


# Processamento de Imagens com Python

Rafael Güntzel Arenhart

# Links importantes

Download da distribuição Python Anaconda:

**https://www.continuum.io/downloads**

Documentação numpy e scipy:

**https://docs.scipy.org/doc/numpy/reference/routines.html**
**http://docs.scipy.org/doc/scipy/reference/**

Documentação PIL:

**http://effbot.org/imagingbook/image.htm**

Documentação Scikit-image:

**http://scikit-image.org/docs/dev/**

# Importar pacotes

```
from __future__ import division
import PIL.Image as pil
import numpy as np
import scipy.ndimage as sp
import matplotlib.pyplot as plt
import skimage as sk
import skimage.filters as filters
import skimage.morphology as morphology
import skimage.measure as measure
import Tkinter as tk
import tkFileDialog as filedialog
```

**Division** – Faz com que divisões não sejam arredondadas

**PIL** – Python Image Library, ferramentas para carregar e salvar imagens

**Numpy** – Suporte para matrizes, úteis para manipular imagens

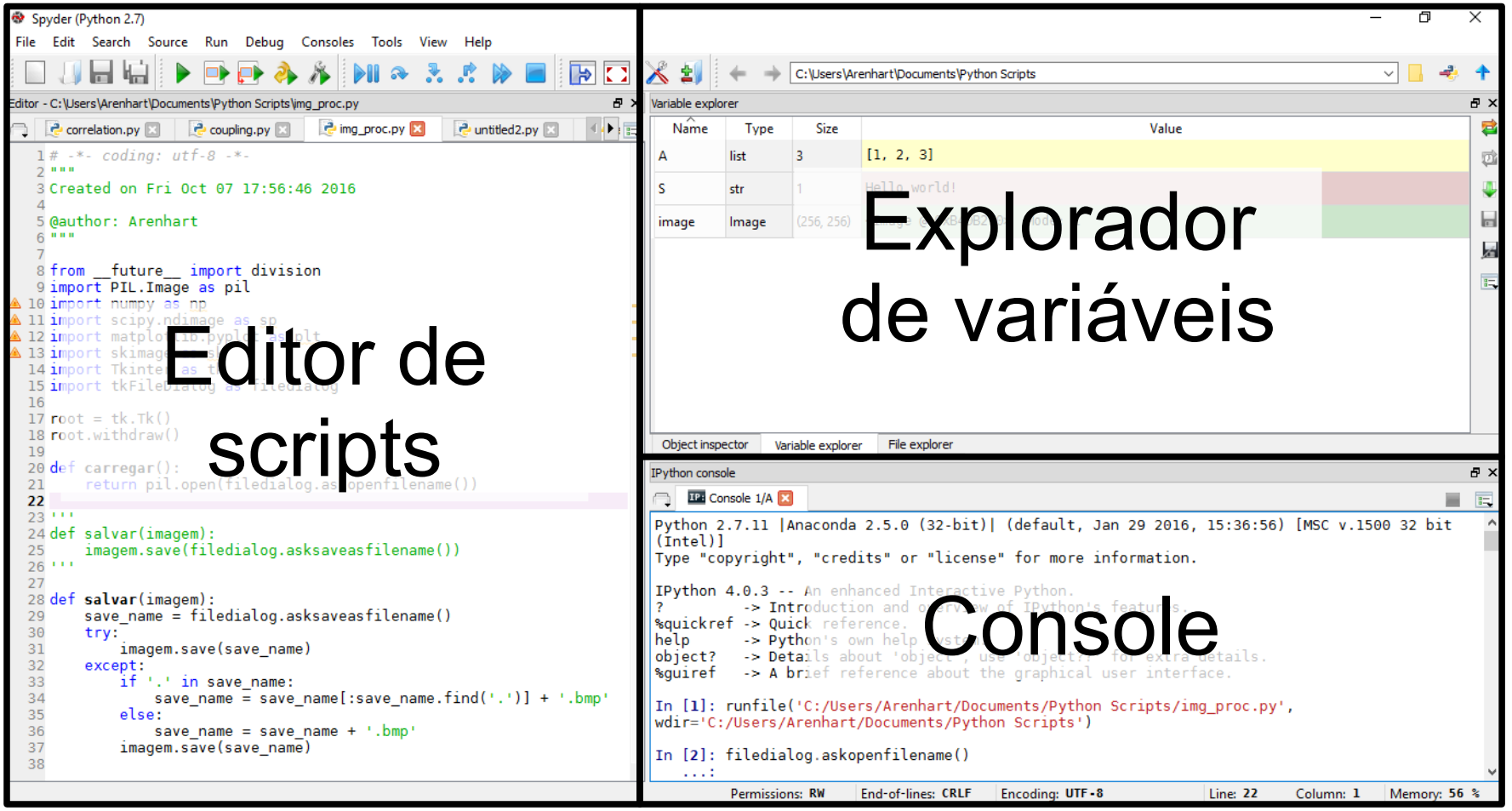
**Scipy.ndimage** – Operações mais simples de processamento de imagens

**Pyplot** – Ferramentas para exibir imagens e gráficos

**Skimage** – Operações avançadas de processamento de imagens

**Tkinter** e **filedialog** – Interface para carregar e salvar imagens

# Interface - Spyder



# Carregar imagens

```
def carregar():
    return pil.open(filedialog.askopenfilename())
```

```
In [9]: image = carregar()

In [10]: image
    ...:
Out[10]:
```

# Salvar imagem

```
    def salvar(imagem):
        imagem.save(filedialog.asksaveasfilename())
```

```
In [21]: salvar(image)
    ...:
Traceback (most recent call last):

  File "<ipython-input-21-765ce5110e9d>", line 1, in <module>
    salvar(image)

  File "C:/Users/Arenhart/Documents/Python Scripts/img_proc.py", line 24, in salvar
    imagem.save(filedialog.asksaveasfilename())

  File "C:\Anaconda2\lib\site-packages\PIL\Image.py", line 1662, in save
    format = EXTENSION[ext]

KeyError: ''
```

Gera erro se o arquivo salvo não tiver uma extensão de imagem válida

# Salvar imagem

```
def salvar(imagem):
    save_name = filedialog.asksaveasfilename()
    try:
        imagem.save(save_name)
    except:
        if '.' in save_name:
            save_name = save_name[:save_name.find('.')] + '.bmp'
        else:
            save_name = save_name + '.bmp'
        imagem.save(save_name)
```

Se a extensão for inválida, salva automaticamente como formato bitmap.

# Operações em imagem

1 – Redimensionar

2 – Cortar

3 – Converter em escala de cinza

4 – Separar canais de cor

5 – Converter imagem em matriz

# Redimensionar

Para redimensionar uma imagem, utilize:
**imagem = imagem.resize(( novo_tamanho_x, novo_tamanho_y ))**
Atenção para os duplos parênteses!

```
In [28]: image
    ...:
Out[28]:
```

```
In [31]: image.resize((300,300))
Out[31]:
```

```
In [29]: image.resize((100,100))
Out[29]:
```

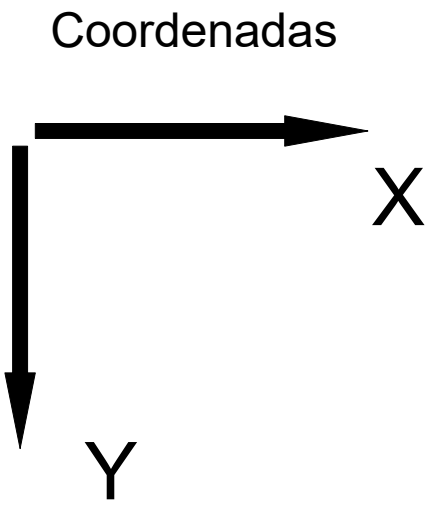
# Cortar imagem

Para selecionar apenas uma região da imagem, utiliza:
**imagem = imagem.crop((coordenada_x_esquerda,**
**coordenada_y_superior,**
**coordenada_x_direita,**
**coordenada_y_inferior))**

Atenção: os valores são para as coordenadas no eixa, e não para o tamanho da imagem cortada

```
In [28]: image
    ...:
Out[28]:
```

```
In [33]: image.crop((0,100,200,200))
Out[33]:
```

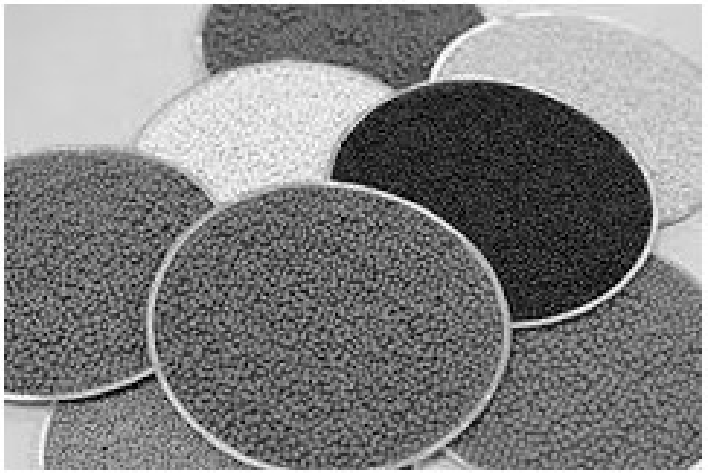


# Converter para escala de cinza

Para converter uma imagem colorida em escala de cinza, utilize:
**imagem = imagem.convert('L')**
L é o código para imagens em escala de cinza

```
In [35]: pigmentos
    ...:
Out[35]:
```

```
In [36]: pigmentos.convert('L')
Out[36]:
```

