

DATA_PROCESSING

June 6, 2023

This notebook contains an example script used to process the data for the article "Unsupervised Analysis of Optical Imaging Data for the Discovery of Reactivity Patterns in Metal Alloy" by R. Li, T. Galochkina, J.F. Lemineur, F. Kanoufi, and V. Shkirskiy (Small Methods journal). We recommend using the Jupyter Lab environment with Python 3 kernel to work with the script.

If you have any questions, please feel free to contact V. Shkirskiy at viacheslav.shkirskiy@cnr.fr.

1 Loading libraries and check versions

```
[1]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import cv2 as cv
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import sklearn
import math

# List the modules
modules = [np, matplotlib, cv, sklearn]

for module in modules:
    print(f"{module.__name__}: {module.__version__}")
```

```
numpy: 1.24.2
matplotlib: 3.7.1
cv2: 4.7.0
sklearn: 1.2.2
```

2 Find coordinates of all Si particles

2.1 load data

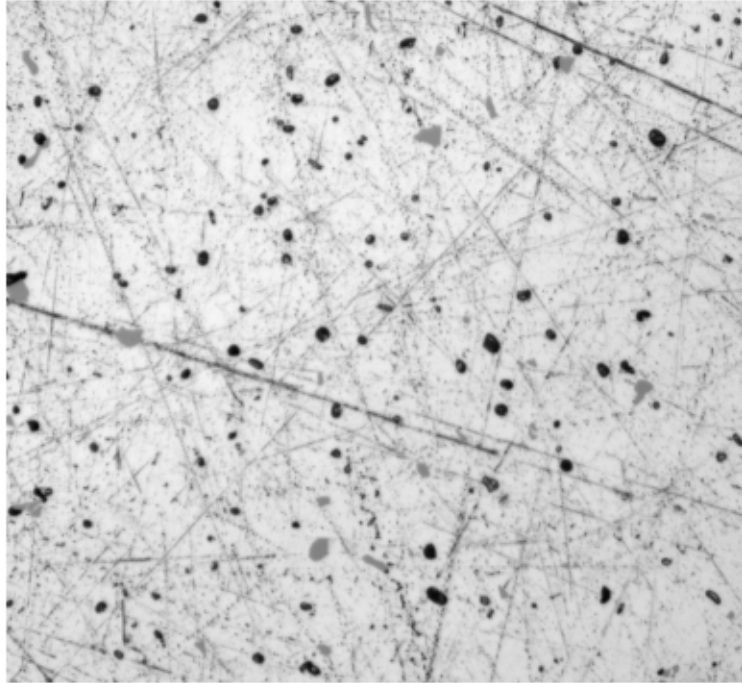
```
[2]: image = np.load('raw_optical_image.npy')

fig = plt.figure(figsize=(5,5))
plt.imshow(image, cmap="gray")
```

```
plt.axis("off");

print('The black contrasted area represents Si-rich particles, while the grey_
↳ contrasted area represents Fe-rich particles.'
      ' The size of each pixel is 62 nm across all the data.')
```

The black contrasted area represents Si-rich particles, while the grey contrasted area represents Fe-rich particles. The size of each pixel is 62 nm across all the data.



2.2 Image segmentation using multiple K-means algorithms

```
[3]: image8bit = (image // 256).astype(np.uint8) # convert to 8 bit
     indy,indx = image.shape
```

2.2.1 Step 1: separate background from particles

```
[4]: im_change_cluster = image8bit.reshape((-1,1))

optimal_model = KMeans(n_clusters=2,random_state=0, n_init=10).
↳ fit(im_change_cluster) # fitting model
clusters_list_pre = optimal_model.labels_ # get label of each pixel (return a_
↳ list)
```

```

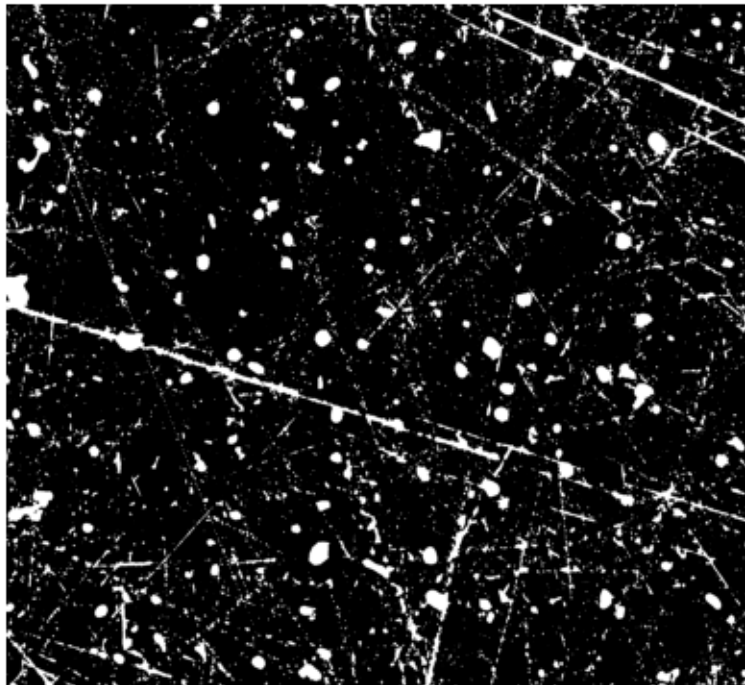
clust_matrix_refl = clusters_list_pre.reshape(indy,indx)

fig = plt.figure(figsize=(5,5))
plt.imshow(clust_matrix_refl, vmin=0, vmax=1, cmap="gray")
plt.axis("off");

print('The image displays particle areas, as well as other inhomogeneities such_
      as scratches, '
      'in white, while the Al matrix is represented in black.')

```

The image displays particle areas, as well as other inhomogeneities such as scratches, in white, while the Al matrix is represented in black.



2.2.2 Step 2: Cluster only the areas that correspond to particle areas

```

[5]: new_cluster = []

# get 1D vector of all values

for i in range(len(clusters_list_pre)):

    if clusters_list_pre[i] == 1:
        new_cluster.append(im_change_cluster[i])

```

```

# cluster particles into 2 categories
# we know that we have two kind of particles from SEM-EDX analysis: Si rich and
↳ Fe rich

optimal_model = KMeans(n_clusters=2, random_state=0, n_init=10).fit(new_cluster)
↳ # fitting model
clusters_list_pre_new = optimal_model.labels_ # get label of each pixel (return
↳ a list)

```

[6]: # transform clustered data back to matrix to represent it in the form of map

```

new_cluster = []

j=0
for i in range(len(clusters_list_pre)):

    if clusters_list_pre[i] == 0:
        new_cluster.append(0)
        continue

    if clusters_list_pre[i] == 1:

        if clusters_list_pre_new[j] == 0:
            new_cluster.append(1)
            j=j+1
            continue

        if clusters_list_pre_new[j] == 1:
            new_cluster.append(2)
            j=j+1
            continue

arr = np.array(new_cluster)
clust_matrix_refl = arr.reshape(indy,indx)

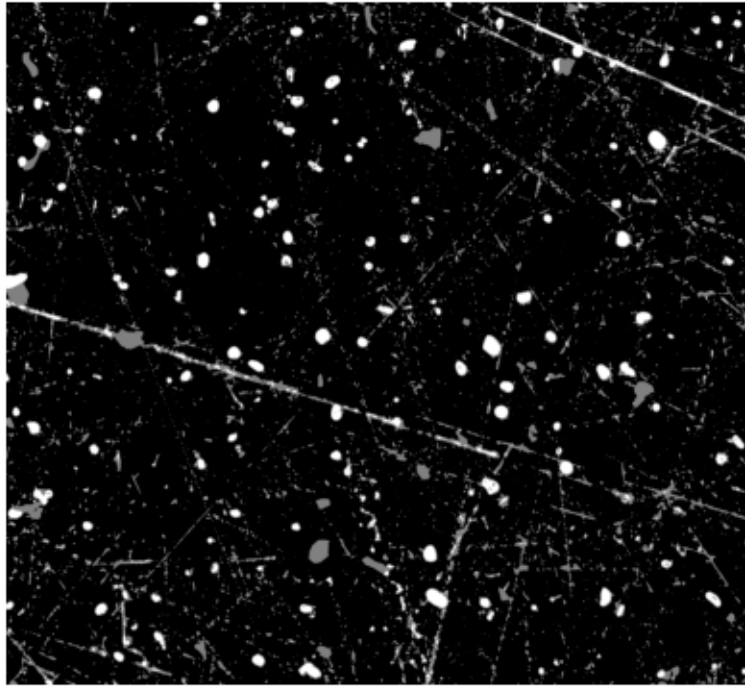
fig = plt.figure(figsize=(5,5))
plt.imshow(clust_matrix_refl, vmin=0, vmax=2, cmap="gray")
plt.axis("off");

print('The image shows Si-rich particles in white, Fe-rich particles in gray,
↳ and the Al matrix in black.'
      'Additionally, there are some falsely identified areas present,
↳ resembling scratches, for example.')

