Pomiary wielkości cząsteczek przy wykorzystaniu teorii ruchów Browna

Piotr Korczyk

Spis treści

1 Wstęp

Pomiar wielkości cząsteczek na podstawie ruchów Browna opiera się na pomiarze przemieszczeń cząsteczki. Na tej podstawie możliwe jest policzenie współczynnika dyfuzji dla cząsteczki:

$$D = \frac{k\,T}{3\pi\,\mu\,d_p} \tag{1}$$

Gdzie k=1,38·10^{-23} JK^{-1} jest stałą Boltzmana a μ jest lepkością płynu w którym poruszają się cząstki (dla wody μ=8,9·10^{-4} kg/ms), T – temperaturą płynu, d_p – poszukiwaną średnicą cząsteczki.

Z drugiej strony:

$$\langle s^2 \rangle = 2\,D\,\Delta t \tag{2}$$

Gdzie s jest przemieszczeniem cząsteczki w czasie Δt. Ponieważ mierzymy przemieszczenia tylko w jednej płaszczyźnie musimy skorzystać z przybliżenia:

$$\langle s^2 \rangle = \langle \Delta x^2 + \Delta y^2 + \Delta z^2 \rangle = \langle \Delta x^2 \rangle + \langle \Delta y^2 \rangle + \langle \Delta z^2 \rangle = 3/2 (\langle \Delta x^2 \rangle + \langle \Delta y^2 \rangle) \quad (3)$$

Zakładamy tutaj, że cząstki poruszają się izotropowo, zatem średnią odległość w kierunku prostopadłym do płaszczyzny fotografii można przybliżyć przez średnie odległości w dwóch mierzonych w eksperymentach kierunkach.

Zatem wielkości jakie należy zmierzyć w eksperymencie to położenia x i y cząsteczki w chwilach czasu oddalonych o Δt.

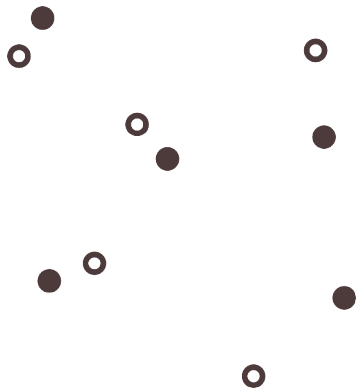
2 Idea algorytmu

Algorytm, który mierzyłby trajektorię cząsteczek powinien lokalizować cząsteczki na kolejnych fotografiach. Odległość jaką przebyła cząstka w czasie Δt byłaby więc odległością pomiędzy położeniami obrazu cząsteczki na dwóch kolejnych fotografiach.

Należy pamiętać, że każda cząsteczka porusza się w tym przypadku chaotycznie i niezależnie od innych cząsteczek. Nie można tu więc stosować metod opartych na statystycznej analizie ruchu większej grupy cząsteczek (tak jak w Particle Image Velocimetry). W tym przypadku potrzebny jest algorytm, który będzie lokalizował pojedyncze cząsteczki na poszczególnych zdjęciach.

Po pierwsze zatem algorytm powinien znajdować obrazy cząsteczek na fotografiach. Można to zrealizować przez szukanie lokalnych maksimów na fotografiach czyli pikseli które mają większą jasność niż ich otoczenie. Należy pamiętać, że z powodu szumów lokalne maksima pojawiają się spontanicznie w różnych miejscach fotografii. Należy więc za położenia cząsteczek uważać tylko odpowiednio jasne maksima. Wartość progu ustala się metodą prób i błędów.

Gdyby fotografie zawierały obrazy tylko jednej i tej samej cząsteczki, to samo zlokalizowanie cząsteczki wystarczyłoby do znalezienia trajektorii. W sytuacji gdy obserwujemy zawiesinę cząsteczek problem staje się bardziej skomplikowany. Po wykryciu pozycji cząsteczek na jednej i drugiej fotografii otrzymujemy dwa zbiory położeń tych samych cząsteczek w dwóch chwilach czasu. Należy następnie przyporządkować sobie elementy z obu zbiorów. Inaczej mówiąc każdemu elementowi z pierwszego zbioru, który jest położeniem cząsteczki na pierwszej fotografii należy przyporządkować położenie tej samej cząsteczki na drugiej fotografii. Przyporządkowanie to musi być wzajemnie jednoznaczne, tzn. jednemu położeniu na pierwszej fotografii można przyporządkować tylko jedno położenie na drugiej fotografii i odwrotnie.

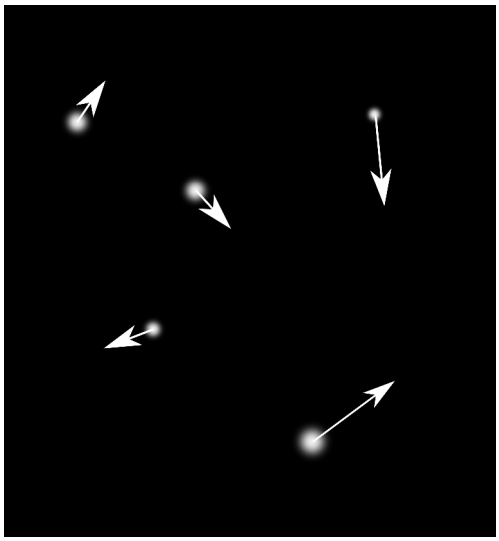


Rys. 1: Ilustracja problemu. Puste kółka – położenia cząsteczek z pierwszej fotografii, kółka pełne – położenia cząsteczek na drugiej fotografii. Znajomość położeń cząsteczek na dwóch fotografiach nie wystarcza do obliczenia przemieszczeń. Położenia należy odpowiednio uporządkować w pary.