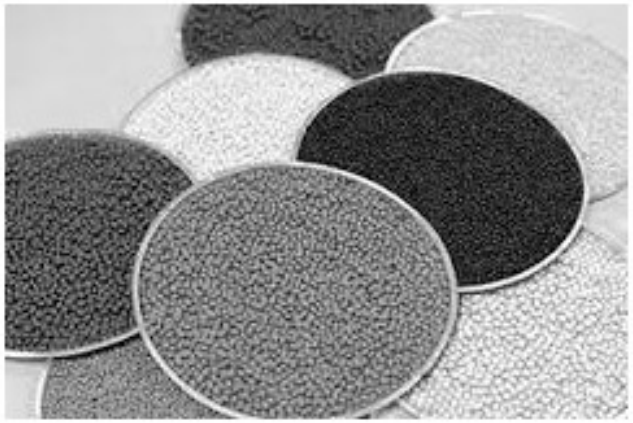


# Separar canais de cor

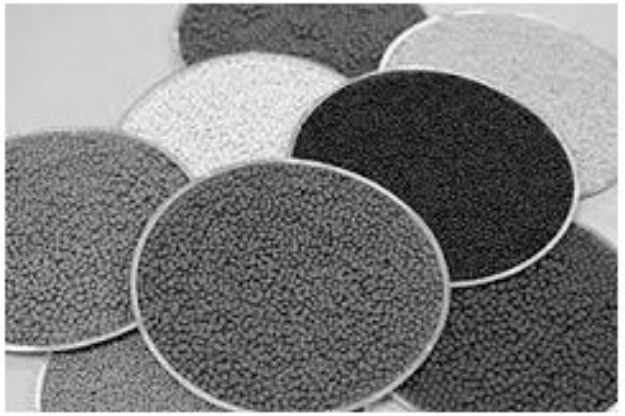
Para separar uma imagem colorida em sues canais de cor, utilize:
**imagem_vermelha, imagem_verde, imagem_azul = imagem.split()**

```
In [39]: pigmentos_red, pigmentos_green, pigmentos_blue = pigmentos.split()
```

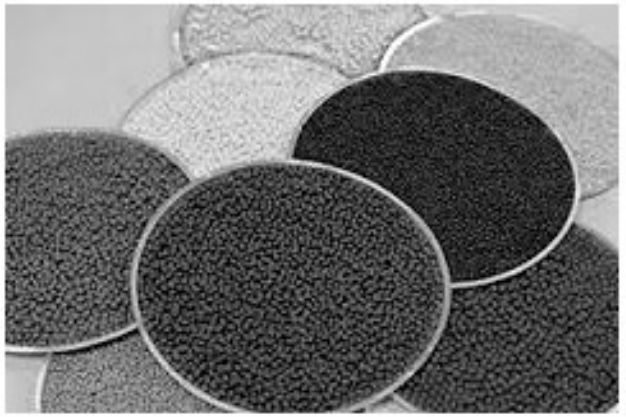
```
In [40]: pigmentos_red
Out[40]:
```



```
In [41]: pigmentos_green
Out[41]:
```



```
In [42]: pigmentos_blue
Out[42]:
```



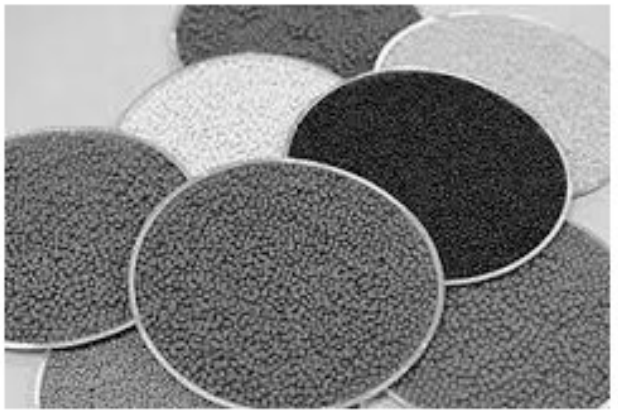
# Converter imagem em matriz

Para converter de imagem para matriz:
**imagem_mat = np.array(imagem)**

```
In [62]: pigmentos_mat = np.array(pigmentos_pb)
    ...: pigmentos_mat
Out[62]:
array([[201, 201, 201, ..., 211, 211, 211],
       [201, 201, 201, ..., 211, 211, 211],
       [202, 202, 202, ..., 211, 211, 211],
       ...,
       [214, 213, 214, ..., 128, 197, 248],
       [214, 213, 214, ..., 229, 238, 243],
       [214, 213, 214, ..., 224, 240, 202]], dtype=uint8)
```

Para converter de matriz para imagem:
**imagem = pil.fromarray(imagem_mat)**

```
In [63]: pigmentos_img = pil.fromarray(pigmentos_mat)
    ...: pigmentos_img
Out[63]:
```



# Diferenças matriz/imagem

A matriz não pode ser diretamente salva como imagem, mas uma forma simples de salvar uma matriz é utilizar:
**salvar(pil.fromarray(imagem))**

A matriz também não é representada como uma imagem no console, e sim como uma série de números, dificultando a sua visualização. Existem duas formas de visualizar uma matriz como imagem:
**1** – Selecionar a imagem no explorador de imagens com o botão direito e selecionar “show image”
**2** – Utilizar o comando pil.fromarray(imagem), sem atribuí-lo a uma variável, isso irá apenas mostrar a imagem no console.

# Visualizar matrizes

Variable explorer

| Name | Type | Size | Value |
|---|---|---|---|
| hist | tuple | 2 | (array([4..., 20432]), array([ ... 255. ])) |
| im | uint8 | (256, 256) | array([[0, 0, 0, ..., 0, 0, 0], |
| image | Image | (256, 256) | <Image @ 0xB40B270> Mo |
| pigmentos | Image | (240, 160) | <JpegImageFile @ 0xB40B8 |
| pigmentos_blue | Image | (240, 160) | <Image @ 0xB40BF90> Mo |
| pigmentos_green | Image | (240, 160) | <Image @ 0xB40BF70> Mo |

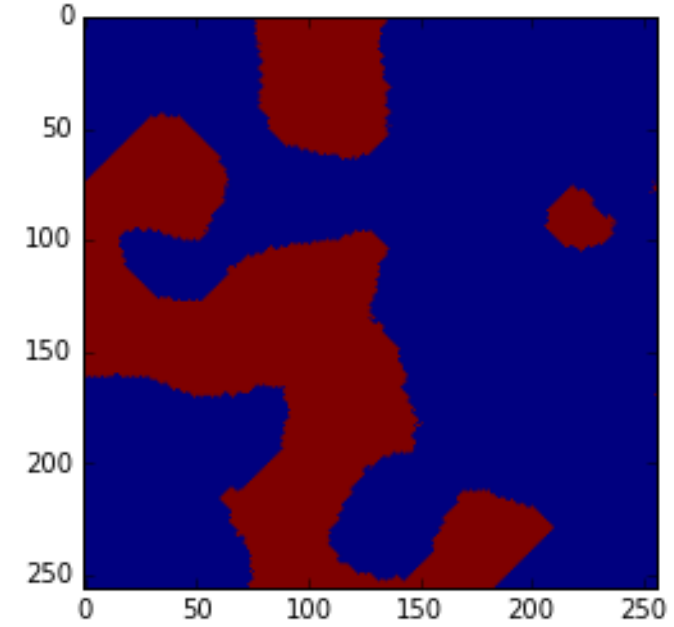
Edit
Plot
Show image
Save array
Insert
Remove
Copy
Paste

Object inspector | Variable explorer | File explorer

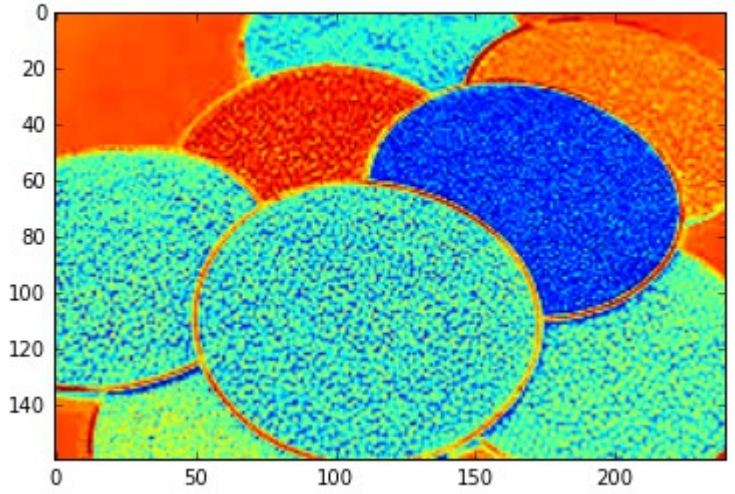
IPython console

Console 1/A

```
In [66]: %varexp --imshow im
```




```
In [67]: %varexp --imshow pigmentos_mat
```




Na imagem gerada, vermelho escuro representa os números de maior valor, enquanto o azul escuro representa o menor valor

# Operações em matriz

1 – Histograma
2 – Expansão de contraste
3 – Equalização do histograma
4 – Binarização
5 – Mapa de distância
6 – Inversão de imagem
7 – Filtro gaussiano
8 – Filtro mediano
9 – Filtros de gradiente
10 – Erosão e dilatação
11 – Abertura e fechamento
12 – Granulometria
13 – Rotulagem
14 – Conectividade
15 – Fatores de forma

# Histograma

Para gerar um histograma, utilize:
**histograma = np.histogram(imagem_mat.flatten(),bins=x)**
Onde “bins” se refere ao número de colunas de largura constante na qual os dados serão divididos. Atente-se à operação .flatten() que deve ser realizada na matriz.