```

The image shows Si-rich particles in white, Fe-rich particles in gray, and the Al matrix in black. Additionally, there are some falsely identified areas

present, resembling scratches, for example.



2.2.3 Step 3: leave only Si particles on the image

```
[7]: new_cluster = []

j=0
for i in range(len(clusters_list_pre)):

    if clusters_list_pre[i] == 0:
        new_cluster.append(0)
        continue

    if clusters_list_pre[i] == 1:

        if clusters_list_pre_new[j] == 0:
            new_cluster.append(0)
            j=j+1
            continue

        if clusters_list_pre_new[j] == 1:
            new_cluster.append(255)
            j=j+1
            continue
```

```

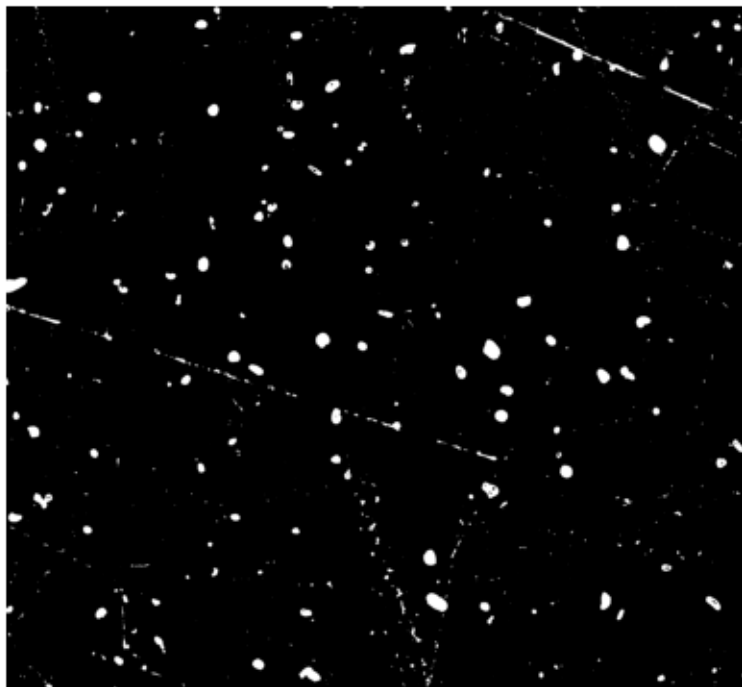
arr = np.array(new_cluster, dtype=np.uint8) # convert matrix to 8 bit to
↳process it with OpenCV
Si_particles = arr.reshape(indy,indx)

fig = plt.figure(figsize=(5,5))
plt.imshow(Si_particles, cmap="gray")
plt.axis("off");

print('The image shows Si-rich particles in white')

```

The image shows Si-rich particles in white



2.3 Image segmentation with OpenCV

```

[8]: # remove small areas with OpenCV function

kernel = np.ones((8,8),np.uint8)
test = cv.morphologyEx(Si_particles, cv.MORPH_OPEN, kernel) # the best function
↳ever for this application

fig = plt.figure(figsize=(5,5))
plt.imshow(test, cmap="gray")
plt.axis("off");

```

```
print('The image has been cleaned by removing small areas that are more likely_  
↳to correspond to artifacts')
```

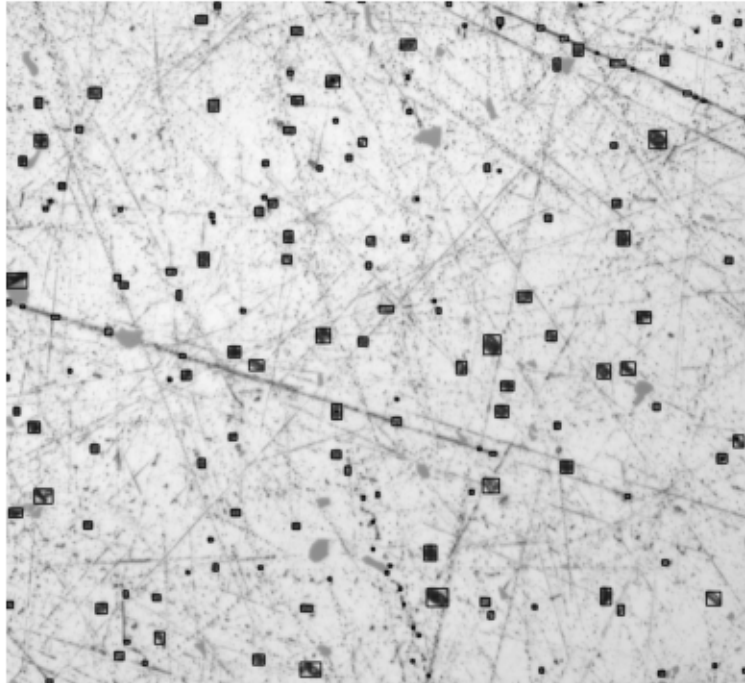
The image has been cleaned by removing small areas that are more likely to correspond to artifacts



```
[9]: # find all Si particles and draw boxes on optical image  
  
contoursSi, _ = cv.findContours(test, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)  
  
representation = image.copy()  
h=40  
  
for c in contoursSi:  
  
    x,y,w,h = cv.boundingRect(c)  
  
    cv.rectangle(representation,(x,y),(x+w,y+h),(0,255,0),3)  
  
fig = plt.figure(figsize=(5,5))  
plt.imshow(representation, cmap="gray")  
plt.axis("off");
```

```
print('All detected Si-rich particles are placed in the boxes.')
```

All detected Si-rich particles are placed in the boxes.



2.4 Remove some falsely identified images using K-means

2.4.1 Exclude particles from the corner and enlarge the particle's box

```
[10]: # modify the countor varibale to avoid including particles on the corner of the
      ↪image
      # here we use 10 pixels from the corner

      enlarge = 10
      contoursSi_cleaned=[]

      for c in contoursSi:

          x,y,w,h = cv.boundingRect(c)

          if w>h:
              plus=w
          else:
              plus=h
```

```

        if (x+plus+enlarge) > indx or (y+plus+enlarge) > indy or (x-enlarge) < 0 or
        ↪(y-enlarge) < 0:
            continue

        contoursSi_cleaned.append(c)

```

```

[11]: enlarge = 10 # use 10 pixels to enlarge the particle box

particles_to_clean=[]

for c in contoursSi_cleaned:

    x,y,w,h = cv.boundingRect(c)

    if w>h:
        plus=w
    else:
        plus=h

    intermed = image8bit[y-enlarge:y+plus+enlarge,x-enlarge:x+plus+enlarge]
    intermed = cv.resize(intermed, (30,30), interpolation = cv.INTER_AREA) #
    ↪reshape so all particles have the same resolution

    particles_to_clean.append(intermed.flatten())

particles_to_clean = np.array(particles_to_clean)

```

2.4.2 Using K-means and PCA visualization to find falsely identified particles.

```

[12]: # reducing the dimensionality of data for visualisation

x_tfm = PCA(n_components=2).fit_transform(particles_to_clean)

# K-means to find outliers

kmeans_model = KMeans(n_clusters=2,random_state=0, n_init=10).fit(x_tfm) #
    ↪fitting model
labels = kmeans_model.labels_ # get label of each pixel (return a list)

fig = plt.figure(figsize=(5, 5))
plt.scatter(x_tfm[:, 0], x_tfm[:, 1], s=100, alpha=1, c=labels)
plt.axis("off");

print('Outliers are marked with black-ish color')

