

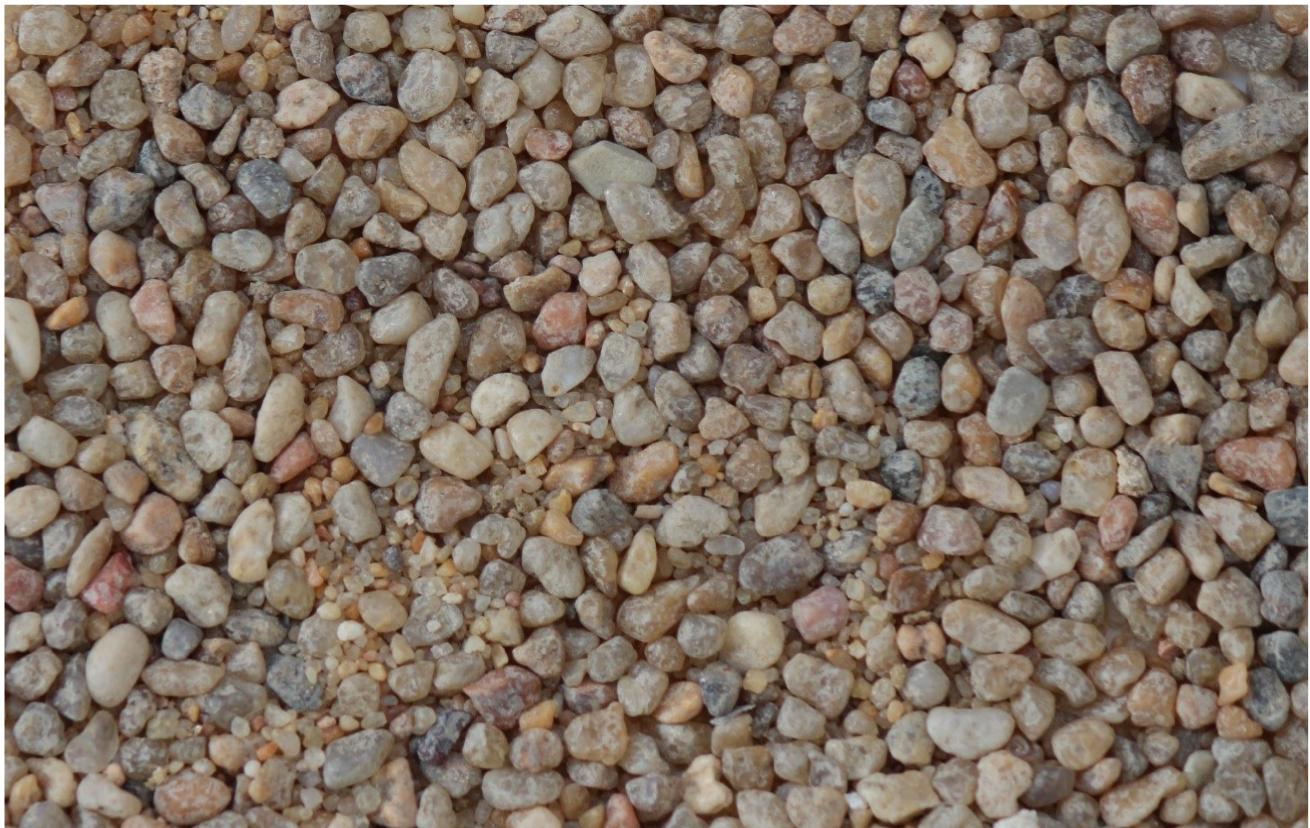
H4

1. Llegim la imatge i la passem a escala de grisos

En aquesta primera part del codi, es llegeix la imatge "pedretes.jpg" i es mostra a través de la funció "imshow". Després, es converteix a escala de grisos utilitzant la funció "rgb2gray", i el resultat s'emmagatzema en la variable "I_bw".

```
I = imread('pedretes.jpg');

% Imatge original
imshow(I)
```