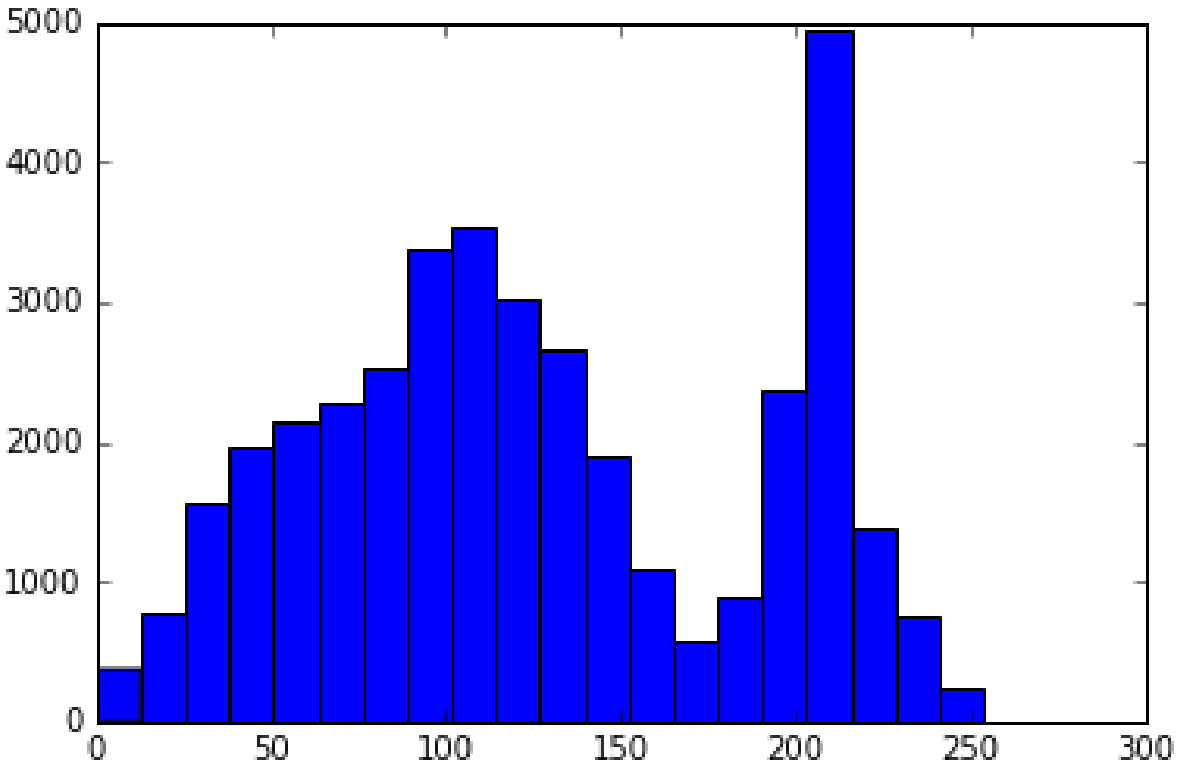
```
In [76]: histograma = np.histogram(pigmentos_mat.flatten(),bins=20)
    ...: histograma
Out[76]:
(array([ 376,  776, 1572, 1957, 2154, 2273, 2522, 3376, 3537, 3013, 2670,
        1905, 1099,  582,  902, 2359, 4942, 1388,  755,  242]),
 array([  0. ,  12.7,  25.4,  38.1,  50.8,  63.5,  76.2,  88.9,
        101.6, 114.3, 127. , 139.7, 152.4, 165.1, 177.8, 190.5,
        203.2, 215.9, 228.6, 241.3, 254. ]))
```

Atenção, a função np.histogram não gera um gráfico, e sim duas listas, uma com a contagem de pontos em cada coluna, e outra com os valores médio de cada coluna.

# Histograma – gerar gráfico

```
def histograma(matriz, bins = 20):
    return plt.hist(matriz.flatten(),bins=bins)
```

```
In [81]: plt.hist(pigmentos_mat.flatten(),bins=20)
Out[81]:
```

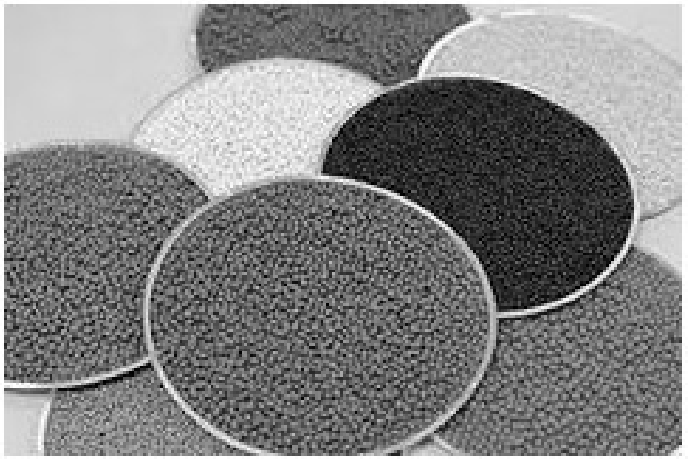



A imagem pode ser salva clicando sobre ela com o botão direito e selecionando “save image as...”.

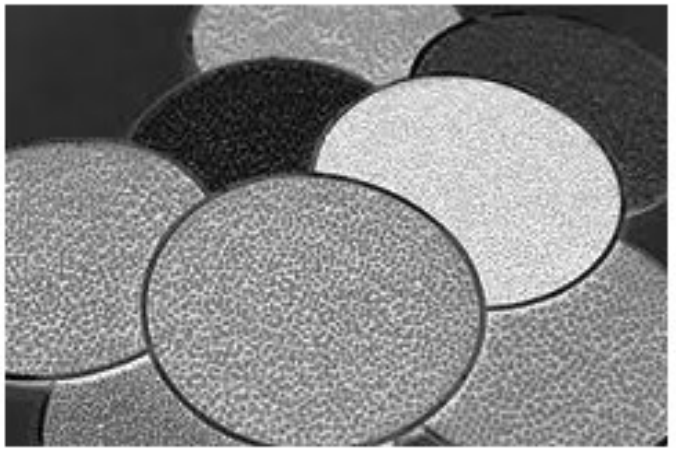
# Inverter imagem

```
def inverter(imagem):
    return 255 - imagem
```

```
In [149]: pil.fromarray(pigmentos_mat)
Out[149]:
```



```
In [148]: pil.fromarray(inverter(pigmentos_mat))
Out[148]:
```



# Expansão de contraste

```
In [170]: catedral_img
Out[170]:
```

Automaticamente modifica o nível de cinza para que eles englobem as cores de branco a preto. Utilize o comando: **sk.exposure.rescale_intensity(imagem_mat)**

```
In [173]: pil.fromarray(sk.exposure.rescale_intensity(catedral_mat))
Out[173]:
```

# Equalização de histograma

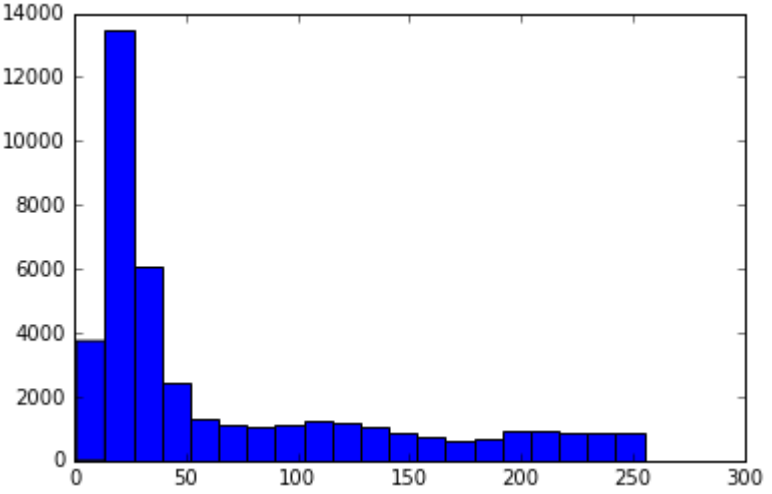
Para realçar uma imagem utilizando a equalização de histograma, pode-se utilizar a função a seguir:
**sk.exposure.equalize_hist(imagem_mat)**