```

Outliers are marked with black-ish color



```
[13]: # Get indices for label 0
indices = [k for k, x in enumerate(labels) if x == 0]

# Calculate grid size for subplots (square root of number of indices)
grid_size = math.isqrt(len(indices)) + (1 if math.isqrt(len(indices))**2 < len(indices) else 0)

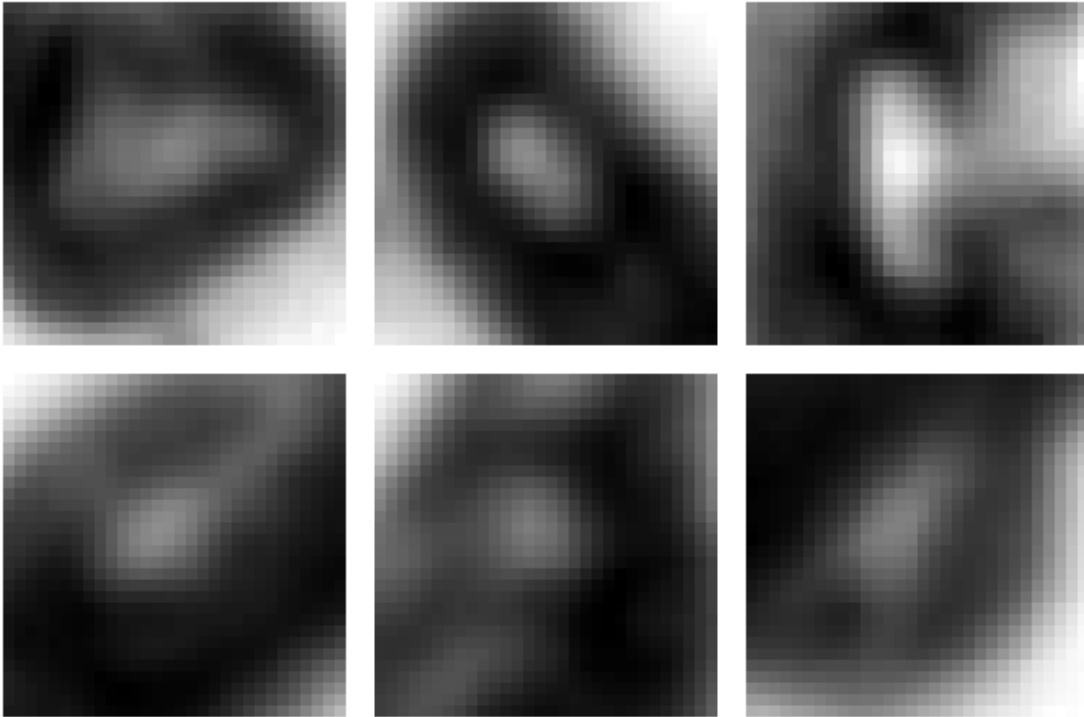
fig, axs = plt.subplots(grid_size, grid_size, figsize=(6, 6))

for i, idx in enumerate(indices):
    representation = particles_to_clean[idx].reshape((30,30))
    axs[i // grid_size, i % grid_size].imshow(representation, cmap="gray")
    axs[i // grid_size, i % grid_size].axis("off")

# Remove unused subplots if any
if len(indices) < grid_size**2:
    for i in range(len(indices), grid_size**2):
        fig.delaxes(axs.flatten()[i])

plt.tight_layout()
plt.show()
```

```
print('Here we display all falsely identified particles that were found after_\n↪performing K-means')
```



Here we display all falsely identified particles that were found after performing K-means

2.4.3 Remove these particles

```
[14]: # modify the countor varibale to remove bad particles

contoursSi_cleaned2=[]

index = 1
indices = [k for k, x in enumerate(labels) if x == index]

for c in indices:

    contoursSi_cleaned2.append(contoursSi_cleaned[c])

# rename the varibale so no need to re-write the code below
contoursSi = contoursSi_cleaned2
```

```
print('Total number of detected Si particles after cleaning = ',  
      len(contoursSi))
```

Total number of detected Si particles after cleaning = 155

3 clustering of Si particles based on their reactivity in 5 mM NaCl solution

3.1 load data

```
[15]: new_water = np.load('rates_of_film_evolution_nacl.npy')
```

3.2 extract all particles

```
[16]: # go through individual particles and write them in an array to prepare for  
      # clustering  
  
particles = []  
  
for c in contoursSi:  
  
    x,y,w,h = cv.boundingRect(c) # find coordinates of individual particles  
  
    if w>h:  
        plus=w  
    else:  
        plus=h  
  
    particle_water = new_water[y-enlarge:y+plus+enlarge,x-enlarge:  
    # x+plus+enlarge] # get the particle  
    particle_water = cv.resize(particle_water, (30,30), interpolation = cv.  
    # INTER_AREA) # reshape so all particles have the same resolution  
    particles.append(particle_water.flatten())  
  
particles = np.array(particles)
```

3.3 K-means to cluster particles activities

3.3.1 elbow method to find the number of clusters

```
[17]: # elbow plot  
  
# plot distortion  
  
distortions = []  
  
n = 20 # The maximum value of n_clusters
```

```

for i in range(2,n):
    model_i = KMeans(n_clusters=i, n_init=10).fit(x_tfm)
    distortion_i = model_i.inertia_ # calculate the distortion score when
    ↪ cluster number equals to i
    distortions.append(distortion_i)

    if i == 2:
        norm = distortion_i

dist_norm=[]

for dist in distortions:
    dist_norm.append(int(dist/norm*100))

fig = plt.figure(figsize=(7,5))
ax = fig.add_subplot(111)

plt.plot(list(range(2,n)),dist_norm,'bo-', linewidth=3, markersize=10)
plt.xlabel('Number of clusters / counts',fontsize=15, fontweight = 'bold')
plt.ylabel('Dispersion / %',fontsize=15, fontweight = 'bold')
#plt.title('Elbow method for images before',fontsize=12)

#ax.spines['top'].set_linewidth(10)
ax.spines['left'].set_linewidth(2)
ax.spines['bottom'].set_linewidth(2)
ax.spines['top'].set_linewidth(0)
ax.spines['right'].set_linewidth(0)

ax.xaxis.set_tick_params(width=2)
ax.yaxis.set_tick_params(width=2)

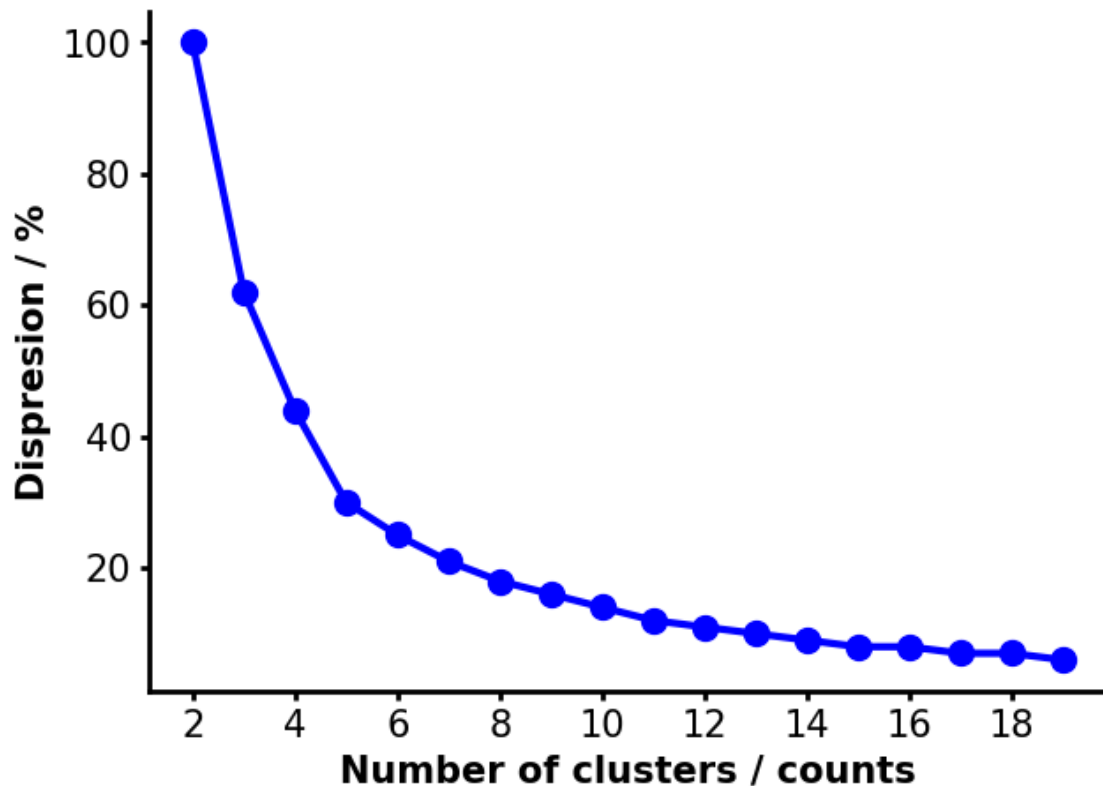
plt.xticks(list(range(2,n,2)))

ax.tick_params(axis='both', which='major', labelsize=15)

print('We conclude that the optimal number of clusters is 3 from this plot')