Jak może wyglądać algorytm, który realizuje takie przyporządkowanie?

Zapostulujmy najpierw dwa warunki jakie muszą spełniać eksperymentalne sekwencje fotografii aby nadawały się do tego typu pomiarów:

- Przemieszczenia cząsteczek powinny być większe od ich dyfrakcyjnego obrazu.
- Przemieszczenia cząsteczek powinny być mniejsze od typowej odległości pomiędzy cząsteczkami.

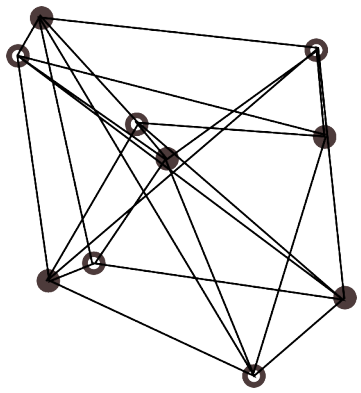


Rys. 2: Fotografie z których można odczytać informacje o przemieszczeniach muszą spełniać dwa warunki: po pierwsze przemieszczenia powinny być większe od wielkości obrazów cząsteczek, po drugie odległości między cząsteczkami powinny być większe od przemieszczeń.

Pierwszy warunek zapewnia, że przemieszczenie cząsteczki będzie mierzone z rozsądną dokładnością. Dokładność pomiaru położenia cząsteczki jest bowiem rzędu wielkości obrazu reprezentującego cząsteczkę. Przemieszczenie powinno więc być dużo większe od dokładności pomiaru położenia, bo wtedy w ogóle ma sens pomiar przemieszczenia. Warunek ten może być spełniony przez odpowiedni dobór interwału Δt w czasie wykonywania eksperymentu.

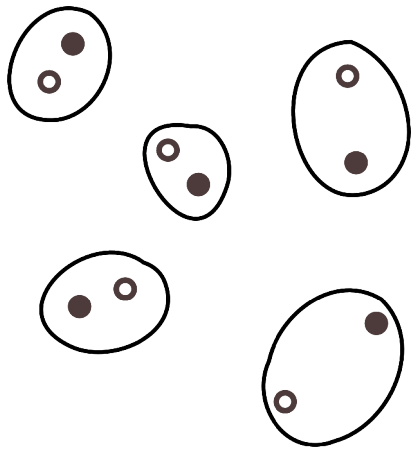
Drugi warunek zapewnia, że możliwe jest przyporządkowanie sobie położeń z dwóch fotografii, czyli identyfikacja ich jako położenia tej samej cząstki w dwóch chwilach czasu. Może on być spełniony przez odpowiedni dobór gęstości zawiesiny.

Idea algorytmu przyporządkowującego sobie położenia cząsteczek jest związana z drugim warunkiem. Jeśli jest spełniony możliwa jest identyfikacja położeń tej samej cząsteczki przez przyporządkowywanie sobie położeń leżących najbliżej siebie. Zakładamy tutaj, że przy spełnionym drugim warunku najbardziej prawdopodobna jest sytuacja, że cząstka na drugiej fotografii leży bliżej swego położenia początkowego niż wszystkie inne cząstki. Inaczej mówiąc, jeśli policzymy odległości pomiędzy daną pozycją z pierwszej fotografii a wszystkimi pozycjami z drugiej fotografii to najmniejsza wartość tej odległości będzie dla pozycji odpowiadających tej samej cząstce.



Rys. 3: Porządkowanie położeń w pary odbywa się przez analizę odległości pomiędzy wszystkimi punktami, które mogą tworzyć parę tzn. dla każdego pustego kółka liczymy odległości do wszystkich pełnych kółek.

Algorytm przyporządkowujący działa więc tak: liczone są odległości pomiędzy wszystkimi pozycjami z pierwszego zbioru a wszystkimi pozycjami z drugiego zbioru co odpowiada policzeniu wszystkich potencjalnych przemieszczeń. Następnie wyszukuje się najmniejszą wartość przemieszczenia. Pozycje odpowiadające tej wartości łączone są w parę, która traktowana jest jako położenia tej samej cząsteczki. Przyporządkowana para położeń wyrzucana jest z analizowanych zbiorów tak, żeby nie była ona uwzględniana ponownie. Po czym procedurę przyporządkowywania sobie pary położeń powtarza się do wyczerpania zbiorów.



Rys. 4: Ilustracja działania algorytmu przyporządkowującego. Puste kółka – położenia cząsteczek z pierwszej fotografii, kółka pełne – położenia cząsteczek na drugiej fotografii. Obwódki pokazują sposób łączenia położeń w pary.

3 Omówienie programów

Proponowane tutaj programy działają, a ściśle mówiąc działały na obrazach, które autor dostał do analizy i nie były testowane na innych fotografiach. Nie jest to oprogramowanie napisane porządnie i uniwersalne, tyczy się to głównie lokalizacji cząsteczek, gdzie algorytm namierzania cząsteczek można byłoby przy pewnym nakładzie czasu usprawnić i dodać podpikselową dokładność pomiaru. Pojawia się tutaj problem jak zdefiniować obraz cząsteczki tzn. jakie kryteria mają być przyjęte w algorytmie, aby cząsteczki były poprawnie identyfikowane. Definicja taka nie jest stała. Na zdjęciach z różnych eksperymentów cząsteczki będą wyglądały zazwyczaj inaczej. Do tego dochodzą jeszcze defekty obrazu, które są różne w różnych eksperymentach. To wszystko sprawia, że napisane programy do lokalizacji cząsteczek nie będą działały poprawnie dla dowolnych fotografiach z cząsteczkami, należy raczej dostosować je do specyfiki danego eksperymentu.