Porém, a função acima gera uma matriz com valores entre 0 e 1, representando os valores máximos e mínimos da escala de cinza. Para se automatizar a conversão da matriz de volta para uma matriz de 256 níveis de cinza, pode-se definir a função abaixo:

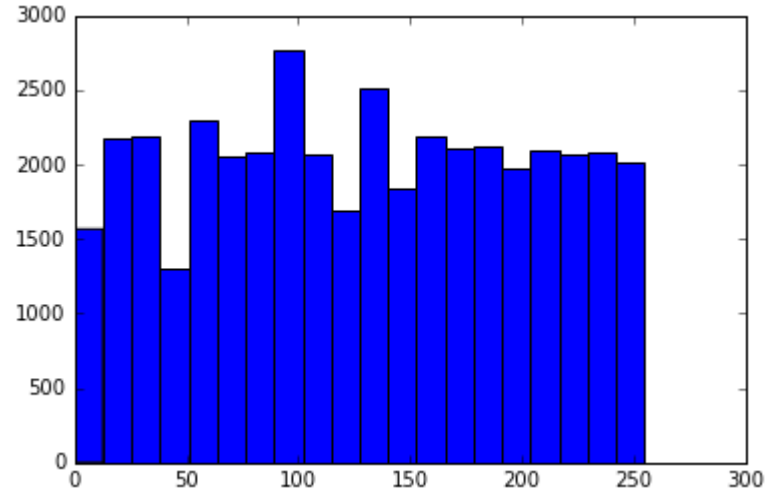
```
def equalizar_histograma(matriz):
    return (sk.exposure.equalize_hist(matriz)*255).astype('uint8')
```

# Equalização de histograma

```
In [191]: catedral_img
Out[191]:
```



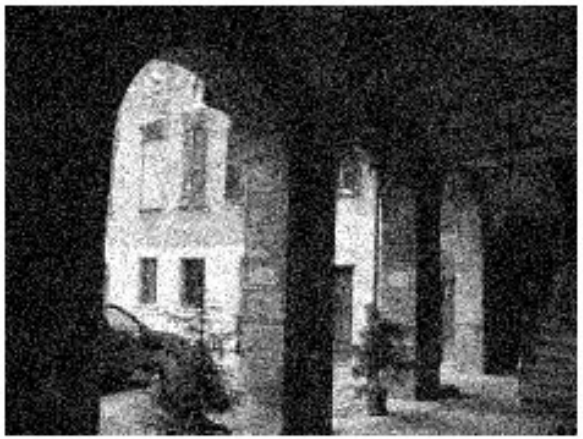

```
In [192]: catedral_img_eq
Out[192]:
```



# Filtro gaussiano

```
def filtro_gaussiano(matriz,sigma):
    return (filters.gaussian_filter(matriz,
                                    sigma=sigma)*255).astype('uint8')
```

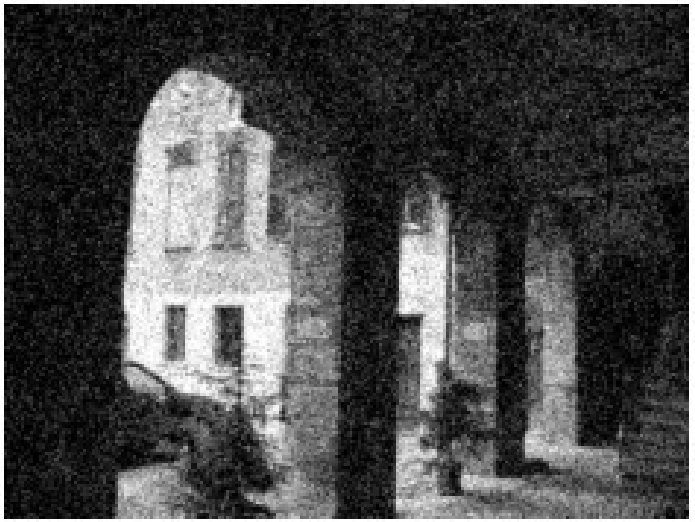
```
In [224]: catedral_ruido_img
Out[224]:
```



```
In [221]: pil.fromarray(filtro_gaussiano(catedral_ruido_mat,sigma=1))
Out[221]:
```

# Filtro gaussiano

**Sigma = 0.5**



**Sigma = 1**

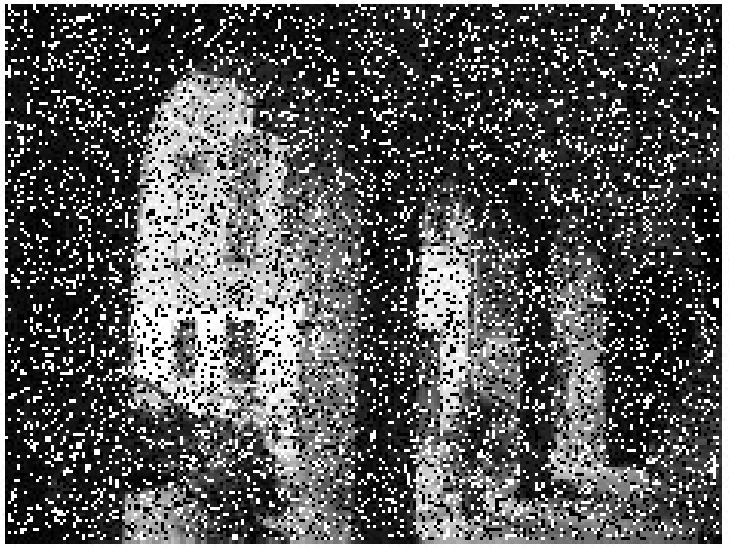
**Sigma = 2**

**Sigma = 3**

# Filtro mediana

```
def filtro_mediana(matriz,tamanho):
    return filters.median(matriz,morphology.disk(tamanho))
```

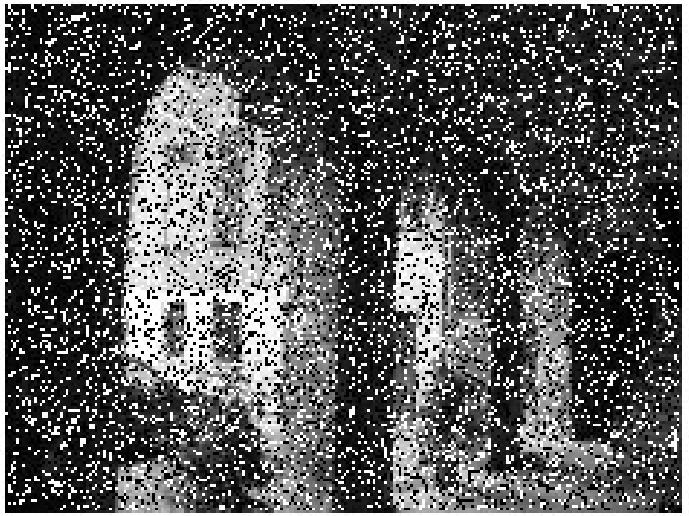
```
In [232]: catedral_ruidosp_img
Out[232]:
```



```
In [238]: pil.fromarray(filtro_mediana(
              catedral_ruidosp_mat,1))
Out[238]:
```

# Filtro mediana

**Original**



**Tamanho 1**

**Tamanho 2**

**Tamanho 3**

# Filtro de realce

```
def filtro_realce(matriz, tamanho):
    return filters.rank.enhance_contrast(matriz,morphology.disk(tamanho))
```

```
In [268]: pil.fromarray(catedral_desfocada_mat)
Out[268]:
```

```
In [270]: pil.fromarray(filtro_realce(catedral_desfocada_mat,1))
Out[270]:
```

# Binarização

```
def binarizar(matriz, limiar=None):
    if limiar == None:
        limiar = filters.threshold_otsu(matriz)
    return ((matriz <= limiar) *255).astype('uint8')
```

Duas formas de utilizar a função:
1 – **binarizar(imagem_mat)** – segmenta utilizando a binarização automática de Otsu
2 – **binariza(imagem_mat, limiar=x)** – segmenta utilizando x como o limiar
Para descobrir o valor numérico para o limiar de Otsu, utilizar:
**filters.threshold_otsu(imagem_mat)**

```
In [119]: pil.fromarray(binarizar(pigmentos_mat))
Out[119]:
```

```
In [120]: filters.threshold_otsu(pigmentos_mat)
Out[120]: 141
```