```

We conclude that the optimal number of clusters is 3 from this plot



3.3.2 applying K-means for clustering and PCA for visualization

```
[18]: x_tfm = PCA(n_components=2).fit_transform(particles)

num_clusters = 3

kmeans_model = KMeans(n_clusters=num_clusters, random_state=0, n_init=10).
    ↪ fit(x_tfm) # fitting model
labels = kmeans_model.labels_ # get label of each pixel (return a list)

fig = plt.figure(figsize=(5, 5))
plt.scatter(x_tfm[:, 0], x_tfm[:, 1], s=100, alpha=1, c=labels)
plt.axis("off");

print('The plot visualizes in 2D the distribution of 3 found clusters. '
      'We number the clusters accordingly:')
print('Cluster 1: green')
print('Cluster 2: black-ish')
print('Cluster 3: orange')
```

The plot visualizes in 2D the distribution of 3 found clusters. We number the clusters accordingly:

Cluster 1: green
Cluster 2: black-ish
Cluster 3: orange



3.3.3 plot centroids

```
[19]: # K-means clustering

num_clusters = 3
kmeans_model = KMeans(n_clusters=num_clusters, random_state=0, n_init=10).
    ↪ fit(particles) # fitting model
labels = kmeans_model.labels_ # get label of each pixel (return a list)
centroids = kmeans_model.cluster_centers_
centroids = centroids.reshape(3,30,30)

label_counts = np.bincount(labels) # count number of each label

fig, axs = plt.subplots(1, num_clusters, figsize=(10,5))

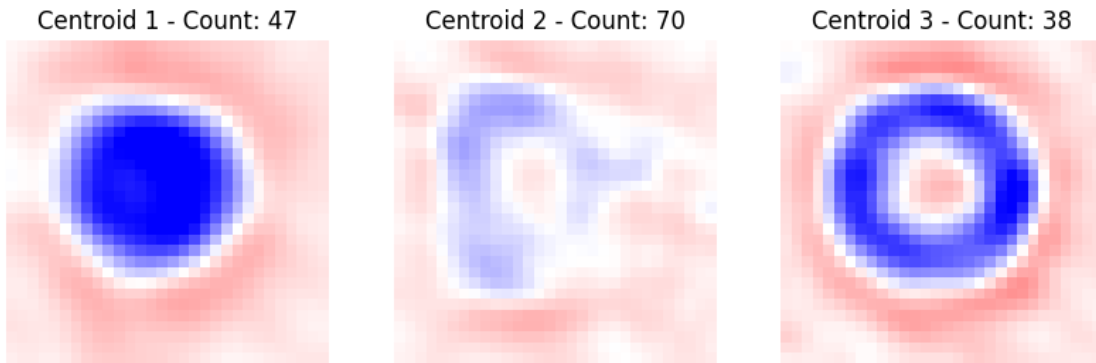
for i in range(num_clusters):
    axs[i].imshow(centroids[i], vmin=-0.007, vmax=0.007, cmap="bwr")
```

```

    axs[i].set_title(f'Centroid {i+1} - Count: {label_counts[i]}') # adding
    ↪title with count
    axs[i].axis("off")

plt.show()

```



3.3.4 unite clusters 2 and 3

```

[20]: indices = [k for k, x in enumerate(labels) if x == 2]
      labels[indices] = 1

```

3.4 plot all particles of category 1

Note that the rate of film evolution does not correspond directly to the correct values that you can find in the paper. Herein, the value of -0.01 corresponds to 0.3 nm/s, and the value of +0.01 corresponds to -0.3 nm/s. Pay attention at the signs

```

[21]: # plot all instances for each category

indices = [k for k, x in enumerate(labels) if x == 0]

fig = plt.figure(figsize=(20,20))

for i in range(len(indices)):

    enlarge = 10 # define border of particle

    x,y,w,h = cv.boundingRect(contoursSi[indices[i]])

    if w>h:
        plus=w
    else:
        plus=h

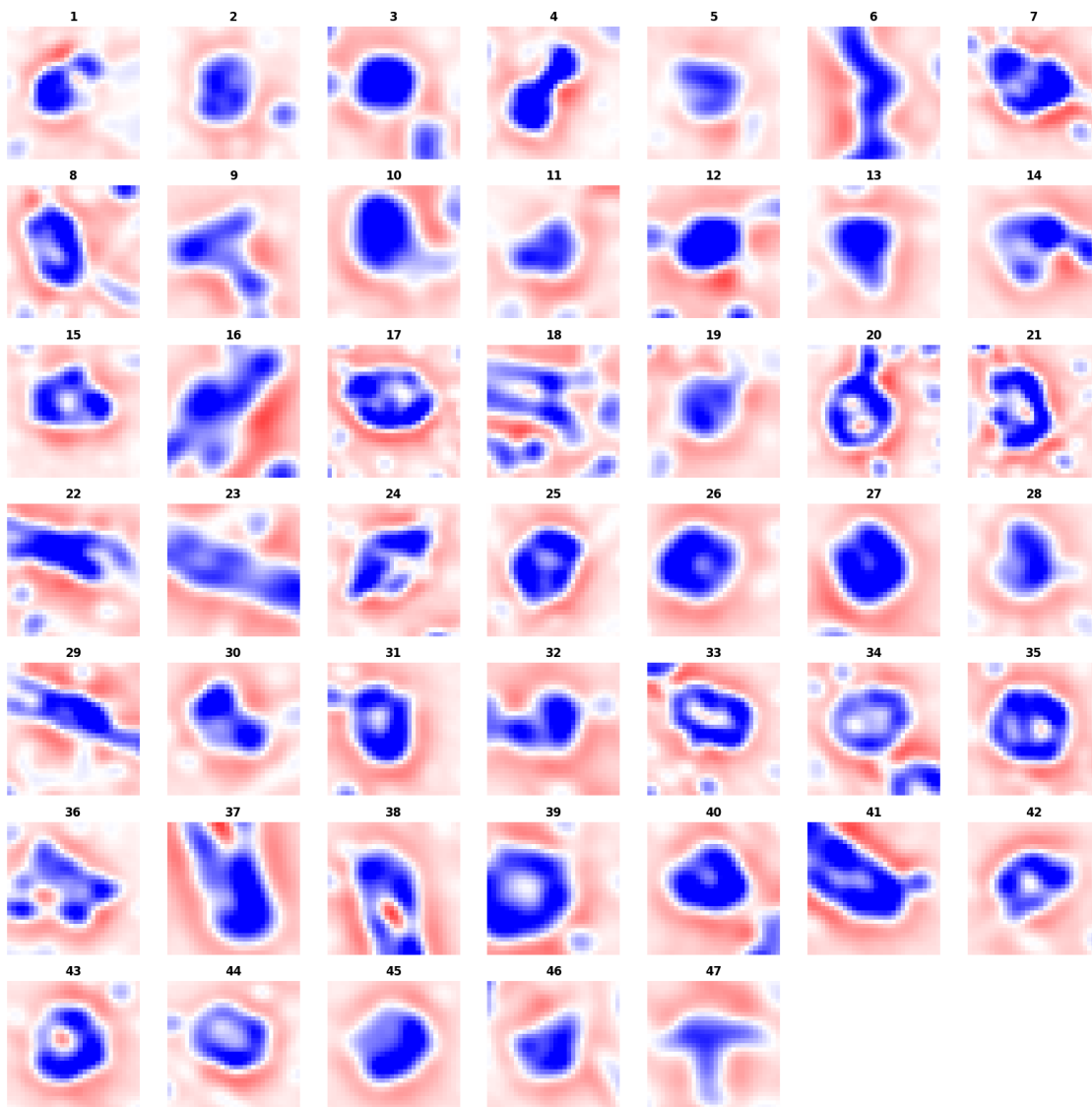
```

```

particle_water = new_water[y-enlarge:y+plus+enlarge,x-enlarge:
↪x+plus+enlarge]
particle_water = cv.resize(particle_water, (30,30), interpolation = cv.
↪INTER_AREA)

plt.subplot(7,7,i+1)
plt.imshow(particle_water,vmin=-0.01, vmax=0.01, cmap="bwr")
plt.axis("off")
plt.title(str(i+1),fontweight="bold")