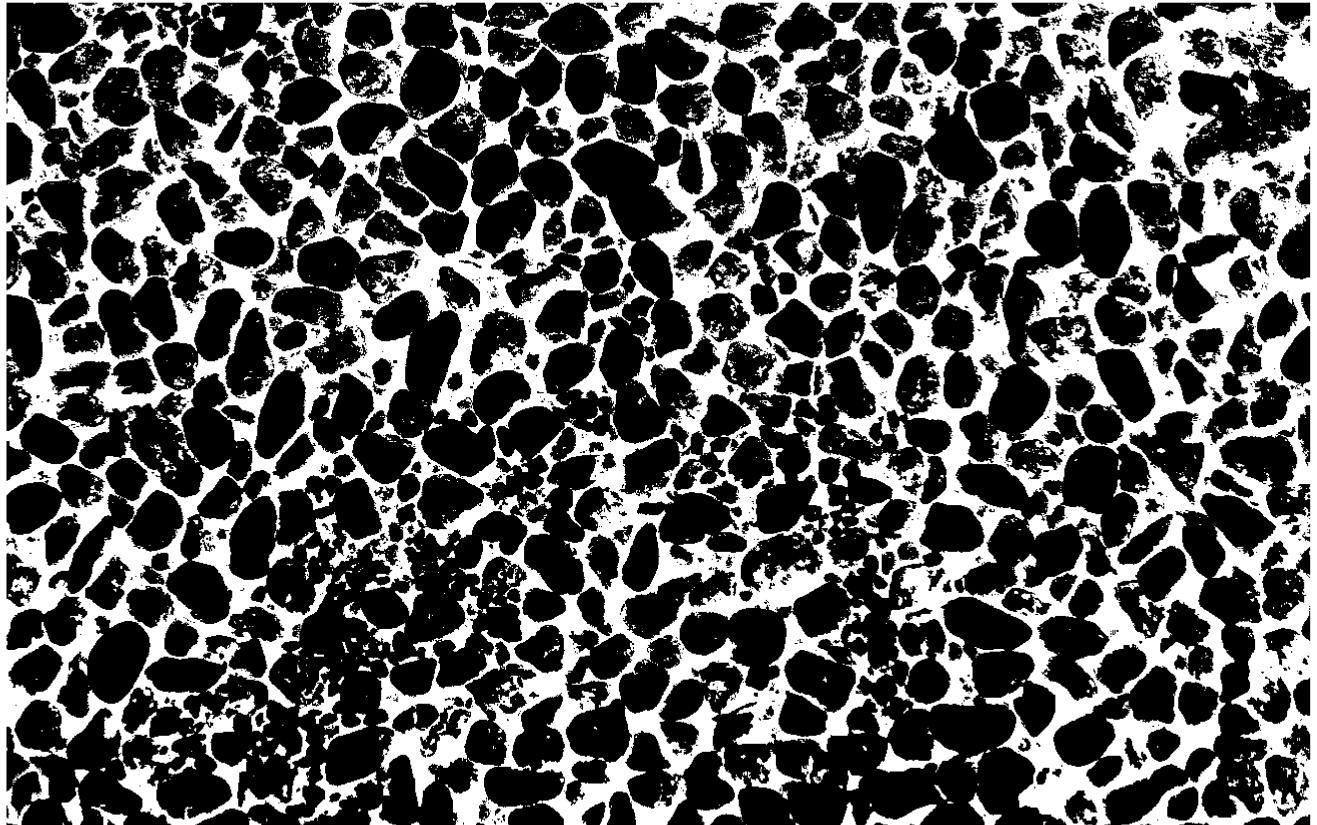
```
I_bw = rgb2gray(I);
```

2. Binaritzem la imatge

En aquesta segona part del codi, s'aplica un procés de binarització de l'imatge. Es realitza amb la funció "imbinarize", que converteix la imatge en blanc i negre, i es guarda el resultat en la variable "I_bin". Aquesta operació ajuda a detectar millor els contorns de les pedres.

```
% BINARITZAT
I_bin = not(im2bw(I , graythresh(I)));
```