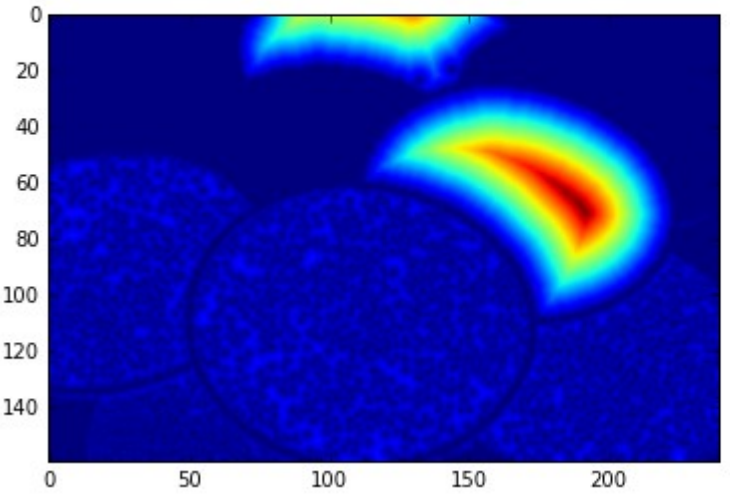
# Mapa de distância

```
def mapa_distancia(matriz_binarizada):
    sp.morphology.distance_transform_edt(matriz_binarizada)
```

```
In [145]: pil.fromarray(pigmentos_bin)
Out[145]:
```

```
In [142]: distance = sp.morphology.distance_transform_edt(pigmentos_bin)

In [143]: %varexp --imshow distance
```



# Filtros de gradiente

```
def filtro_prewitt(matriz):
    return (255-filters.prewitt(matriz)*255).astype('uint8')

def filtro_sobel(matriz):
    return (255-filters.sobel(matriz)*255).astype('uint8')

def filtro_scharr(matriz):
    return (255-filters.scharr(matriz)*255).astype('uint8')
```

Os filtros de gradiente utilizam métodos diferentes que possuem sensibilidades à orientação do gradiente diferentes.

A ordem de maior sensibilidade para menor é:
Prewitt → Sobel → Scharr

# Filtros de gradiente

Original

Prewitt

Sobel

Scharr

# Erosão e Dilatação

```
def erosao(matriz_binaria,tamanho=1):
    matriz_binaria = matriz_binaria//255
    return (morphology.binary_erosion(
            matriz_binaria,morphology.disk(tamanho))*255).astype('uint8')

def dilatacao(matriz_binaria,tamanho=1):
    matriz_binaria = matriz_binaria//255
    return (morphology.binary_dilation(
            matriz_binaria,morphology.disk(tamanho))*255).astype('uint8')
```

Erosão

Original

Dilatação

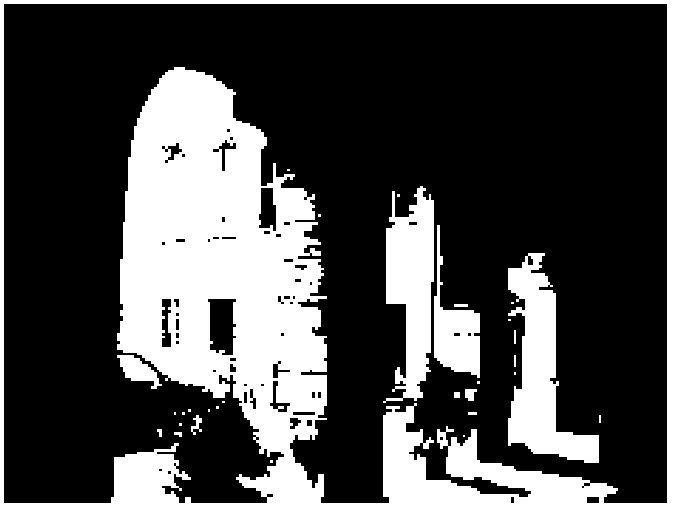
# Abertura e Fechamento

```
def abertura(matriz_binaria,tamanho=1):
    matriz_binaria = matriz_binaria//255
    return (morphology.binary_opening(
            matriz_binaria,morphology.disk(tamanho))*255).astype('uint8')

def fechamento(matriz_binaria,tamanho=1):
    matriz_binaria = matriz_binaria//255
    return (morphology.binary_closing(
            matriz_binaria,morphology.disk(tamanho))*255).astype('uint8')
```

Abertura

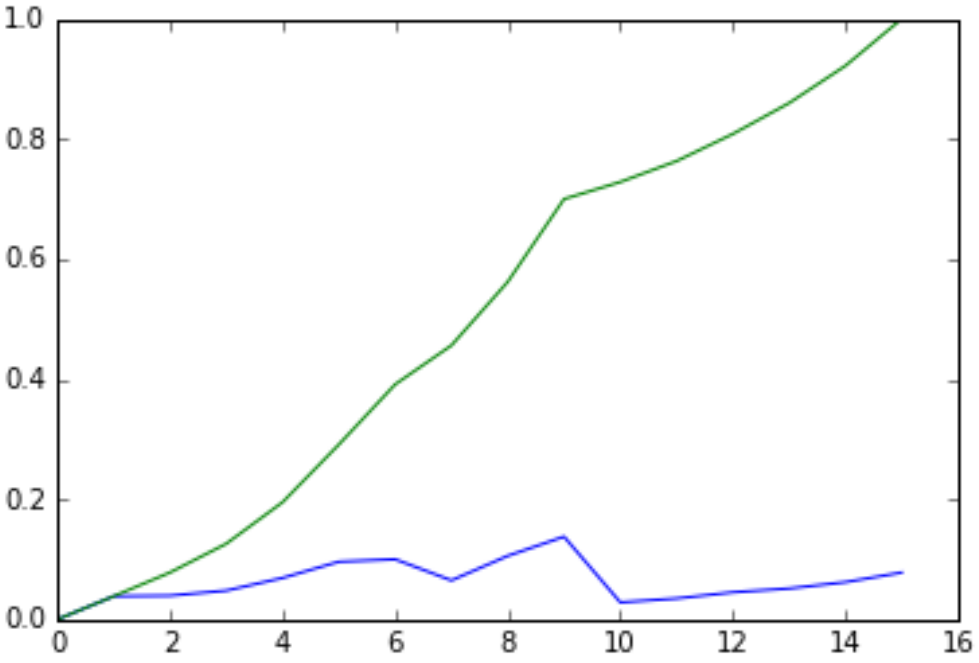
Original



Fechamento

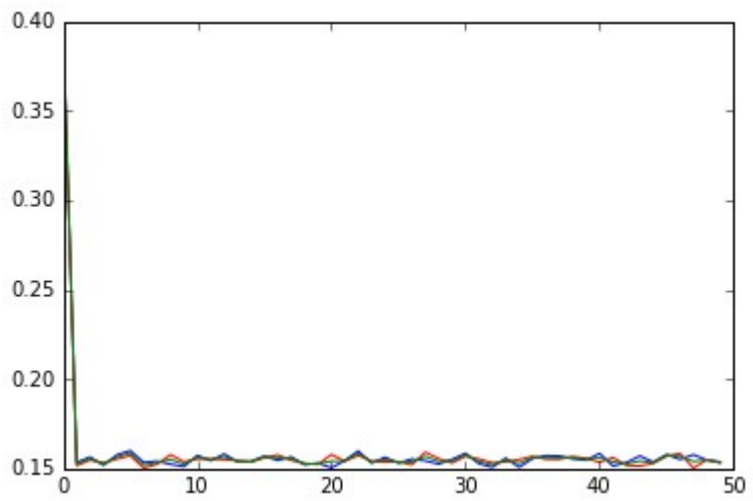
# Granulometria

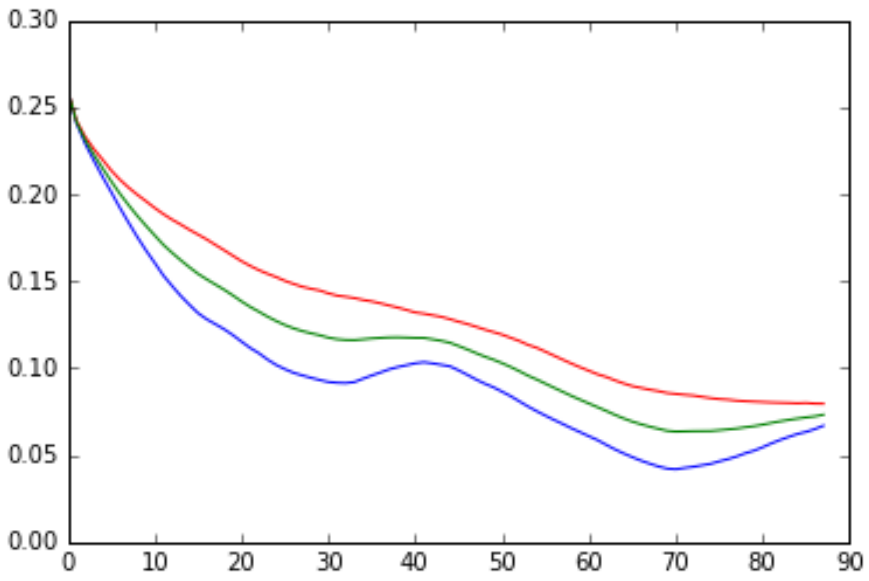
```
def granulometria(matriz_binaria):
    matriz_binaria = matriz_binaria
    area_inicial = sum(matriz_binaria.flatten())
    raio = [0]
    area_cf = [0]
    area = [0]
    i = 1
    while area_cf[-1] < 1:
        raio.append(i)
        new_area = 1 - (sum(abertura(matriz_binaria,i).flatten())/area_inicial)
        area.append(new_area-area_cf[-1])
        area_cf.append(new_area)
        i += 1
    plt.plot(raio,area,color='blue')
    plt.plot(raio,area_cf,color='green')
    plt.show()
    print raio,area
```