```



3.5 plot all particles of category 2

```
[22]: # plot all instances for each category

indices = [k for k, x in enumerate(labels) if x == 1]

fig = plt.figure(figsize=(20,40))

for i in range(len(indices)):

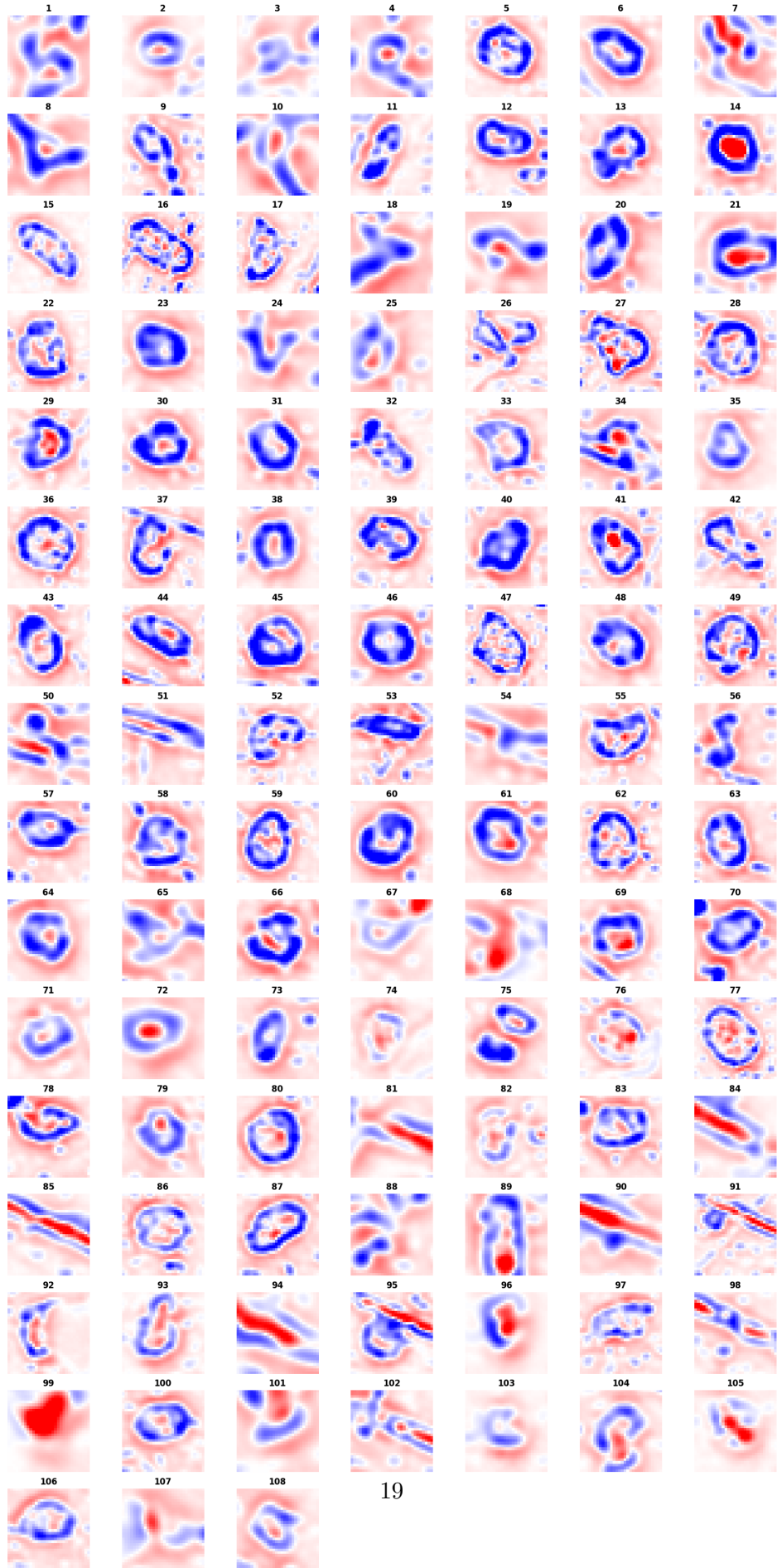
    enlarge = 10 # define border of particle

    x,y,w,h = cv.boundingRect(contoursSi[indices[i]])

    if w>h:
        plus=w
    else:
        plus=h

    particle_water = new_water[y-enlarge:y+plus+enlarge,x-enlarge:
↪x+plus+enlarge]
    particle_water = cv.resize(particle_water, (30,30), interpolation = cv.
↪INTER_AREA)

    plt.subplot(16,7,i+1)
    plt.imshow(particle_water,vmin=-0.01, vmax=0.01, cmap="bwr")
    plt.axis("off")
    plt.title(str(i+1),fontweight="bold")
```



4 Clustering of Si particles according to the reactivity change after addition of 10 mM H₂SO₄

4.1 load data

```
[23]: image_water = np.load('rates_of_film_evolution_nacl.npy')
      image_acid = np.load('rates_of_film_evolution_acid.npy')
```

4.2 segment images

```
[24]: indy,indx = image_water.shape

      # segmentation of images

      cluster_water = image_water.reshape((-1,1))

      optimal_model = KMeans(n_clusters=5,random_state=0, n_init=10).
      ↪fit(cluster_water) # fitting model
      clusters_list_water = optimal_model.labels_ # get label of each pixel (return a
      ↪list)

      cluster_water = clusters_list_water.reshape(indy,indx)

      cluster_acid = image_acid.reshape((-1,1))

      optimal_model = KMeans(n_clusters=5,random_state=0, n_init=10).
      ↪fit(cluster_acid) # fitting model
      clusters_list_acid = optimal_model.labels_ # get label of each pixel (return a
      ↪list)

      cluster_acid = clusters_list_acid.reshape(indy,indx)
```

4.3 build correlation matrices for all particles

```
[25]: # go through individual particles

      particles = []

      for c in contoursSi:

          x,y,w,h = cv.boundingRect(c) # find coordinates of individual particles
```

```

x1=x
x2=x+w
y1=y
y2=y+h

particle_water = cluster_water[y1:y2,x1:x2]
particle_water = particle_water.reshape((-1,1))

particle_acid = cluster_acid[y1:y2,x1:x2]
particle_acid = particle_acid.reshape((-1,1))

# plot correlation matrix

correlation_particle = np.zeros((5,5))

for i in range(5):
    for j in range(5):

        cluster_a = i
        cluster_b = j

        indices_b = [k for k, x in enumerate(particle_water) if x == ␣
↪cluster_b]
        indices_a = [k for k, x in enumerate(particle_acid) if x == ␣
↪cluster_a]

        number_common = len(list(set(indices_a).intersection(indices_b)))

        correlation_particle[i,j] = int(number_common) # correlation matrix ␣
↪for individual particle

    correlation_particle = correlation_particle/np.sum(correlation_particle)*100

    particles.append(correlation_particle.flatten()) # save correlation matrix ␣
↪of particle

particles = np.array(particles)

```

4.4 applying K-means for clustering and PCA for visualization

```

[26]: # reducing the dimensionality of data

x_tfm_1 = PCA(n_components=2).fit_transform(particles)

# Kmeans