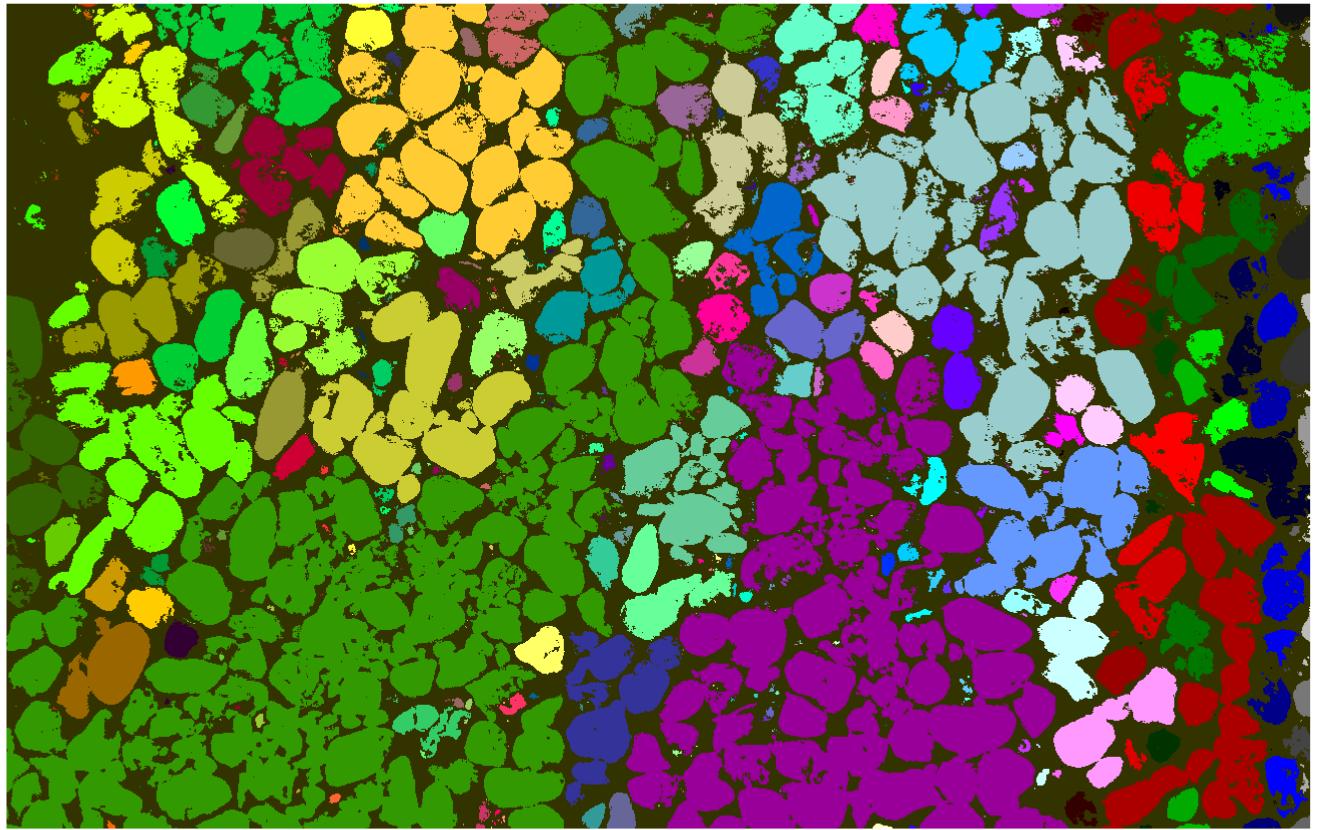
```
% BINARITZAT EXPERIMENTAL  
% I_bin = not(imbinarize(I_bw, "adaptive"));  
  
% Imatge binaritzada  
imshow(I_bin)
```



3. Etiquetem l'imatge per veure si es separen correctament les pedres

En aquesta tercera part del codi, es realitza un procés d'etiquetatge de les regions de l'imatge. Això ajuda a distingir les diferents pedres presents en la imatge. Aquesta operació es fa amb la funció "bwlabel", i es guarda el resultat en la variable "I_etiq". Després, es mostra la imatge etiquetada utilitzant la funció "imshow".