# Correlação

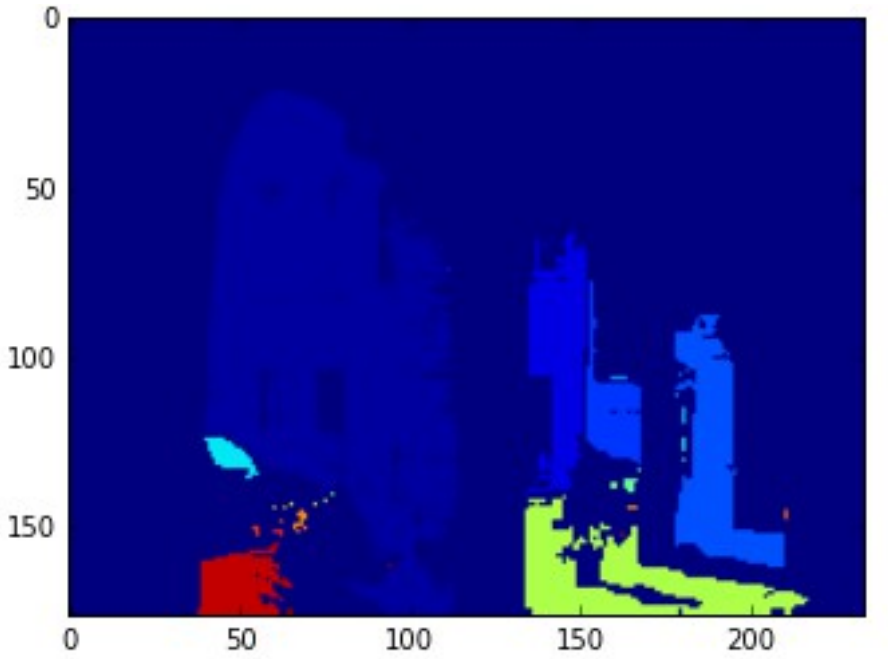
```
def correlacao(matriz_binaria):
    matriz_binaria = matriz_binaria // 255
    comprimento = range(min(matriz_binaria.shape)//2)
    tamanho_total = matriz_binaria.shape[0]*matriz_binaria.shape[1]
    correlacao_x = []
    correlacao_y = []
    matriz = matriz_binaria.flatten()
    for i in comprimento:
        correlacao_x.append(sum(matriz * np.append(matriz[i:],matriz[:i]))/tamanho_total)
    matriz = matriz_binaria.transpose().flatten()
    for i in comprimento:
        correlacao_y.append(sum(matriz * np.append(matriz[i:],matriz[:i]))/tamanho_total)
    correlacao = (np.array(correlacao_x) + np.array(correlacao_y))/2
    plt.plot(comprimento,correlacao_x,color='blue')
    plt.plot(comprimento,correlacao_y,color='red')
    plt.plot(comprimento,correlacao,color='green')
    plt.show()
```



# Rotulagem

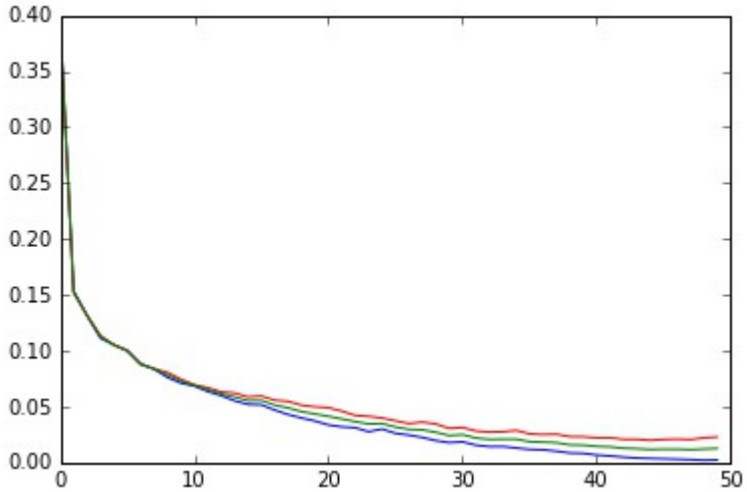
```
def rotular(imagem_binaria):
    return measure.label(imagem_binaria,background=0)
```

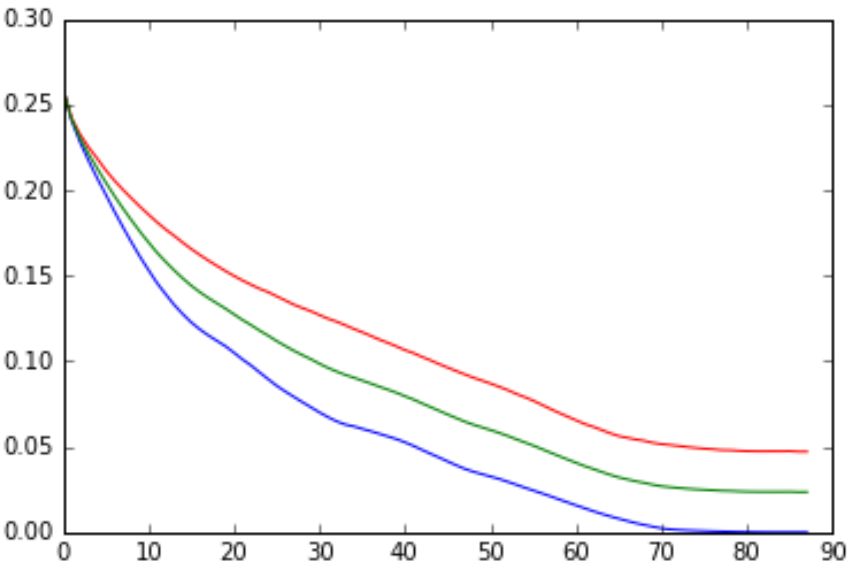



Para cada objeto conexo, é atribuído um rótulo, que é um valor numérico inteiro exclusivo para os pixeis daquele objeto.

# Conectividade

```
def conectividade(matriz_binaria):
    matriz_binaria = rotular(matriz_binaria)
    comprimento = range(min(matriz_binaria.shape)//2)
    tamanho_total = matriz_binaria.shape[0]*matriz_binaria.shape[1]
    conectividade_x = []
    conectividade_y = []
    matriz = matriz_binaria.flatten()
    for i in comprimento:
        matriz_deslocada = np.append(matriz[i:],matriz[:i])
        matriz_sobreposta = np.logical_and(matriz_deslocada==matriz,matriz != -1)
        conectividade_x.append(sum(matriz_sobreposta)/tamanho_total)
    matriz = matriz_binaria.transpose().flatten()
    for i in comprimento:
        matriz_deslocada = np.append(matriz[i:],matriz[:i])
        matriz_sobreposta = np.logical_and(matriz_deslocada==matriz,matriz != -1)
        conectividade_y.append(sum(matriz_sobreposta)/tamanho_total)
    conectividade = (np.array(conectividade_x) + np.array(conectividade_y))/2
    plt.plot(comprimento,conectividade_x,color='blue')
    plt.plot(comprimento,conectividade_y,color='red')
    plt.plot(comprimento,conectividade,color='green')
    plt.show()
```