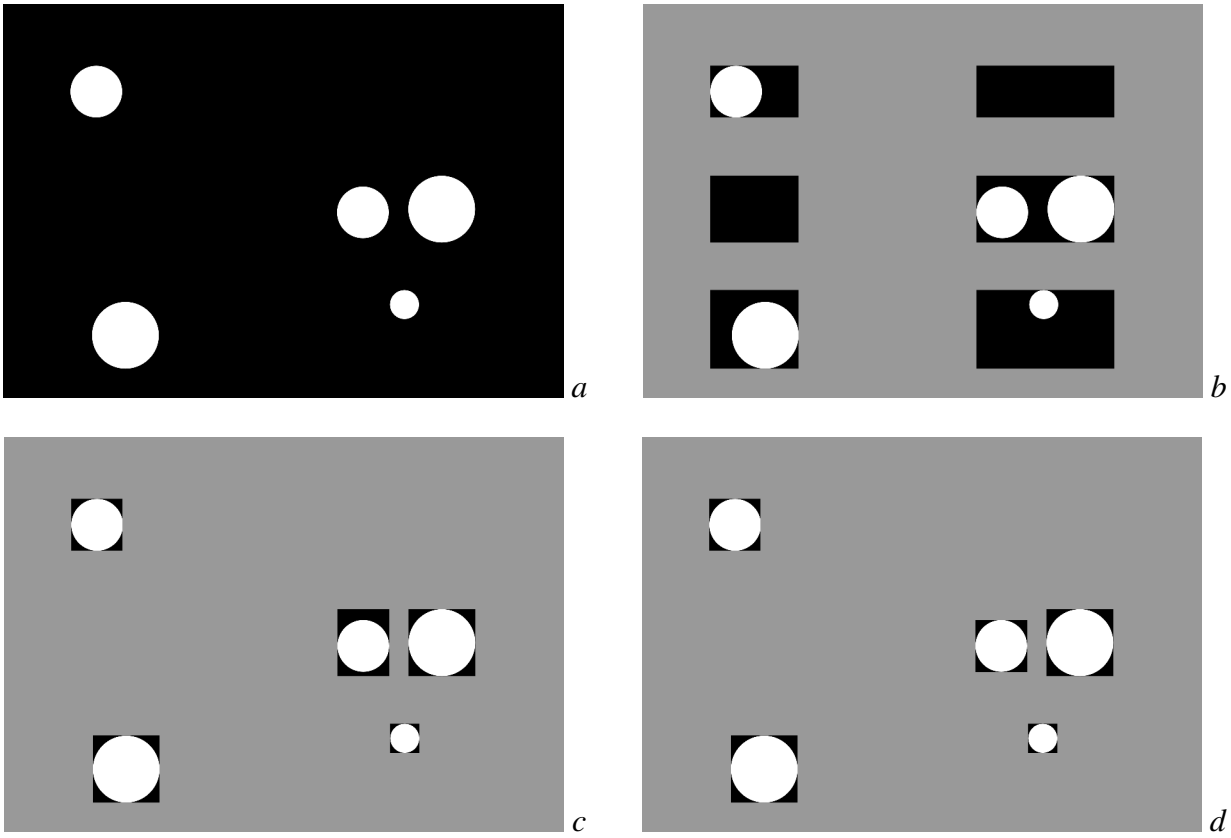
3.1 Lokalizacja cząsteczek

Proponowany tutaj program `lokalizacja.m` (rozdz. 5.1.1) pobiera obrazek `im` a na wyjściu daje wektory X i Y, które zawierają odpowiednio współrzędne poziome i pionowe cząsteczek. Należy zwrócić uwagę na wiersze 7 i 10, w których zaczerniane są obszary z defektami, które mogłyby wprowadzać błędy. Należy dostosować parametry w tych liniach do specyfiki eksperymentu.

W liniach 13-29 fotografie są jeszcze filtrowane. Przez zastosowanie splotu z gausowskim kernelem wzmacniane są obrazy cząsteczek. Zakres wielkości wzmacnianych cząsteczek jest ustawiany w liniach 20-21. Te parametry należy również dostosować do konkretnego eksperymentu.

W wierszu 32 tworzona jest maska przez odpowiednie progowanie. Wielkości tego progowania należy również dobrać w eksperymencie. Maska zawiera jedynki w miejscach gdzie znajdują się obrazy cząsteczek zero w pozostałym obszarze.

