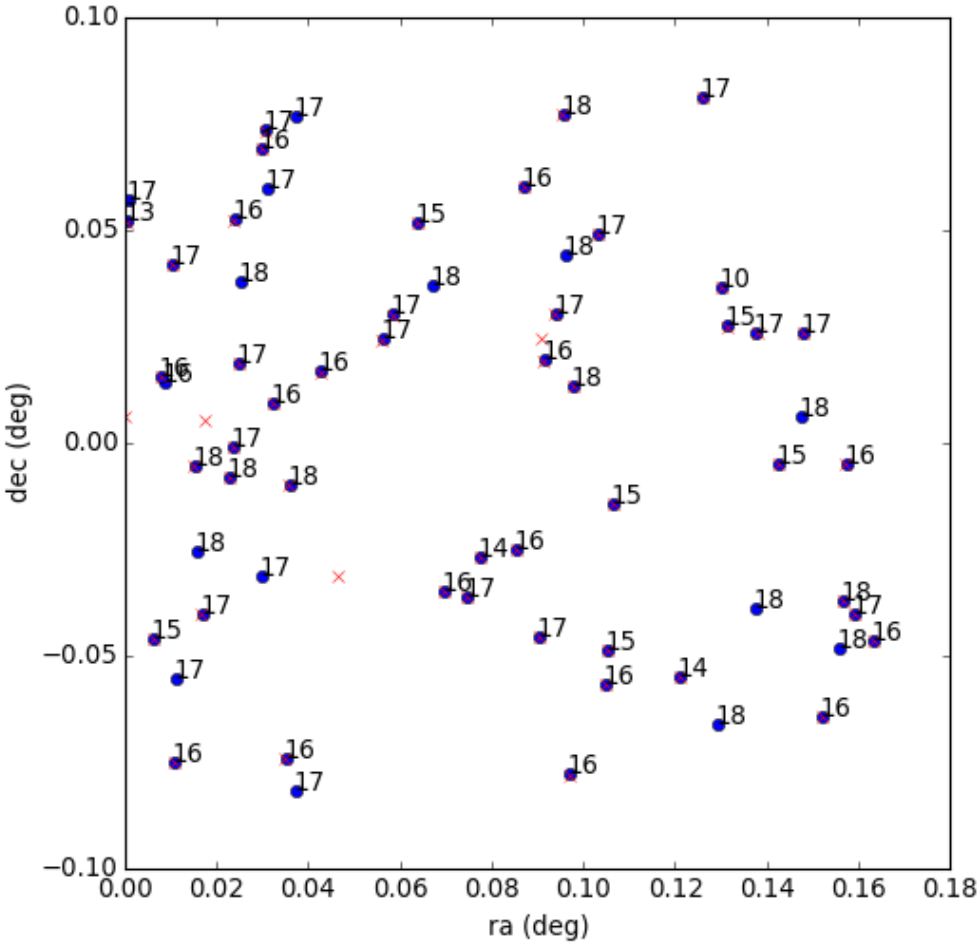
La requête ci-dessous permet de récupérer des étoiles dans le champ stellaire identique à celui de l'image traitée plus haut. On récupère la position de ces étoiles et les magnitudes pour les couleurs U (ultraviolet), G (vert), R (rouge) et I (infrarouge proche). On trace les positions avec la valeur entière de la magnitude rouge.

```
import sqlcl
import StringIO

ra = 150.0
dec = 0.0
width = 1.0/60*10.0
height = 1.0/60*10.0
ra1 = ra-width/2
```

```
ra2 = ra+width/2
dec1 = dec-height/2
dec2 = dec+height/2
"""
Magnitudes :
    U : 0.36 micrometres
    R : 0.64
    G : 0.52
    I : 0.79
"""
requeteSQL = "SELECT  ra,dec,psfMag_u,psfMag_g,psfMag_r,psfMag_i  FROM star\
            WHERE (ra BETWEEN %f AND %f)\
            AND (dec BETWEEN %f AND %f)\
            AND (psfMag_r BETWEEN 1 AND 19)"%(ra1,ra2,dec1,dec2)
result = sqlcl.query(requeteSQL).read()
print(result)
result = result.replace(',',' ')
data = numpy.loadtxt(StringIO.StringIO(result),skiprows=2,unpack=True)
ra = data[0]
dec = data[1]
mag_u = data[2]
mag_g = data[3]
mag_r = data[4]
mag_i = data[5]

figure(figsize=(7,7))
for i in range(ra.size):
    x = ra2-ra[i]
    y = dec[i]
    plot([x],[y],"ob")
    text(x,y,"%d"%(int(mag_r[i])))
xlabel("ra (deg)")
ylabel("dec (deg)")
for p in positions:
    x = p[0]*width/shape[1]
    y = dec1+(shape[0]-p[1])*height/shape[0]
    plot([x],[y],'xr')
```



Pour comparaison, les étoiles détectées par le traitement de l'image ont été représentées sous forme de croix rouges.

La magnitude est une échelle logarithmique. Plus elle est basse, plus l'étoile est lumineuse. L'étoile rouge la plus lumineuse du champ a une magnitude de 10. On voit que certaines étoiles peu lumineuses de magnitude 17 ou 18 n'ont pas été détectées par l'analyse de l'image. On peut y remédier en augmentant le seuil de détection. Il faut par ailleurs remarquer que les magnitudes récupérées par la requête SQL sont déterminées sur des images filtrées (par exemple un filtre rouge spécifique). L'intensité de chaque tâche est déterminée par ajustement avec la fonction d'étalement du point (PSF : point spread function) et non pas par simple comptage des pixels.