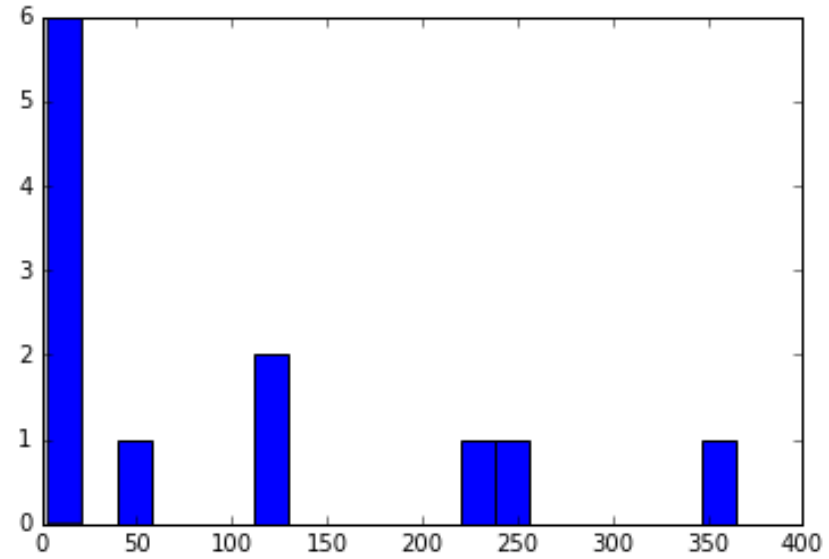



# Fatores de forma

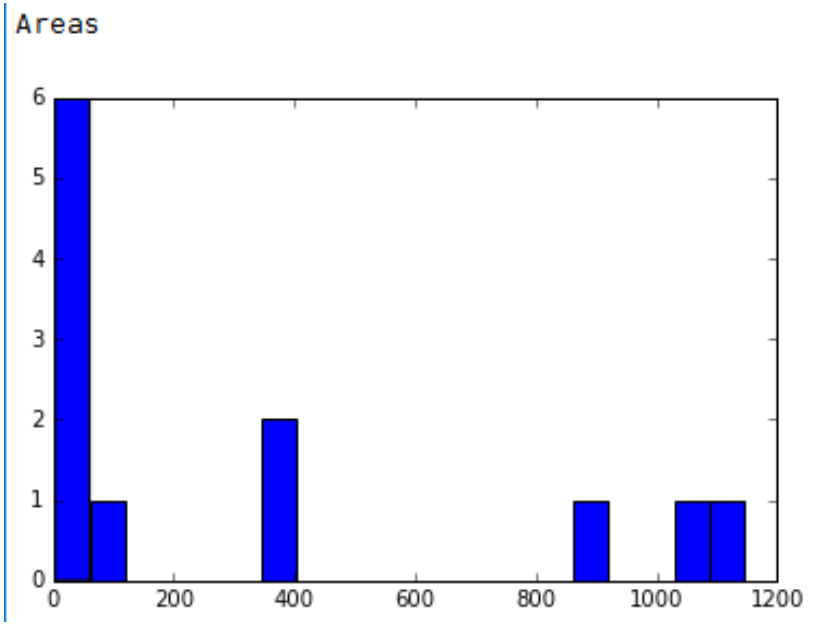
```
def propriedades(matriz_rotulada,bins=20):
    prop = measure.regionprops(matriz_rotulada)
    perimetros = []
    areas = []
    alongamento = []
    rugosidade = []
    for p in prop:
        if p['minor_axis_length'] == 0 : continue
        perimetros.append(p['perimeter'])
        areas.append(p['area'])
        rugosidade.append(p['perimeter']**2/(4*np.pi*p['area']))
        alongamento.append(p['major_axis_length']/p['minor_axis_length'])
    print 'Perimetros'
    plt.hist(perimetros,bins=bins)
    plt.show()
    print 'Areas'
    plt.hist(areas,bins=bins)
    plt.show()
    print 'Alongamento'
    plt.hist(alongamento,bins=bins)
    plt.show()
    print 'rugosidade'
    plt.hist(rugosidade,bins=bins)
    plt.show()
```

# Fatores de forma

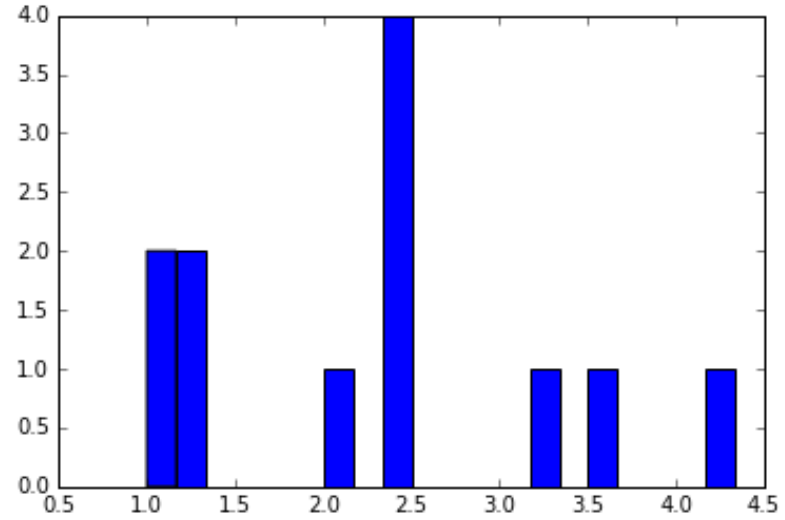
Perimetros



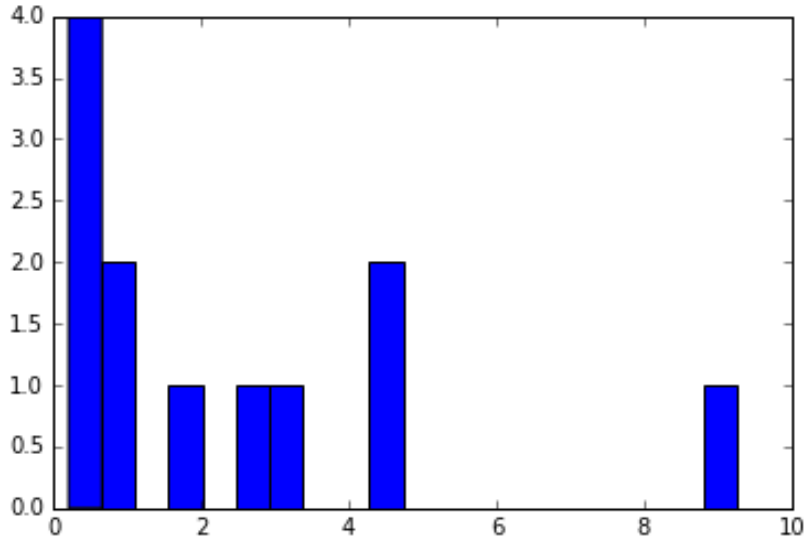

Areas




Alongamento




rugosidade



# Segmentação por Watersheding

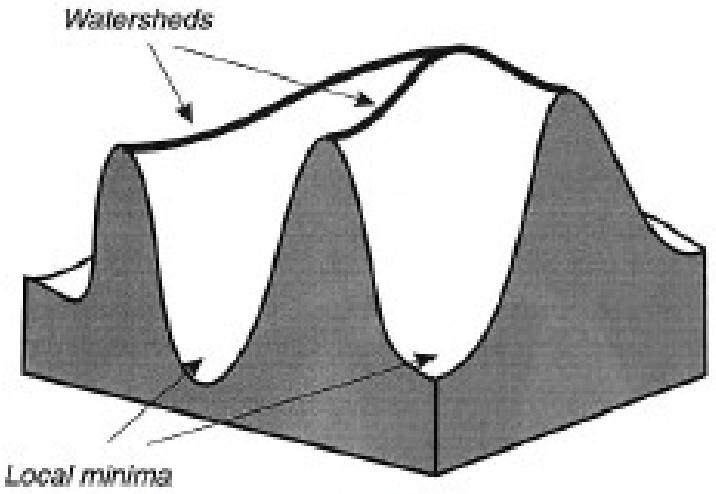



Fig. 2.22. The idea of watershed detection.

