Methoden moderner Röntgenphysik: Streuung und Abbildung

Lecture 14	Vorlesung zum Haupt- oder Masterstudiengang Physik, SoSe 2019		
	G. Grübel, F. Lehmkühler, L. Müller, O. Seeck, L. Frenzel, M. Martins, W. Wurth		
Location	Lecture hall AP, Physics, Jungiusstraße		
Date	Tuesday	12:30 - 14:00	(starting 2.4.)
	Thursday	8:30 - 10:00	(until 11.7.)

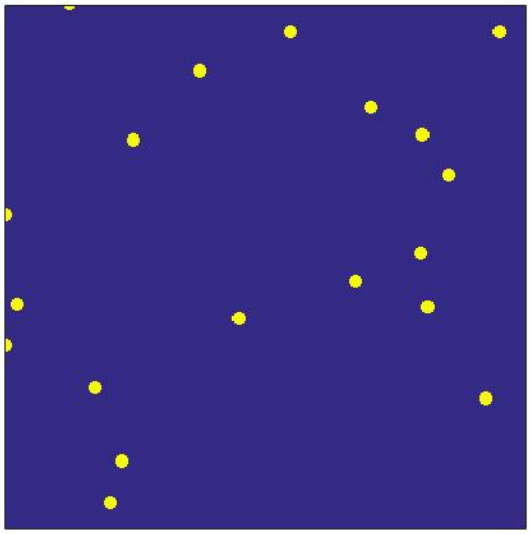
Soft Matter – Timeline

- Di 07.05.2019 Soft Matter studies I: Methods & experiments
  Definitions, complex liquids, colloids, storage ring and FEL experiments, setups, liquid jets, **…**
- Do 09.05.2019 Soft Matter studies II: Structure
  SAXS & WAXS applications, X-ray cross correlations, **…**
- Di 14.05.2019 Soft Matter studies III: Dynamics
  XPCS applications, diffusion, dynamical heterogeneities, ...
- Do 16.05.2019 XPCS and XCCA simulation and modelling
- Di 21.05.2019 Case study I: Glass transition
  Supercooled liquids, glasses vs. crystals, glass transition concepts, structure-dynamics relations, ...
- Do 23.05.2019 Case study II: Water
  Phase diagram, anomalies, crystalline and glassy forms, FEL studies, ...

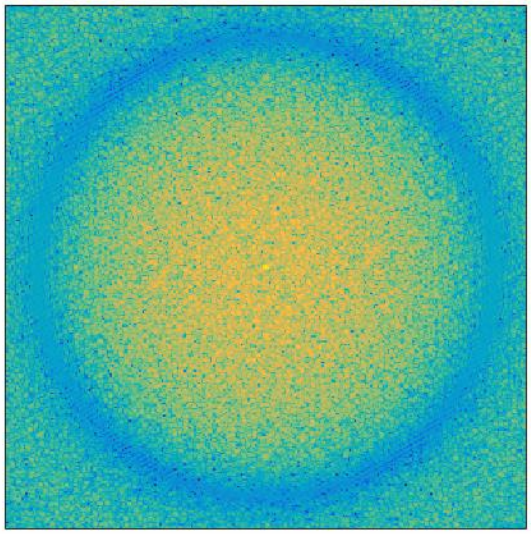
XPCS modelling

Diffusing particles
- Real space particle placement
- Model scattering – SAXS
- Correlation functions