Należy zwrócić uwagę, że sama lokalizacja pikseli zawierających jedynki nie daje nam pozycji cząsteczek. Zazwyczaj jedna cząsteczka składa się z wielu pikseli. Należy więc znaleźć grupy pikseli tworzące pojedyncze cząsteczki. Problem ten rozwiązuje program `wyc.m` (rozdz. 5.1.2) wywoływany w lini 36. Program ten wymazuje z macierzy maski wszystkie wiersze i kolumny, które nie zawierają jedynek. Pozostają tylko prostokątne segmenty. Następnie sprawdzane jest czy te segmenty nie zawierają też pustych wierszy albo kolumn i ewentualnie rozbijane są na coraz mniejsze segmenty. Po zakończeniu tej operacji pojedyncze segmenty zawierać będą tylko pojedyncze cząsteczki. Można zatem zlokalizować cząsteczki znając współrzędne segmentów. Algorytm ten jest szybki, chociaż może nie działać dobrze w przypadku gdy cząsteczki stykają się, ale taką ewentualność wykluczyliśmy wcześniej. Działanie jego zilustrowane jest na rys. 5. Należy zwrócić uwagę, że liczba kroków algorytmu nie zależy od ilości cząsteczek na fotografii, ale od ich ułożenia. Jest on więc bardzo wydajny dla fotografii na których mamy dużo cząsteczek.

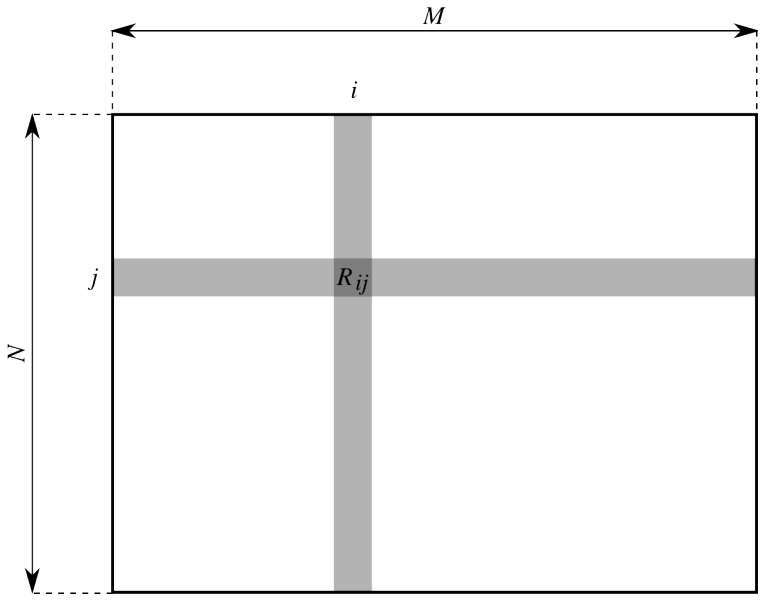


Rys. 5: Przykład działania algorytmu lokalizującego cząstki dla macierzy maski; a – maska; b, c, d – kolejne kroki algorytmu. W pierwszym kroku algorytm wymazuje z maski wszystkie puste wiersze i kolumny, w ten sposób otrzymujemy sześć segmentów (b). W kolejnym kroku każdy z segmentów analizowany jest osobno. Dzięki temu usuwamy puste segmenty, dokonujemy ewentualnego podziału segmentów na mniejsze i dopasowujemy lepiej rozmiar segmentów do znajdujących się w nich cząsteczek (c). Operację tę powtarzamy dla otrzymanych segmentów do czasu kiedy nie znajdujemy już pustych wierzy ani kolumn (d). Następnie środki cząsteczek mogą być oszacowane przez środki segmentów.

3.2 Porządkowanie zbiorów położeń

Wcześniej omówiliśmy wykrywanie cząsteczek i pomiar ich położeń. W wyniku tej operacji otrzymujemy zbiór położeń dla zbioru cząsteczek widocznych na fotografii. Dla dwóch fotografii otrzymujemy dwa zbiory. Aby znaleźć przemieszczenia cząsteczek potrzebne jest przyporządkowanie sobie elementów z obu zbiorów. Idea algorytmu została omówiona wcześniej. Jak można go zrealizować w praktyce? Oznaczmy indeksem *i* elementy zbioru położeń cząsteczek z pierwszej fotografii. Dla drugiej fotografii wybierzmy indeks *j*. Niech *M* i *N* będą liczbami określającymi rozmiar pierwszego i drugiego zbioru (Należy zwrócić uwagę, że oba zbiory nie muszą mieć takiej samej liczby elementów. Jest to spowodowane tym, że cząsteczki mogą uciec z pola widzenia lub zostać „niezauważone" przez program lokalizujący). Niech dalej x_i i y_i oznaczają parę współrzędnych w zbiorze pierwszym, natomiast x'_j i y'_j parę współrzędnych w zbiorze drugim. Zdefiniujmy na ich podstawie następującą macierz:

$$R_{ij}=\sqrt{(x_i-x'_j)^2+(y_i-y'_j)^2} \tag{4}$$



Rys. 6: Macierz R. Znalezienie najmniejszego elementu R_{ij} pozwala na przyporządkowanie sobie elementów ze zbiorów położeń. i-temu elementowi z pierwszego zbioru przyporządkowany jest j-ty element z drugiego zbioru położeń. Następnie, aby elementy te nie były brane w następnym kroku pod uwagę i-ta kolumna i j-ty rząd macierzy wypełniane są dużymi liczbami. Element macierzy R_{ij} należy interpretować jako odległość pomiędzy i-tą pozycją ze zbioru pierwszego a j-tą pozycją ze zbioru drugiego, i-ty wiersz macierzy R należy interpretować jako zbiór odległości pomiędzy pozycją i-tej cząsteczki z pierwszej fotografii a wszystkimi pozycjami z drugiej fotografii.