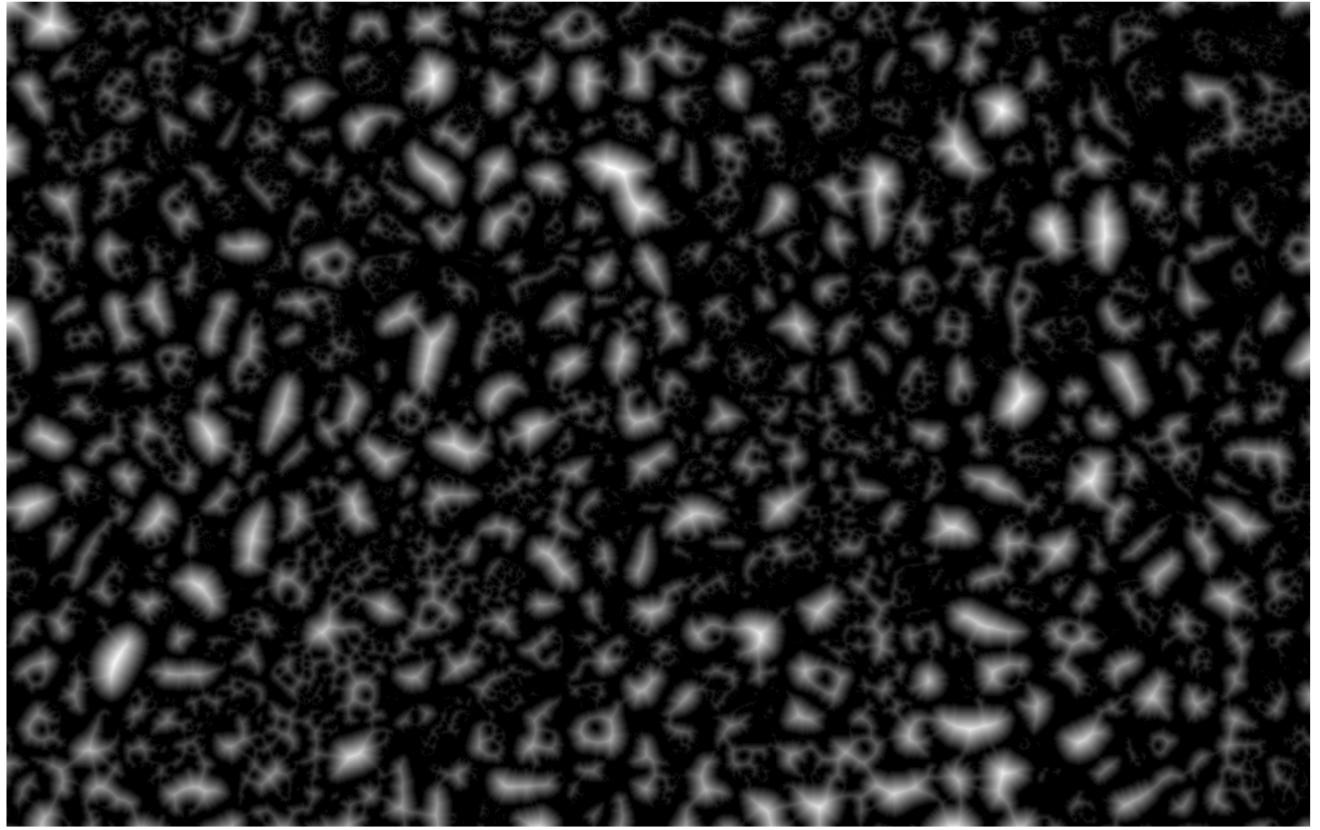
```
I_etiq = bwlabel(~I_bin);  
  
% Imatge etiquetada  
imshow(I_etiq, []), colormap('colorcube');
```



4. Calculem la transformada de distància

En aquesta quarta part del codi, es calcula la transformada de distància de l'imatge binaritzada. Aquesta operació ajuda a detectar millor els contorns de les pedres. Es fa amb la funció "bwdist", i es guarda el resultat en la variable "I_td". Després, es mostra la imatge transformada de distància utilitzant la funció "imshow".

```
I_td = bwdist(I_bin);  
  
% Transformada de distancia  
imshow(I_td, []);
```