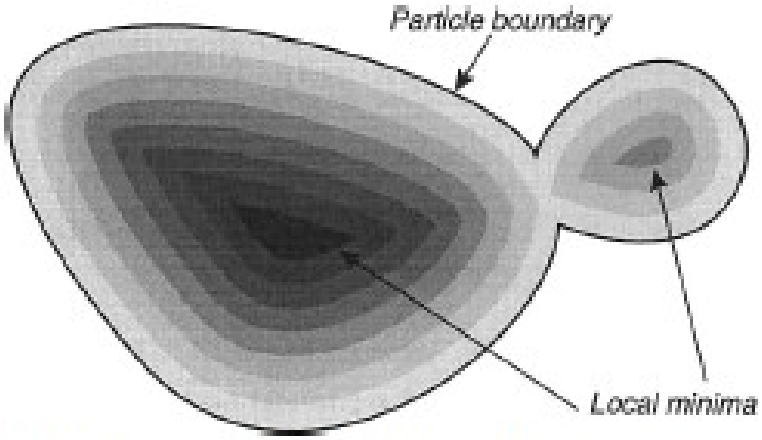
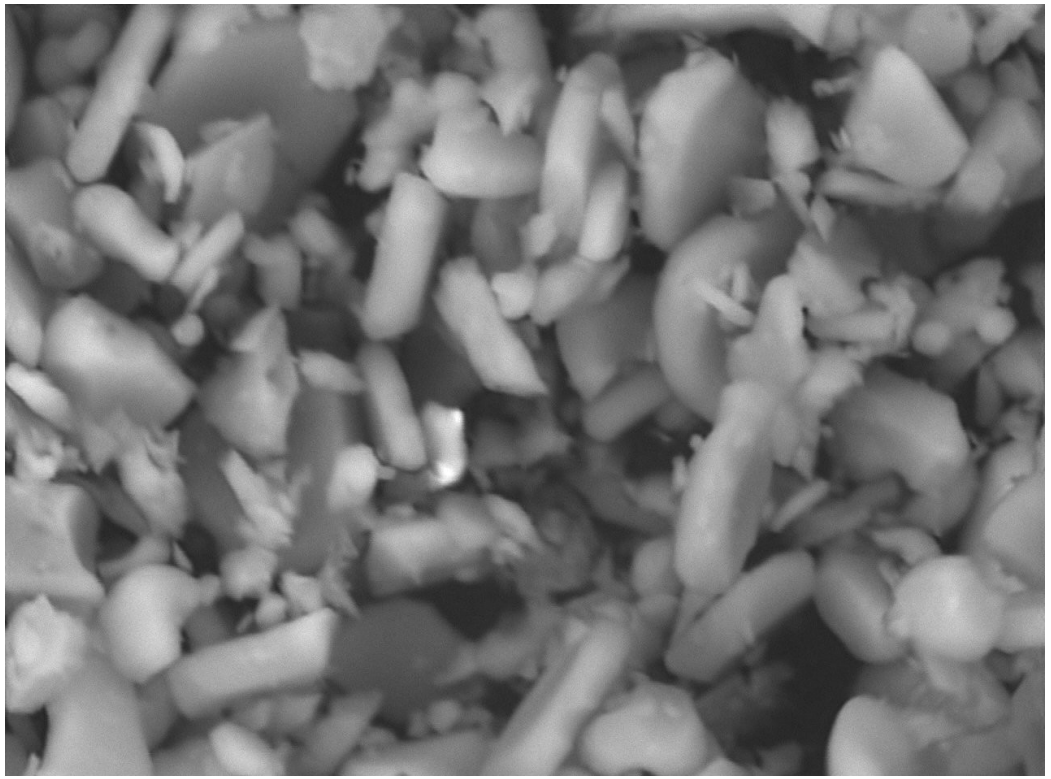
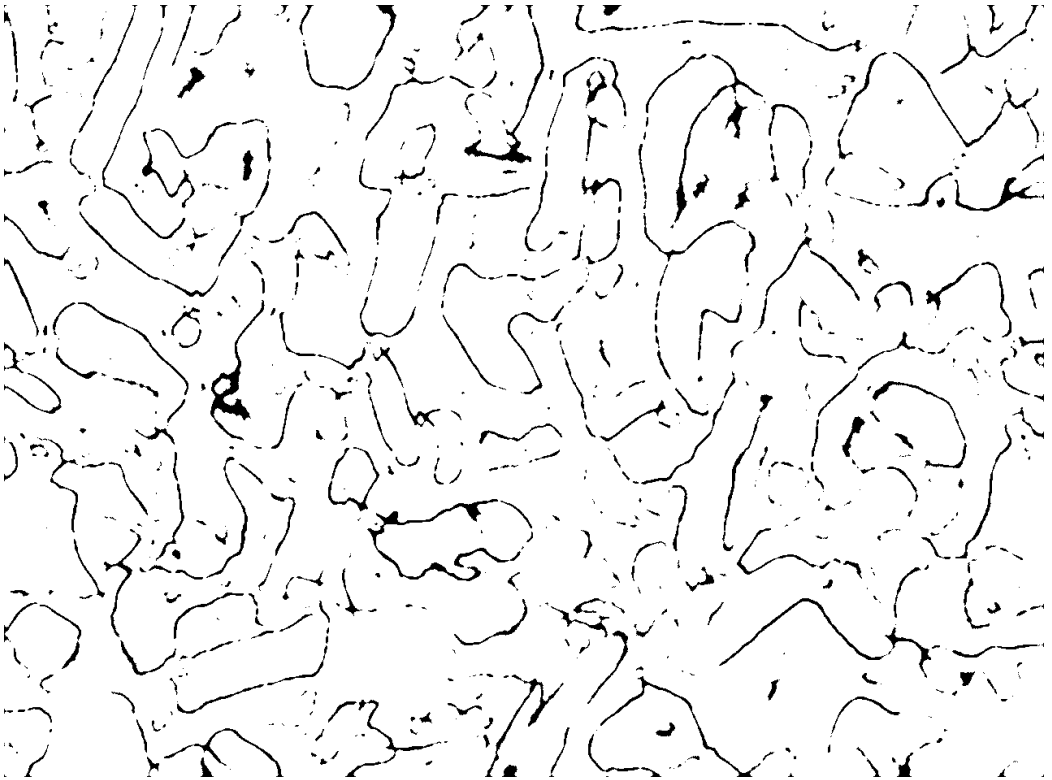



Fig. 2.23. Distance image built on the basis of binary image of a concave particle. More precisely, it is a negative of the distance image, as the particle center is its darkest point.

# Extração das bordas





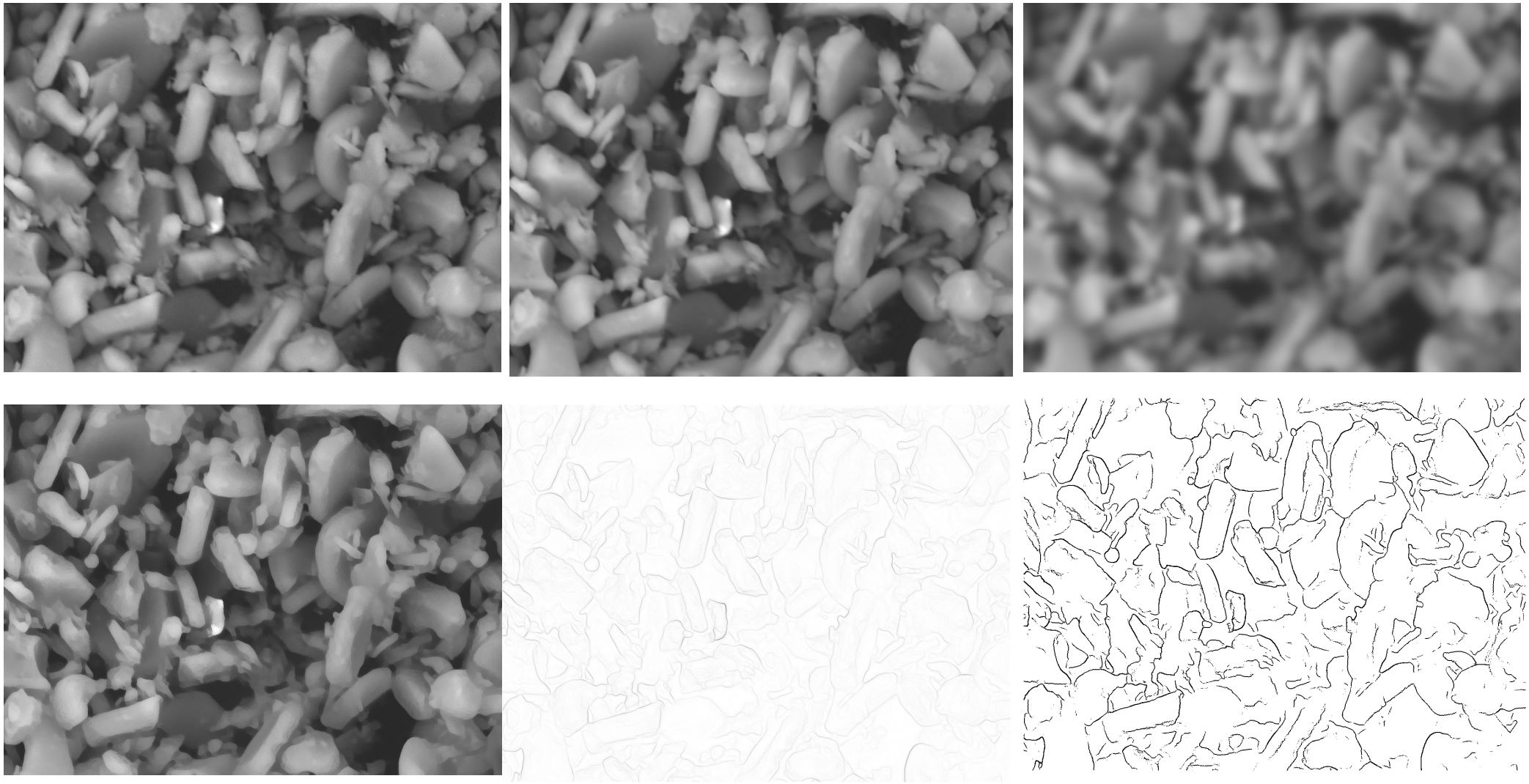
# Extração de bordas

```
def extrair_bordas(matriz, mediana = 1, gaussiano = 2, realce = 2,
                   limiar = None, bordas = None, mediana2 = 0, fechamento = 2,
                   janela = 100, offset = 0):

    bordas = filtro_mediana(matriz,mediana)
    bordas = filtro_gaussiano(bordas, gaussiano)
    bordas = filtro_realce(bordas,realce)
    bordas = filtro_scharr(bordas)
    bordas = (filters.threshold_adaptive(bordas,janela,
                                         offset=offset)*255).astype('uint8')
    fundo  = binarizar(matriz, limiar)
    bordas = bordas * (fundo//255)
    bordas = filtro_mediana(bordas,mediana2)

    return bordas
```

# Extração bordas



# Watersheding

```
def segregacao_watershed(bordas, pegada = 5, limiar = 0):


    dist = mapa_distancia(bordas)
    picos = feature.peak_local_max(dist,indices = False,
                                   labels = bordas,
                                   footprint = np.ones((pegada,pegada)),
                                    threshold_rel = limiar)
    marcadores = sp.label(picos)

    rotulos = morphology.watershed(-dist, marcadores[0], mask = bordas)


    return rotulos
```

# Watersheding