Macierz *R* zawiera odległości pomiędzy każdym elementem ze zbioru pierwszego a każdym elementem ze zbioru drugiego. Element $R_{i,j}$ macierzy to odległość pomiędzy *i*-tym położeniem ze zbioru pierwszego a *j*-tym położeniem ze zbioru drugiego. Macierz taka jest wykorzystywana w

programie `trajektorie_ba2.m` (rozdz. 5.2.1, wiersz 43). Położenia są przyporządkowywane w pętli (wiersze 52-69). W każdym kroku pętli znajdowany jest najmniejszy element macierzy R (wiersz 53). Na podstawie położenia tego elementu przyporządkowujemy parę położeń. Następnie cały wiersz i cała kolumna w których znajduje się najmniejszy element jest oznaczany przez wpisanie do nich dużej wartości. W ten sposób nie będą one brane już pod uwagę w następnym kroku. Co oznacza duża wartość? Jest to graniczna wartość przemieszczenia. Spodziewamy się, że przemieszczenia będą mniejsze od pewnej wielkości. Wielkość ta jest zależna od eksperymentu. W omawianym tu programie graniczna wartość przemieszczenia jest zapisana w zmiennej `maxmov`. W wierszu 52 pętla zaczyna się od warunku, w którym sprawdzamy czy macierz R zawiera jakieś elementy nie większe od `maxmov`. W każdym kroku pętli po znalezieniu najmniejszego elementu, który musi być jednocześnie mniejszy od `maxmov` zapełniamy wiersz i kolumnę zawierającą dany element przez wartość `maxmov+1` (wiersze 66-67). W ten sposób liczba elementów niewiększych od `maxmov` sukcesywnie maleje. Pętla kończy się więc gdy w macierzy R nie znajdują się już żadne elementy niewiększe od `maxmov`. Część położeń nie zostanie więc w wyniku takiej operacji uporządkowana w pary, bo takie uporządkowanie prowadziłoby do pomiaru niefizycznych przemieszczeń. Jest to spowodowane tym, że część cząsteczek widocznych na pierwszej fotografii jest niewidoczna na drugiej i odwrotnie.

Program `trajektorie_ba2.m` wykonuje operację dla całej serii danych. Wczytuje on kolejno pliki z danymi o położeniach, które zostały przygotowane wcześniej (ścieżka jest podawana przy wywoływaniu programu w zmiennej `path`) i analizuje każdą następującą po sobie parę plików (1 i 2, 2 i 3, 3 i 4, 4 i 5,...). W ten sposób otrzymujemy ciągi położeń opisujące trajektorie cząsteczek. Wynikowe macierze to zbiory tych ciągów. Każda kolumna macierzy to jeden ciąg odpowiadający jednej cząsteczce. Przy takiej analizie pojawia się problem urywających się ciągów, gdy cząsteczka znika. Ciągi takie są przez program wyrzucane.

4 Podsumowanie i krótka instrukcja

Niniejsza praca nie jest instrukcją obsługi programów napisanych na potrzeby pomiaru trajektorii cząsteczek. Stanowi raczej szkicowe omówienie metody tego pomiaru. Autor zdaje sobie sprawę, że programy dołączone do pracy są w dużej mierze niedoskonałe. Napisane zostały szybko na potrzeby jednego eksperymentu i niestety bez dbałości o wygodę użytkownika. Zastosowanie ich do ciągu fotografii pokazuje jednak skuteczność opisanej wyżej metody.

Cała procedura pomiaru z wykorzystaniem opisanego tutaj oprogramowania powinna wyglądać następująco. No początku z wykorzystaniem programu `lokalizacja.m` należy zmierzyć położenia cząsteczek na serii fotografii. Dla obrazów zgromadzonych w danym katalogu operację tę można przeprowadzić z wykorzystaniem przykładowego skryptu:

```
1. path='/media/sda4/praca/silver-nanoparticles-50x-2';
2.
3. obrazek=dir([path,'/*.bmp']);
4.
5. for i=1:length(obrazek)
```

```
6.     i
7.     nazwa=obrazek(i).name;
8.     im=imread([path,'/',nazwa]);
9.     [X,Y]=lokalizacja(im);
10.
11.     eval(['save ',path,'/',nazwa(1:end-4),' X Y -V6']);
12.end
```