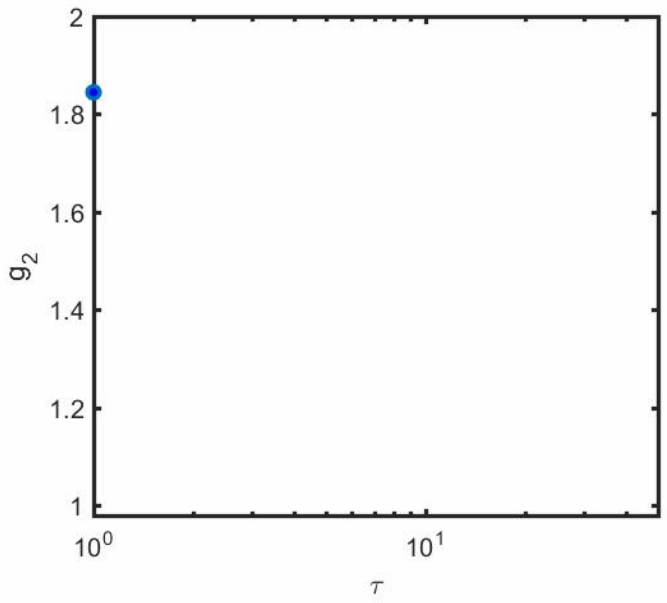
Use Matlab for simulation – many alternatives availble, e.g. Octave, FreeMat, Python



Diffusing particles



Speckle pattern



g_2 function

Matlab basics

Vector and matrices

- `v=[1,2,3]=[1 2 3]` is a linevector
- `w=[1;2;3]` is a row vector
- `A=[1,2;3,4;5,6]` generates 3x2 matrix
- Transpose: `v=w'`
- Scalar: `x=1` as 1x1 vector

Basic algebra

- `A+B, A-B` : Addition/Subtraction of vectors/matrices of same type
- `A*B, A/B` : Matrixproduct/-division
- `A.*B, A./B` : point-wise Product/Division
- `.^` : point-wise exponent

- Example: `A=[1 2;0 4]; B=[-2 1;3 -1].`
  - `A*B=[4 -1;12 -4]` → Matrix-Multiplication
  - `A.*B=[-2 2;0 -4]` → point-wise Multiplication

Matlab basics

Functions

- Lots of basic vector/matrix functions, e.g.: `abs, max, min, size, length, sum, mean, …`
- In addition, pre-defined function such as: `sin, cos, tan, exp, log, log10`, … and many more
- How to use: `y=abs(x)`. x can be a number, vector, matrix etc. for most functions. Attention: operation may be done point-wise or column-wise!
  - Example: `B=[-2 1;3 -1]`.
    - `abs(B)=[2 1;3 1]` → absolute value of each entry
    - `mean(B)=[0.5 0]` → mean of each column
    - `min(B)=[-2 -1]` → minimum of each column

- One can define any type of function
- Information about any function can be obtained by typing „`help FUNCTIONNAME`“, e.g. `help mean`

Matlab basics

Scripts

Typically, scripts are used to work with matlab. Simple example:

```
% This is a comment.
x=[0:1:6]; % define vector x=[0 1 2 3 4 5 6]
y=sin(x); % calculate sinus of x
figure(1) % open a figure
plot(x,y,'o') % plot y vs. x as open circles
```

- Scripts are save as .m file and can be called from the Command Window
- The same script can be written more flexible as function, e.g., to change input parameters

Matlab basics

Write simple functions

```
function y=plot_sin_square(x)

% This functions plots the squared sinus of x. y as defined in
% the function is the output.

y=(sin(x)).^2; % calculate squared sinus of x
figure(1) % open a figure
plot(x,y,'o') % plot y vs. x as open circles
```

- Like scripts, functions are stored as .m file, but input parameters.
- Example: `x=0:0.1:6; out=plot_sin_square(x);`