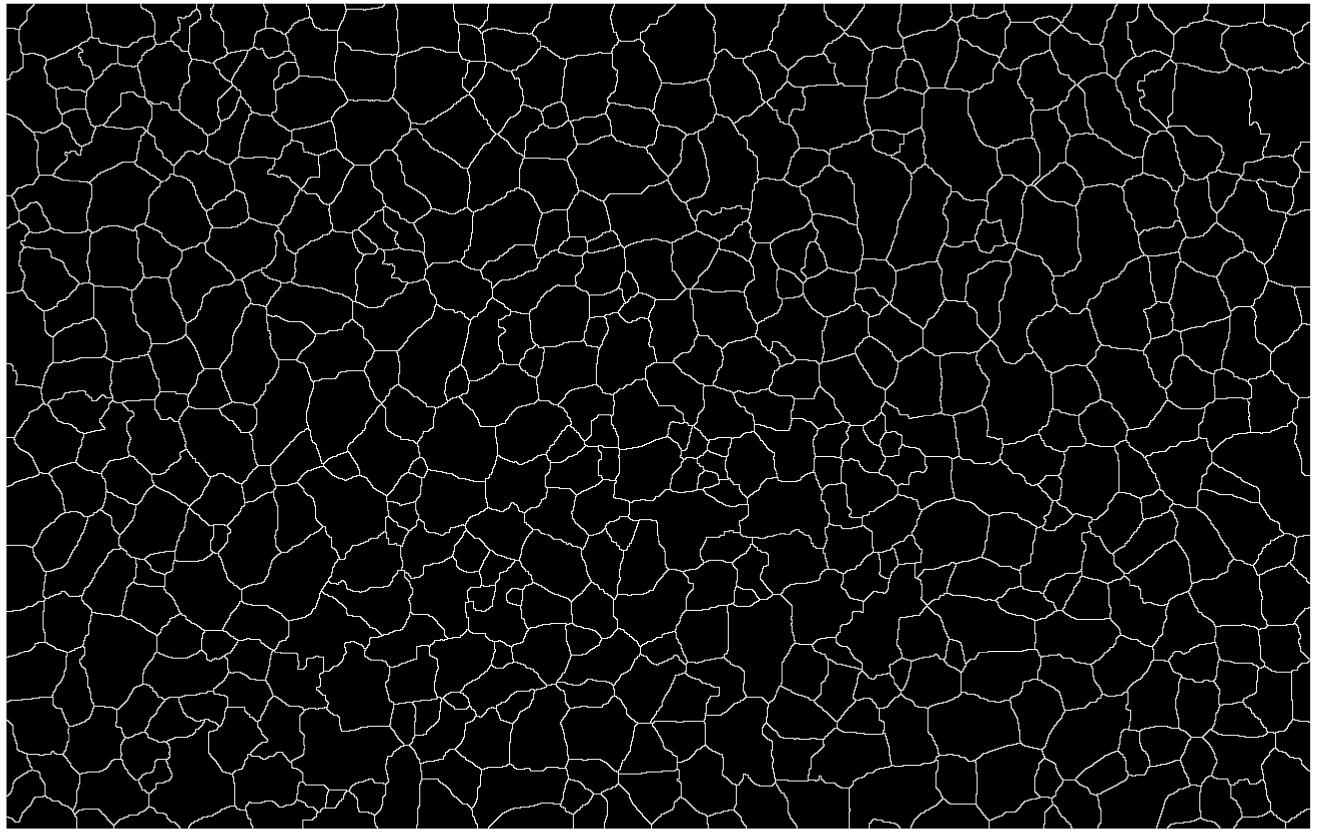


5. Calculem watershed

En aquesta cinquena part del codi, es realitza un procés de segmentació d'imatge. Això ajuda a separar les diferents pedres que hi ha a la imatge. Aquesta operació es realitza amb la funció "imhmax" i la funció "watershed". El resultat es guarda en la variable "I2_seg". Després, es mostra la imatge amb els blobs separats utilitzant la funció "imshow".

```
I2 = imhmax(I_td,4);