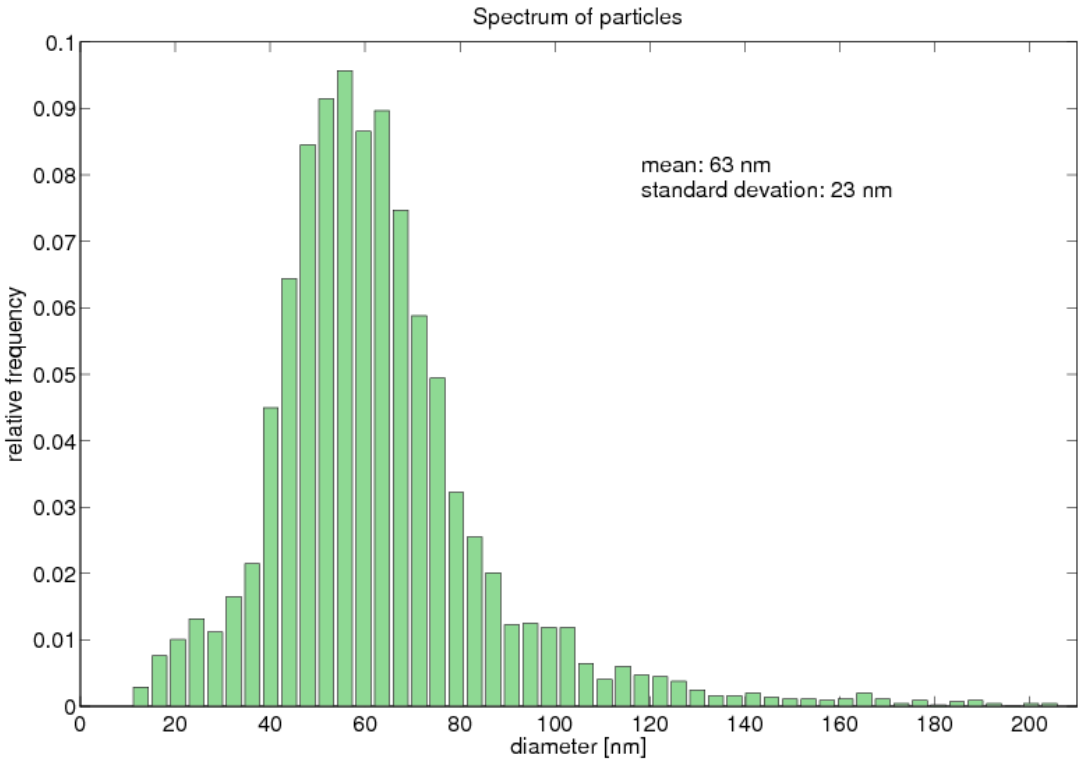
Zmiennej `path` przyporządkowana zostaje ścieżka do katalogu z obrazkami. Każda fotografia jest wczytywana w pętli (wiersz 8) i stosowany jest do niej program `lokalizacja.m` (wiersz 10). Następnie wynik (czyli wektory `X` i `Y` zawierające położenia cząsteczek) jest zapisywany w postaci binarnej w formacie MATLABA. Należy pamiętać o wcześniejszym dostosowaniu programu `lokalizacja.m` do warunków eksperymentu przez odpowiedni dobór parametrów.

Następnym krokiem jest wykonanie programu `trajektorie_ba2.m`. Przy wywoływaniu tego programu należy podać ścieżkę do katalogu, w którym uprzednio umieściliśmy pliki z pozycjami cząsteczek na poszczególnych fotografiach. Otrzymane w ten sposób trajektorie należy przeliczyć z pikseli na rzeczywiste wymiary.

Zamiast programu `trajektorie_ba2.m` można wywołać również jego zmodyfikowaną wersję - program `policz_srednice.m` (rozdz. 5.2.2). Program ten nie liczy trajektorii dla całej serii fotografii, ale dla krótszych serii. Zaletą tego jest to, że przy krótszych trajektoriach pozostaje więcej cząsteczek które nie znikają. Dzięki temu możemy zmierzyć trajektorie dla większej liczby cząsteczek co pozwala na uzyskanie większej próbki do analizy statystycznej. Ponadto pomiar krótszych trajektorii może być wykonywany wielokrotnie gdy za każdym razem serię zaczynamy od innego pliku z danymi o pozycji cząsteczek. Ponieważ dla każdej skróconej serii pomiarów rozważana jest ta sama próbka cząsteczek, uzyskane w ten sposób średnice cząsteczek można połączyć w jedną próbę statystyczną i w ten sposób podnieść dokładność analizy.

Przykłady trajektorii znalezionych w ten sposób prezentowane są tutaj: http://fluid.ippt.gov.pl/pkor/webpage/laboratory/brown motion/.

Następnie, znając odpowiednie parametry fizyczne przy pomocy wzorów 1-3 można znaleźć na podstawie trajektorii średnice cząsteczek. Dla zbioru średnic można stworzyć histogram dla rozkładu wielkości cząsteczek (patrz rys. 7).



Rys. 7: Przykład widma wielkości cząsteczek zmierzonych na podstawie fotografii z próbnego eksperymentu.

5 Kompletne skrypty

Wszystkie omawiane tu programy napisane w środowisku MATLAB są dostępne tutaj: *http://fluid.ippt.gov.pl/pkor/webpage/laboratory/brown%20motion/matlab_progs.zip*

5.1 Wykrywanie cząsteczek i znajdowanie ich położeń

5.1.1 lokalizacja.m

```
1. function [X,Y]=lokalizacja(im)
2.
3. % funkcja wyrzuca wspolrzedne czasteczek znalezionych na obrazku im
4.
5. im=double(im);
6.
7. im(1:10,:)=0; % wyrzucamy naglowek
8.
9. %wyrzucamy plamke - staly defekt fotografii
10.im(752:828,540:622)=0;
11.
12.
13.% filtrujemy obrazy przez zastosowanie operacji splotu z gausowskimi
   kernelami
```

```
14.
15.%tworzymy kernel
16.x=[-3:3];
17.x=x(ones(7,1),:);
18.r=sqrt(x.^2+x'.^2);
19.
20.sigma1=.7;
21.sigma2=2;
22.
23.g1=exp(-(r.^2)/(2*sigma1^2));
24.
25.g2=exp(-(r.^2)/(2*sigma2^2));
26.
27.% liczymy sploty
28.kor1=kern_kor(im,g1);
29.kor2=kern_kor(im,g2);
30.
31.% mask jest obrazem po progowaniu - jedynki odpowiadaja obrazom czasteczek
32.mask=(((kor2>.5)|(kor1>.5)).*(im>60));
33.
34.
35.% lokalizujemy poszczegolne czasteczki
36.[X1,Y1,X2,Y2]=wyc(mask);
37.
38.X=(X1+X2)/2;
39.Y=(Y1+Y2)/2;
```