```

```

kmeans_model = KMeans(n_clusters=2,random_state=0, n_init=10).fit(particles) #  

    ↪ fitting model  

labels = kmeans_model.labels_ # get label of each pixel (return a list)  

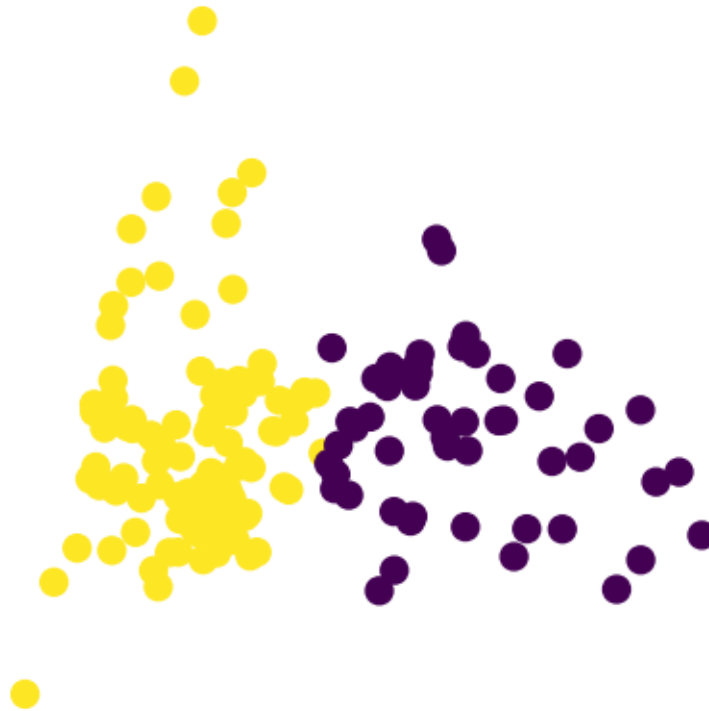
  

fig = plt.figure(figsize=(5, 5))  

plt.scatter(x_tfm_1[:, 0], x_tfm_1[:, 1], s=100, alpha=1, c=labels)  

plt.axis("off");

```



4.5 finding the closest particles to centroids

```

[27]: centroids = kmeans_model.cluster_centers_  

averaged = centroids[1]  

indi, nothing = particles.shape  

score=[]  

for i in range(indi):  

    matrix = particles[i]  

    total=0  

    for j in range(len(matrix)):

```

```

        total = (matrix[j]-averaged[j])*(matrix[j]-averaged[j])+total

    score.append(total)

print('Particle, closest to centroid 1: ' + str(score.index(min(score))))

averaged = centroids[0]

indi, nothing = particles.shape
score=[]

for i in range(indi):
    matrix = particles[i]

    total=0
    for j in range(len(matrix)):
        total = (matrix[j]-averaged[j])*(matrix[j]-averaged[j])+total

    score.append(total)

print('Particle, closest to centroid 2: ' + str(score.index(min(score))))

```

Particle, closest to centroid 1: 97

Particle, closest to centroid 2: 39

4.5.1 plotting these particles

```

[28]: number = 97

# show this particles on the image

fig = plt.figure(figsize=(5,3))
enlarge = 10

x,y,w,h = cv.boundingRect(contoursSi[number])

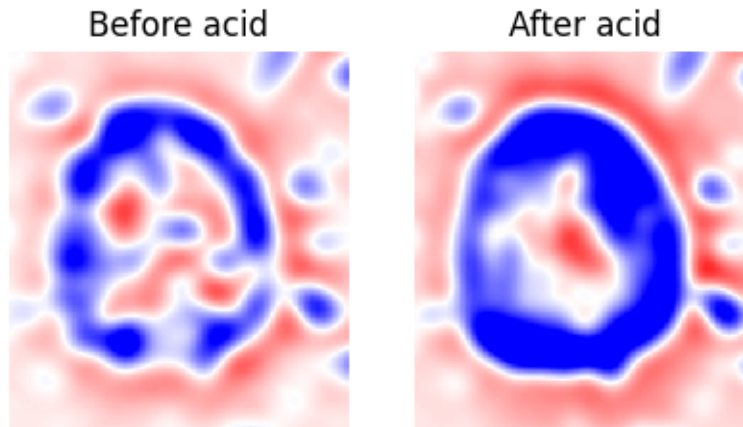
plt.subplot(1,2,1)
plt.imshow(image_water[y-enlarge:y+h+enlarge,x-enlarge:x+w+enlarge],vmin=-0.01,
    ↪vmax=0.01, cmap="bwr")
plt.title('Before acid')
plt.axis("off");

plt.subplot(1,2,2)
plt.imshow(image_acid[y-enlarge:y+h+enlarge,x-enlarge:x+w+enlarge],vmin=-0.01,
    ↪vmax=0.01, cmap="bwr")
plt.title('After acid')
plt.axis("off");

```

```
print("Pseudo ceontroid 1")
```

Pseudo ceontroid 1



```
[29]: number = 39

# show this particles on the image

fig = plt.figure(figsize=(5,3))
enlarge = 10

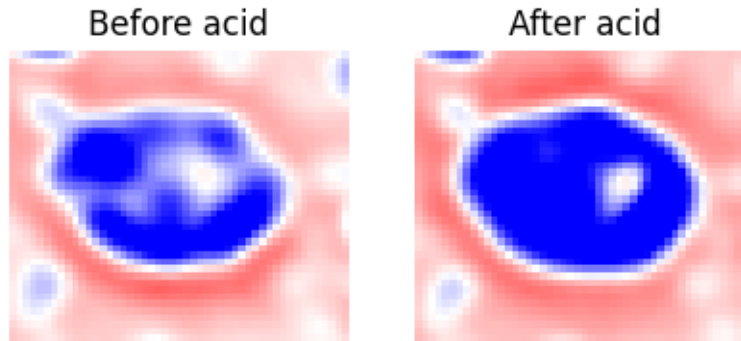
x,y,w,h = cv.boundingRect(contoursSi[number])

plt.subplot(1,2,1)
plt.imshow(image_water[y-enlarge:y+h+enlarge,x-enlarge:x+w+enlarge],vmin=-0.01,
           ↪vmax=0.01, cmap="bwr")
plt.title('Before acid')
plt.axis("off");

plt.subplot(1,2,2)
plt.imshow(image_acid[y-enlarge:y+h+enlarge,x-enlarge:x+w+enlarge],vmin=-0.01,
           ↪vmax=0.01, cmap="bwr")
plt.title('After acid')
plt.axis("off");

print("Pseudo centroid 2")
```

Pseudo centroid 2



5 plot correlation matrix for for demonstration

5.1 choose particle and segment images

```
[30]: x,y,w,h = cv.boundingRect(contoursSi[7]) # choose particle
      enlarge=10

      image_water_particle = image_water[y-enlarge:y+h+enlarge,x-enlarge:x+w+enlarge]
      image_acid_particle = image_acid[y-enlarge:y+h+enlarge,x-enlarge:x+w+enlarge]

      indy,indx = image_water_particle.shape

      # segmentation of images

      cluster_water = image_water_particle.reshape((-1,1))

      optimal_model = KMeans(n_clusters=5,random_state=0, n_init=10).
      ↪fit(cluster_water) # fitting model
      clusters_list_water = optimal_model.labels_ # get label of each pixel (return a
      ↪list)

      cluster_water = clusters_list_water.reshape(indy,indx)

      cluster_acid = image_acid_particle.reshape((-1,1))

      optimal_model = KMeans(n_clusters=5,random_state=0, n_init=10).
      ↪fit(cluster_acid) # fitting model
      clusters_list_acid = optimal_model.labels_ # get label of each pixel (return a
      ↪list)

      cluster_acid = clusters_list_acid.reshape(indy,indx)
```

5.2 Change the segmented values to be in ascending order

5.2.1 for image in NaCl

```
[31]: # find average values of each cluster

indices_b0 = [i for i, x in enumerate(clusters_list_water) if x == 0]
indices_b1 = [i for i, x in enumerate(clusters_list_water) if x == 1]
indices_b2 = [i for i, x in enumerate(clusters_list_water) if x == 2]
indices_b3 = [i for i, x in enumerate(clusters_list_water) if x == 3]
indices_b4 = [i for i, x in enumerate(clusters_list_water) if x == 4]

water_values = image_water.reshape((-1,1))

b0 = []
for i in indices_b0:
    b0.append(water_values[i])

b0=np.array(b0)

b1 = []
for i in indices_b1:
    b1.append(water_values[i])

b1=np.array(b1)

b2 = []
for i in indices_b2:
    b2.append(water_values[i])

b2=np.array(b2)

b3 = []
for i in indices_b3:
    b3.append(water_values[i])

b3=np.array(b3)

b4 = []
for i in indices_b4:
    b4.append(water_values[i])

b4=np.array(b4)

print('data for NaCl')

print('cluster 0 = ', round(np.mean(b0),4), "\u00B1", round(np.std(b0),4))
```

```

print('cluster 1 = ', round(np.mean(b1),4), "\u00B1", round(np.std(b1),4))
print('cluster 2 = ', round(np.mean(b2),4), "\u00B1", round(np.std(b2),4))
print('cluster 3 = ', round(np.mean(b3),4), "\u00B1", round(np.std(b3),4))
print('cluster 4 = ', round(np.mean(b4),4), "\u00B1", round(np.std(b4),4))