I2_seg = watershed(-I2);

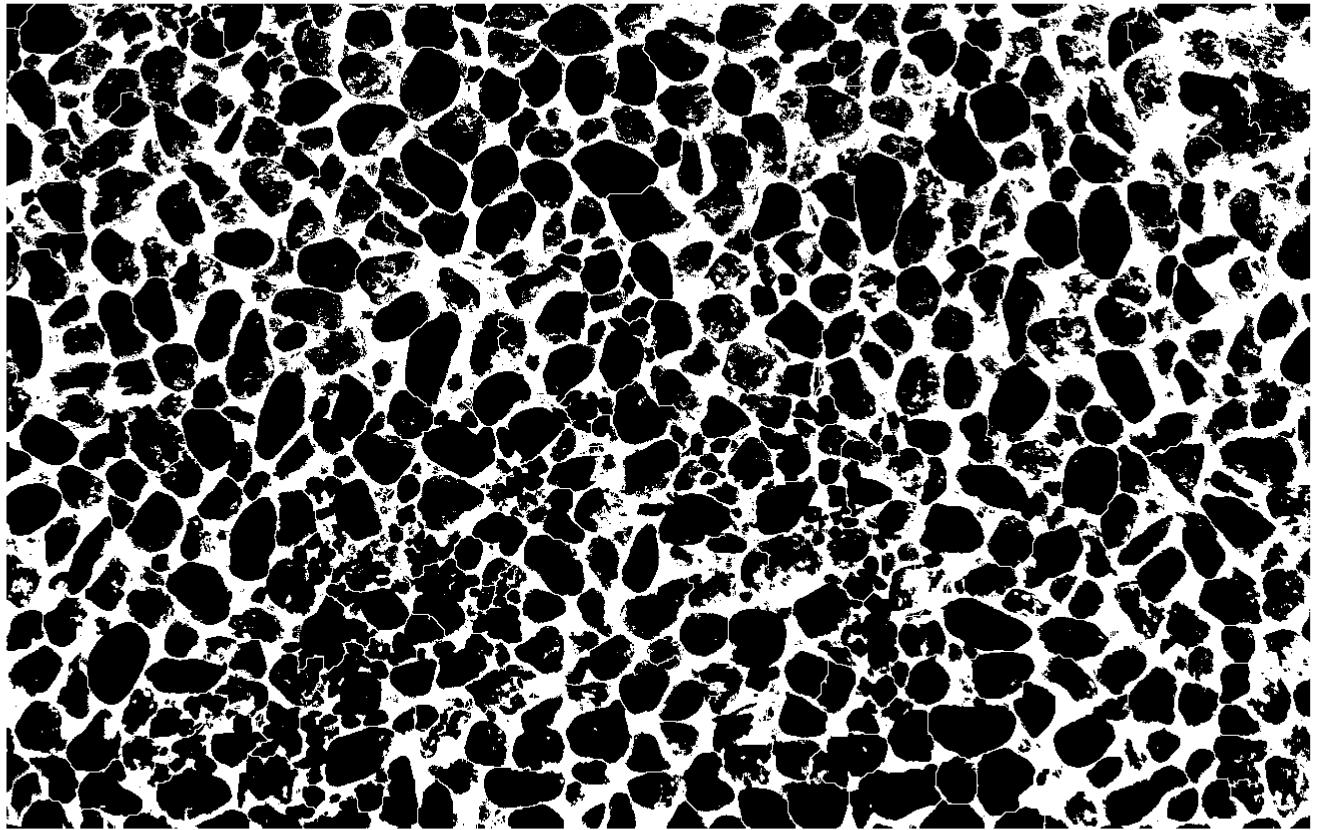
% Separaciode blobs
imshow(I2_seg == 0, []);
```



6. Imatge binaritzada amb watershed

En la sisena part del codi, s'aplica l'operació OR lògic entre la imatge binaritzada (`I_bin`) i la imatge segmentada per watershed (`I2_seg == 0`), això ens proporciona una imatge que té totes les pedres segmentades per watershed i també totes les pedres que ja havíem trobat a la imatge binaritzada.