5.1.2 wyc.m

```
1. function [X1,Y1,X2,Y2]=wyc(im)
2.
3. %funkcja przez odpowiednie progowanie w funkcji wycieranie2
4. %pozwala zgrugsza oszacowac polozenia kropelek
5.
6. im=double(im);
7.
8. global X1 Y1 X2 Y2
9. X1=[];
10.Y1=[];
11.X2=[];
12.Y2=[];
13.
14.SX=[];
15.SY=[];
16.R=[];
17.%Rp=[];
18.%M=[];
19.
20.[n,m]=size(im);
21.
22.imw=wycieranie2(im,1,1,m,n);
23.
24.
25.function imw=wycieranie2(ims,x1,y1,x2,y2)
26.
27.% funkcja pobiera obrazek zawierajacy kropelki
28.% wyjsciowy obrazek zawiera kwadraty ograniczone do kropelek
29.global X1 X2 Y1 Y2
```

```
30.
31.im=double( ims(y1:y2,x1:x2) );
32.
33.
34.%mask=im<120;
35.mask=im;
36.
37.%sprawdzamy, ktore linie nie zawieraja jedynek
38.t1=sum(mask)>0;
39.t2=sum(mask,2)>0;
40.
41.
42.%sprawdzamy czy segment nie pokrywa sie z calym obrazkiem
43.
44.if ( (mean(t1)==1) && (mean(t2)==1) )
45.   %zapisujemy polozenia segmentow, ktorych nie da sie juz zmniejszyc
46.    X1=[X1,x1];
47.    X2=[X2,x2];
48.    Y1=[Y1,y1];
49.    Y2=[Y2,y2];
50.
51.else
52.
53.
54.    %ustalamy pozycje wszystkich segmentow
55.
56.    %zaznaczamy punkty, ktore sa poczatkami segmentow
57.    %(poczatek segmentu ma po lewej stronie sasiada, ktory nie nalezy do
58.    %segmentu)
59.    start1=[t1(1),t1(2:end)&(~t1(1:end-1))];
60.    start2=[t2(1),(t2(2:end)&(~t2(1:end-1)))'];
61.
62.    %zaznaczamy punkty, ktore sa koncami segmentow
63.    stop1=[t1(1:end-1)&(~t1(2:end)),t1(end)];
64.    stop2=[(t2(1:end-1)&(~t2(2:end)))',t2(end)];
65.
66.    %znajdujemy wspolrzedne segmentow
67.    start1=x1+find(start1)-1;
68.    start2=y1+find(start2)-1;
69.    stop1=x1+find(stop1)-1;
70.    stop2=y1+find(stop2)-1;
71.
72.    %przeszukujemy wszystkie segmenty
73.    for i=1:length(start1)
74.
75.        for j=1:length(start2)
76.
77.            ims=wycieranie2(ims,start1(i),start2(j),stop1(i),stop2(j));
78.
79.        end
80.
81.    end
82.
83.
84.
85.end
86.
87.imw=ims;
```

5.1.3 kern_kor.m

```
function kor=kern_kor(im1,im2)

tic
im1=double(im1(:,:,1));
im2=double(im2(:,:,1));

%dla obrazu im liczymy lokalnie wspolczynnik korelacji przy uzyciu kernela
%im2
%

%okreslamy rozmiar kernela
[a1,a2]=size(im2);

wer=im2;
wer=wer-mean(wer(:)); %odjecie sredniej

%poszerzenie probki do wymiarow okna przeszukiwan
w1s=zeros(size(im1));
w1s(1:a1,1:a2)=wer;

%w2=kwadrat(im2,a+2*rmax,x,y);
w2=im1;

f1=fft2(w1s);
f2=fft2(w2);

%licznik
kor=real((ifft2(f2.*conj(f1))));

%mianownik
mi1=sum(sum(wer.^2));

mask=zeros(size(im1));
mask(1:a1,1:a2)=1;
mm=conj(fft2(mask));
su2k=real(ifft2(fft2(w2.^2).*mm));
su2=real(ifft2(f2.*mm));

mi2=(su2k - (su2.^2)/(a1*a2));
mi2=mi2.*(mi2>=0);
%sum(sum(mi2<0))
mi=sqrt(mi1*mi2);
mi0=(mi==0); %sprawdzamy gdzie mianownik osiaga zero

%laczenie
kor=(kor./(mi0+mi)).*(~mi0);%uwazamy zeby nie bylo dzielenia przez zero
kor=kor(1:end-a1+1,1:end-a2+1);

% dodajemy zera na brzegach, żeby miec odpowiedni rozmiar
KOR=zeros(size(im1));
KOR((a1-1)/2+1:end-(a1-1)/2,(a2-1)/2+1:end-(a2-1)/2)=kor;
kor=KOR;

toc
```