```

data for NaCl

```

cluster 0 = 0.0008 ± 0.0012
cluster 1 = 0.0007 ± 0.0014
cluster 2 = 0.0006 ± 0.0015
cluster 3 = 0.0007 ± 0.0016
cluster 4 = 0.0005 ± 0.0015

```

Note that the rate of film evolution does not correspond directly to the correct values that you can find in the paper. Herein, the value of -0.01 corresponds to 0.3 nm/s, and the value of +0.01 corresponds to -0.3 nm/s. Pay attention at the signs

```

[32]: # change clusters

clusters_list_water_changed = clusters_list_water.copy()

indices_b0 = [i for i, x in enumerate(clusters_list_water) if x == 0]
indices_b1 = [i for i, x in enumerate(clusters_list_water) if x == 1]
indices_b2 = [i for i, x in enumerate(clusters_list_water) if x == 2]
indices_b3 = [i for i, x in enumerate(clusters_list_water) if x == 3]
indices_b4 = [i for i, x in enumerate(clusters_list_water) if x == 4]

for i in indices_b0:
    clusters_list_water_changed[i] = 3

for i in indices_b1:
    clusters_list_water_changed[i] = 1

for i in indices_b2:
    clusters_list_water_changed[i] = 4

for i in indices_b3:
    clusters_list_water_changed[i] = 2

for i in indices_b4:
    clusters_list_water_changed[i] = 0

```

5.2.2 for image in Acid

```

[33]: indices_b0 = [i for i, x in enumerate(clusters_list_acid) if x == 0]
indices_b1 = [i for i, x in enumerate(clusters_list_acid) if x == 1]
indices_b2 = [i for i, x in enumerate(clusters_list_acid) if x == 2]
indices_b3 = [i for i, x in enumerate(clusters_list_acid) if x == 3]
indices_b4 = [i for i, x in enumerate(clusters_list_acid) if x == 4]

```

```

acid_values = image_acid.reshape((-1,1))

b0 = []
for i in indices_b0:
    b0.append(acid_values[i])

b0=np.array(b0)

b1 = []
for i in indices_b1:
    b1.append(acid_values[i])

b1=np.array(b1)

b2 = []
for i in indices_b2:
    b2.append(acid_values[i])

b2=np.array(b2)

b3 = []
for i in indices_b3:
    b3.append(acid_values[i])

b3=np.array(b3)

b4 = []
for i in indices_b4:
    b4.append(acid_values[i])

b4=np.array(b4)

print('data for acid')

print('cluster 0 = ', round(np.mean(b0),4), "\u00B1", round(np.std(b0),4))
print('cluster 1 = ', round(np.mean(b1),4), "\u00B1", round(np.std(b1),4))
print('cluster 2 = ', round(np.mean(b2),4), "\u00B1", round(np.std(b2),4))
print('cluster 3 = ', round(np.mean(b3),4), "\u00B1", round(np.std(b3),4))
print('cluster 4 = ', round(np.mean(b4),4), "\u00B1", round(np.std(b4),4))

data for acid
cluster 0 = 0.0006 ± 0.0015
cluster 1 = 0.0005 ± 0.0024
cluster 2 = 0.0004 ± 0.0022
cluster 3 = 0.0005 ± 0.0021
cluster 4 = 0.0003 ± 0.0026

```

Note that the rate of film evolution does not correspond directly to the correct values that you can find in the paper. Herein, the value of -0.01 corresponds to 0.3 nm/s, and the value of +0.01 corresponds to -0.3 nm/s. Pay attention at the signs

```
[34]: clusters_list_acid_changed = clusters_list_acid.copy()

indices_b0 = [i for i, x in enumerate(clusters_list_acid) if x == 0]
indices_b1 = [i for i, x in enumerate(clusters_list_acid) if x == 1]
indices_b2 = [i for i, x in enumerate(clusters_list_acid) if x == 2]
indices_b3 = [i for i, x in enumerate(clusters_list_acid) if x == 3]
indices_b4 = [i for i, x in enumerate(clusters_list_acid) if x == 4]

for i in indices_b0:
    clusters_list_acid_changed[i] = 3

for i in indices_b1:
    clusters_list_acid_changed[i] = 0

for i in indices_b2:
    clusters_list_acid_changed[i] = 2

for i in indices_b3:
    clusters_list_acid_changed[i] = 4

for i in indices_b4:
    clusters_list_acid_changed[i] = 1
```

5.3 generate matrix and plot it

```
[35]: correlation = np.zeros((5,5))
total = len(clusters_list_acid_changed)

for i in range(5):
    for j in range(5):

        cluster_a = i
        cluster_b = j

        indices_b = [k for k, x in enumerate(clusters_list_water_changed) if x == cluster_b]
        indices_a = [k for k, x in enumerate(clusters_list_acid_changed) if x == cluster_a]

        number_common = len(list(set(indices_a).intersection(indices_b)))

        #correlation[i,j] = int(number_common/total*100)
        correlation[i,j] = int(number_common)
```

```

[36]: plt.figure(figsize=(5,5))

mat = np.flip(correlation)/np.sum(correlation)*100

plt.imshow(mat, vmin=0,vmax=5, cmap='Blues')

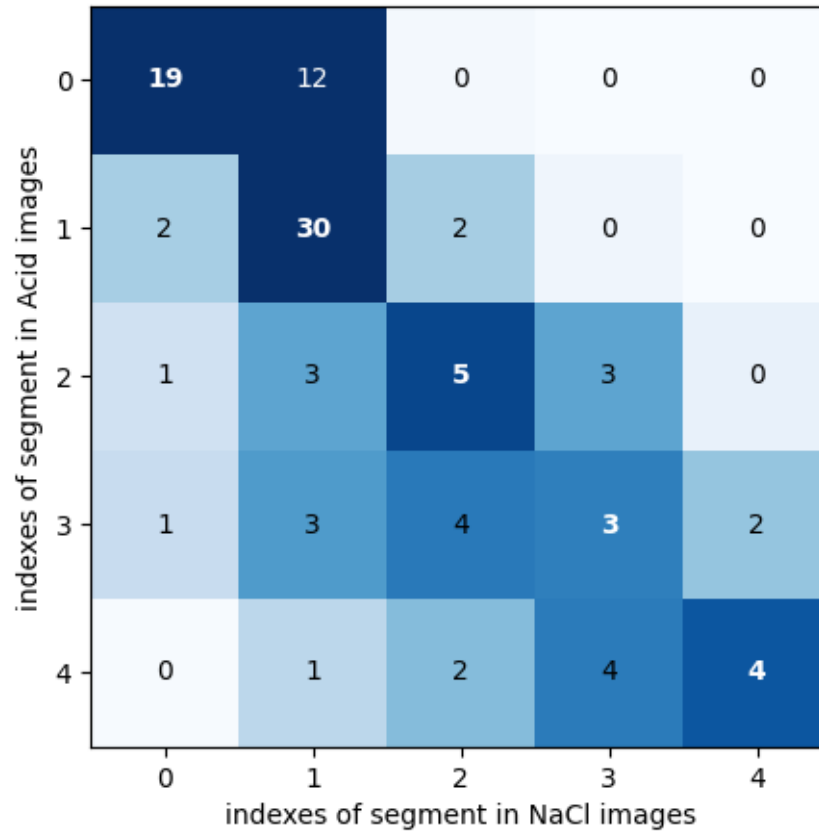
plt.xlabel('indexes of segment in NaCl images')
plt.ylabel('indexes of segment in Acid images')
#plt.colorbar()

# Add the cell values to the plot using annotate
for i in range(mat.shape[0]):
    for j in range(mat.shape[1]):

        text = str(round(mat[i][j]))
        color = 'white' if (i == j) or (j==1 and i==0) else 'black'
        fontweight = 'bold' if i == j else 'normal'
        fontsize = 10 # Set fontsize
        plt.annotate(text, xy=(j, i), ha='center', va='center',
            ↪fontweight=fontweight, color=color,fontsize = fontsize)

# Show the plot
plt.show()