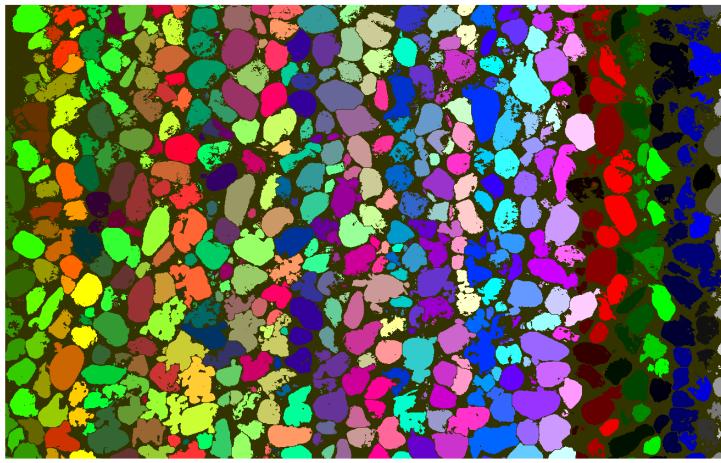
```
I_res = I_bin | (I2_seg == 0);  
  
% Separacio  
imshow(I_res)
```



7. Imatge etiquetada 2

A partir de la setena part del codi, es procedeix a etiquetar la imatge resultant de l'operació OR amb la funció bwlabel, que assigna un valor numèric diferent a cada regió o objecte de la imatge.

```
I_etiq_2 = bwlabel(~I_res);  
  
% Imatge etiquetada  
imshow(I_etiq_2, []), colormap('colorcube');
```



8. Conteig de pedres i dibuix de caixa contenedora

A la vuitena part del codi, s'obtenen les propietats de cada regió utilitzant la funció regionprops. Aquestes propietats inclouen l'àrea de cada regió i la caixa contenedora (bounding box) de cada regió.

Després, s'estableixen els criteris de tamany mínim i màxim de les pedres (size_min i size_max) que volem comptar a l'imatge. A continuació, es recorre cada regió i es comprova si la seva àrea està dins del rang establert per size_min i size_max. Si la regió compleix aquesta condició, s'incrementa el contador num_piedras, que conté el número total de pedres de la imatge.

Per a cada regió que compleixi els criteris de tamany, s'obté la caixa contenedora i s'afegeix a la imatge original (I_output) utilitzant la funció insertShape.

```
% Obtenim les propietats de cada regió
L = I_etiq_2;
props = regionprops(L, 'Area', 'BoundingBox');

% Imatge sobre la que contrastar la sortida
I_output = I;

% Definim el tamany minim y maxim de les pedres
size_min = 100;
size_max = 3000;
```

```
% Contem les pedres que compleixen els criteris de tamany, i dibuixem la
% caixa contenedora d'aquelles que compleixin els criteris
num_piedras = 0;
for i = 1:length(props)
    if props(i).Area >= size_min && props(i).Area <= size_max
        % Aquesta regió correspon a una pedra
        num_piedras = num_piedras + 1;

        % EXPERIMENTAL
        % Dibuixem la caixa contenedora al voltant de la pedra
        %bbox = props(i).BoundingBox;
        %x = bbox(1);
        %y = bbox(2);
        %w = bbox(3);
        %h = bbox(4);
        %I_output = insertShape(I_output, 'Rectangle', [x y w h], 'LineWidth', 3);
    end
end
I_output = imoverlay(I_output, I2_seg == 0);
```

9. Output

Finalment, s'afegeix el número total de pedres a la imatge original utilitzant la funció `insertText` i es mostra la imatge amb totes les pedres i els seus contorns.

```
% Mostrem el numero de pedres de l'imatge
I_output = insertText(I_output, [10 10], num2str(num_piedras), 'FontSize', 30);
imshow(I_output);
```

492



Com es pot observar, es contén correctament totes les pedretes amb un tamany d'entre 100 i 3000 pixels