5.2 Porządkowanie cząsteczek

5.2.1 trajektorie_ba2.m

```
function [x,y]=trajektorie_ba2(path)

dane=dir([path,'/*.mat']);

load( [path,'/',dane(1).name] );
x=[X];
y=[Y];

size(x)%

length(dane);

for i=2 :length(dane)
    %i
    X=[];
    Y=[];
    dane(i).name
    load( [path,'/',dane(i).name] );
    X1=x(end,:);
    Y1=y(end,:);

    X2=X;
    Y2=Y;

    maxmov=20; %maksymalne przesuniecie czastek

    %size(X1)
    %size(X2)

    L1=length(X1);
    L2=length(X2);

    XX1=X1(ones(1,L2),:);
    YY1=Y1(ones(1,L2),:);

    XX2=X2(ones(L1,1),:)';
    YY2=Y2(ones(L1,1),:)';

    %size(XX1)
    %size(XX2)

    % obliczamy dlugosc wszystkich potencjalnych przemieszczen
    R=sqrt( (XX1-XX2).^2 + (YY1-YY2).^2 );

    x1=[];
    y1=[];
    x2=[];
    y2=[];
    sx=[];
    sy=[];

    while sum(R(:)<=maxmov)>0
        [n,m]=find(R==min(R(:)));
        %wyjmujemy wlasciwa kolumne z macierzy danych
```

```
55.         % i dodajemy nastepny punkt
56.         sx=[x(:,m(1));X2(n(1))];
57.         sy=[y(:,m(1));Y2(n(1))];
58.
59.         %tworzymy z wybranych kolumn nowa macierz danych
60.         x1=[x1,sx];
61.         y1=[y1,sy];
62.
63.
64.         %size(x1)
65.
66.         R(:,m(1))=maxmov+1;
67.         R(n(1),:)=maxmov+1;
68.
69.     end
70.
71.     x=x1;
72.     y=y1;
73.
74.end
```

5.2.2 policz_srednice.m

```
75.function policz_srednice
76.
77.path='/media/sda4/praca/silver-nanoparticles-50x-2';
78.
79.% path - sciezka do plikow z pozycjami czasteczek
80.
81.% program wczytuje pliki z pozycjami czasteczek w seriach po 100 plikow
82.% i laczy je w trajektorie.
83.% kazda seria zaczyna sie od pliku o numerze ty , gdzie ty jest w petli
   zmieniane co 10.
84.% dzieki temu otrzymujemy trajektorie tych samych czasteczek liczone dla
   roznych czasow startowych ty.
85.% trajektorie sa krotsze, ale za to jest ich wiecej, bo mniej czasteczek
   "ucieka" dla 100 fotografii
86.% niz gdybysmy przeprowadzili te opearacje dla calej serii.
87.
88.
89.dt=0.02; %interwal pomiedzy dwiema fotografiami
90.mu=8.9*(10^-4); %lepkosc plynu
91.k=1.38*10^(-23); %stala Boltzmana
92.T=(27+273); %temperatura plynu
93.lh=(300*10^-6)/1234; %skala [m/pixel]
94.
95.
96.
97.
98.dane=dir([path,'/*.mat']);
99.
100.
101.
102.
103.for ty=1:10:(length(dane)-100)
104.
105.
106.
```

```
107.
108.
109.    load( [path,'/',dane(ty).name] );
110.    x=[X];
111.    y=[Y];
112.
113.    %size(x)
114.
115.    length(dane);
116.
117.    for i=ty+1 : ty+100 %length(dane)
118.        %i
119.        X=[];
120.        Y=[];
121.        dane(i).name
122.        load( [path,'/',dane(i).name] );
123.        %dane(i).name
124.        X1=x(end,:);
125.        Y1=y(end,:);
126.
127.        X2=X;
128.        Y2=Y;
129.
130.        maxmov=20; %maksymalne przesuniecie czastek
131.
132.        %size(X1)
133.        %size(X2)
134.
135.        L1=length(X1);
136.        L2=length(X2);
137.
138.        XX1=X1(ones(1,L2),:);
139.        YY1=Y1(ones(1,L2),:);
140.
141.        XX2=X2(ones(L1,1),:)';
142.        YY2=Y2(ones(L1,1),:)';
143.
144.        %size(XX1)
145.        %size(XX2)
146.
147.        % obliczamy dlugosc wszystkich potencjalnych przemieszczen
148.        R=sqrt( (XX1-XX2).^2 + (YY1-YY2).^2 );
149.
150.        x1=[];
151.        y1=[];
152.        x2=[];
153.        y2=[];
154.        sx=[];
155.        sy=[];
156.
157.        while sum(R(:)<=maxmov)>0
158.            %   sum(R(:)<=maxmov)
159.            [n,m]=find(R==min(R(:)));
160.            %wyjmujemy wlasciwa kolumne z macierzy danych
161.            % i dodajemy nastepny punkt
162.            %   size(x(:,m(1)))
163.            %  size(X2(n(1)))
164.            sx=[x(:,m(1));X2(n(1))];
165.            sy=[y(:,m(1));Y2(n(1))];
```

```
166.
167.
168.            %size(sx)
169.
170.            %tworzymy z wybranych kolumn nowa macierz danych
171.            x1=[x1,sx];
172.            y1=[y1,sy];
173.
174.
175.            %size(x1)
176.
177.            R(:,m(1))=maxmov+1;
178.            R(n(1),:)=maxmov+1;
179.
180.
181.
182.        end
183.
184.        x=x1;
185.        y=y1;
186.
187.        %size(x)
188.
189.    end
190.
191.    x=x*lh;
192.    y=y*lh;
193.    size(x)
194.    s2=1.5*( mean( ( x(2:end,:)-x(1:end-1,:)).^2) + mean((y(2:end,:)-
  y(1:end-1,:)).^2));
195.    size(s2)
196.
197.    D=s2/(2*dt);
198.    size(D)
199.    dp=(D.^-1)*(k*T)/(3*pi*mu);
200.    dp=dp*(10^9); % przeliczamy na nanometry
201.
202.    name=['dia',num2str(ty)];
203.    eval(['save ',name,' dp -V6']);
204.
205.
206.end
```