```



6 COMSOL vs experiment

6.1 load, process and plot data from COMSOL simulations

```
[37]: def process_data(file_name):
    data3 = np.loadtxt(file_name)
    length = data3[:,0]
    ynew = data3[:,1]
    rad25 = length
    O_rad25 = np.flip(ynew)
    azm = np.linspace(0, 2 * np.pi)
    r, th = np.meshgrid(rad25, azm)
    z = np.tile(O_rad25, (r.shape[0], 1))
    return th, r, z

th_nacl, r_nacl, z_nacl = process_data("comsol_nacl.txt")
th_acid, r_acid, z_acid = process_data("comsol_acid.txt")

fig, axs = plt.subplots(1, 2, subplot_kw=dict(projection='polar'),
    figsize=(8,4))
```

```

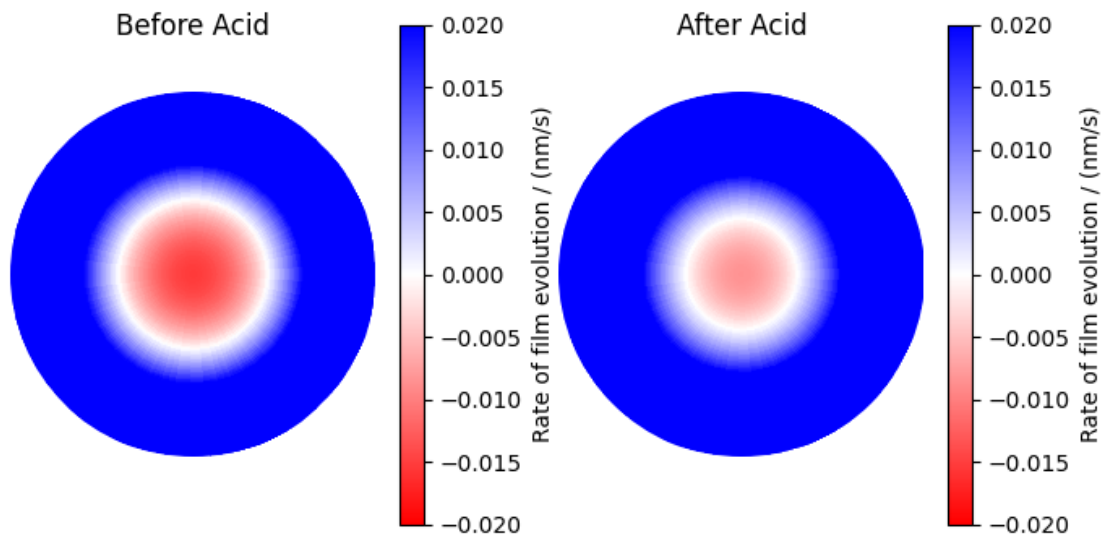
axs[0].pcolormesh(th_nacl, r_nacl, z_nacl, vmin=-0.02, vmax=0.02, cmap="bwr_r")
axs[0].axis("off")
axs[0].set_title("Before Acid")

c = axs[1].pcolormesh(th_acid, r_acid, z_acid, vmin=-0.02, vmax=0.02,
↪cmap="bwr_r")
axs[1].axis("off")
axs[1].set_title("After Acid")

fig.colorbar(c, ax=axs[0], label='Rate of film evolution / (nm/s)')
fig.colorbar(c, ax=axs[1], label='Rate of film evolution / (nm/s)')

plt.show()

```



6.2 compare 1D profiles from COMSOL simulations and experiment

6.2.1 Comsol

```

[38]: def process_data(file_name):

    data3 = np.loadtxt(file_name)

    length = data3[:,0]
    ynew = data3[:,1]

    inter = length+np.max(length)
    x = np.concatenate((length,inter))
    y = np.concatenate((ynew,np.flip(ynew)))

```

```

    return x, y

x_nacl, y_nacl = process_data("comsol_nacl.txt")
x_acid, y_acid = process_data("comsol_acid.txt")

```

```

[39]: fig = plt.figure(figsize=(7,5))

ax = fig.add_subplot(111)

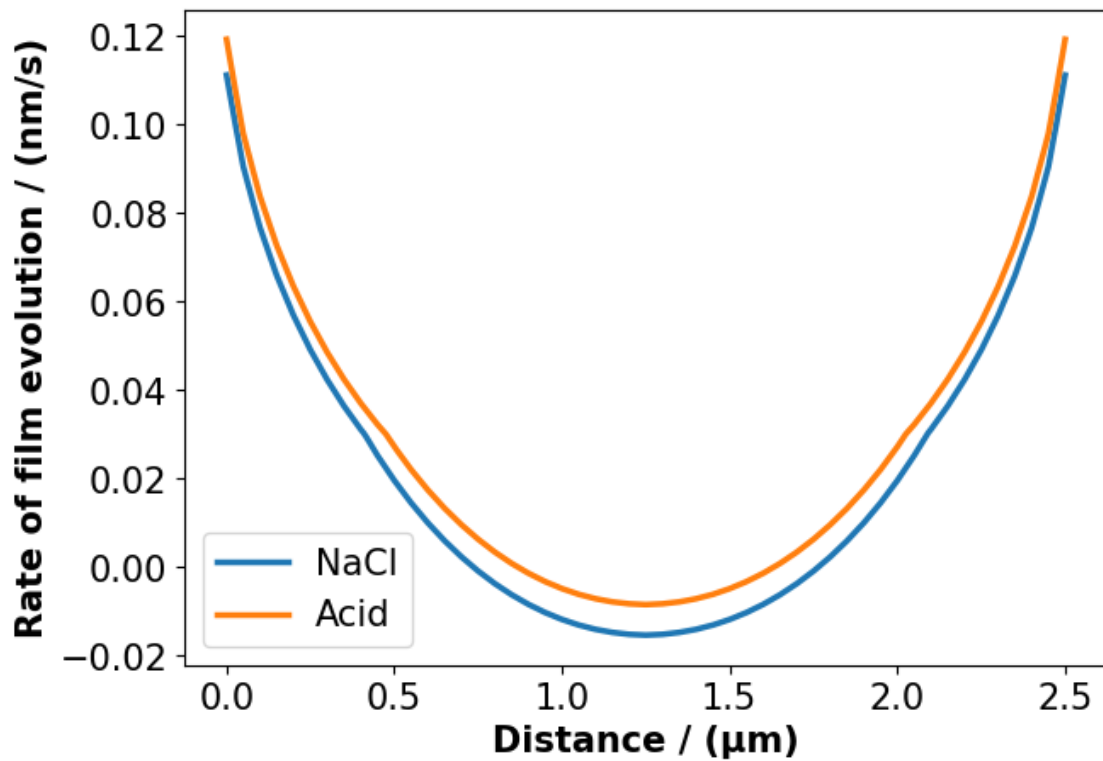
plt.plot(x_nacl,y_nacl, linewidth=2.5, label='NaCl');
plt.plot(x_acid,y_acid, linewidth=2.5, label='Acid');

plt.xlabel('Distance / (\u00B5m)',fontsize=15, fontweight = 'bold')
plt.ylabel('Rate of film evolution / (nm/s)',fontsize=15, fontweight = 'bold')

ax.tick_params(axis='both', which='major', labelsize=15)

plt.legend(fontsize='15');

```



6.2.2 Experiment

```
[40]: def process_image(image, contoursSi, particle_number):  
    # Bounding Rectangle for a chosen particle  
    x,y,w,h = cv.boundingRect(contoursSi[particle_number])  
  
    # Extract the particle from the image  
    particle_image = image[y:y+h,x:x+w-3]  
  
    # Determine the shape of the particle image  
    yind, xind = particle_image.shape  
  
    # Calculate y  
    y = -particle_image[int(yind/2),:]*0.3/0.01 # conversion of the units to  
    ↪ the right one  
  
    # Calculate x  
    x = np.arange(0, xind*0.062, 0.062) # size of one pixel is 0.062 nm  
  
    return x, y  
  
number = 7 # choose the particle  
  
x_nacl, y_nacl = process_image(image_water, contoursSi, number)  
x_acid, y_acid = process_image(image_acid, contoursSi, number)  
  
fig = plt.figure(figsize=(7,5))  
  
ax = fig.add_subplot(111)  
  
plt.plot(x_nacl,y_nacl, linewidth=2.5, label='NaCl');  
plt.plot(x_acid,y_acid, linewidth=2.5, label='Acid');  
  
plt.xlabel('Distance / (nm)',fontsize=15, fontweight = 'bold')  
plt.ylabel('Rate of film evolution / (nm/s)',fontsize=15, fontweight = 'bold')  
  
ax.tick_params(axis='both', which='major', labelsize=15)  
  
plt.legend(fontsize='15');
```