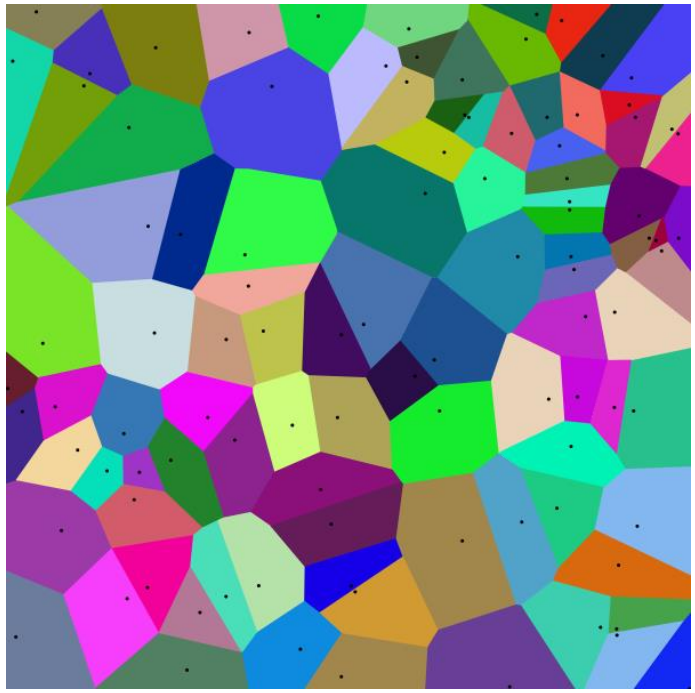
SAXS and XPCS from 2D diffusing discs

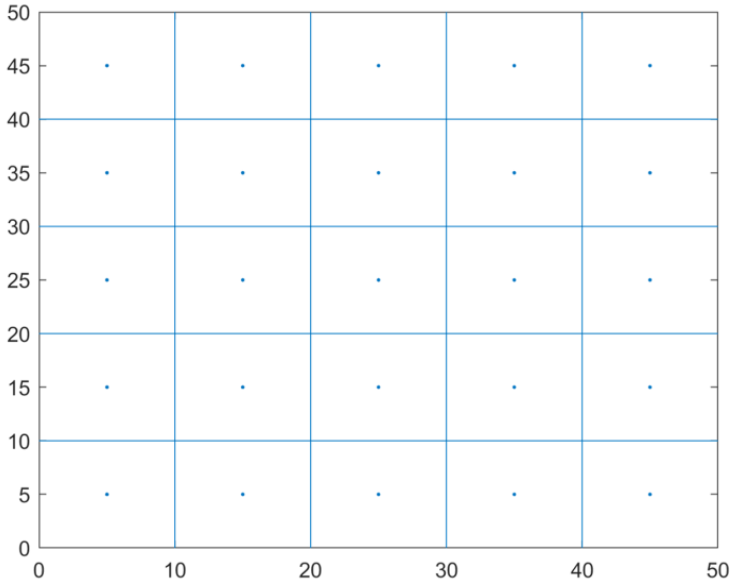
- How to design a 2D system?
- How to simulate diffusion?
- How to simulate scattering?
- How to analyze?

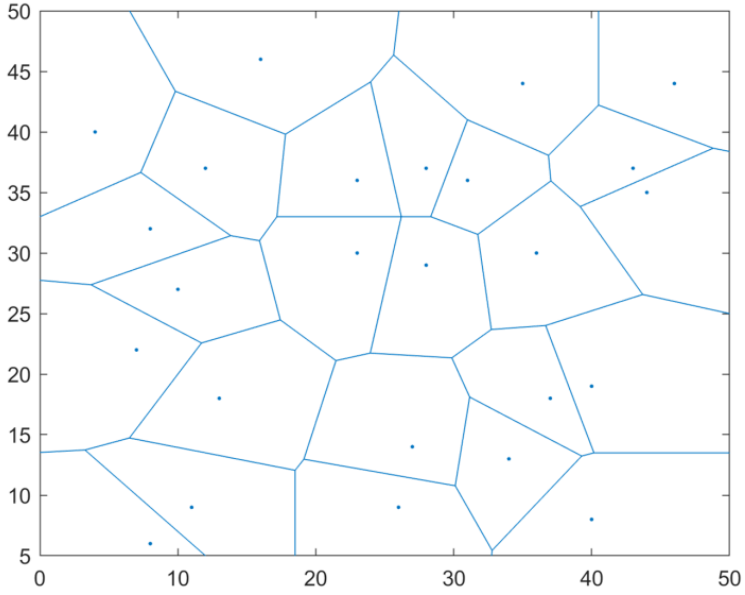
SAXS and XPCS from 2D diffusing discs

Placing particles in 2D

1. Place particles on lattice points
2. Iteratively move particles inside their Voronoi region
3. Repeat some times



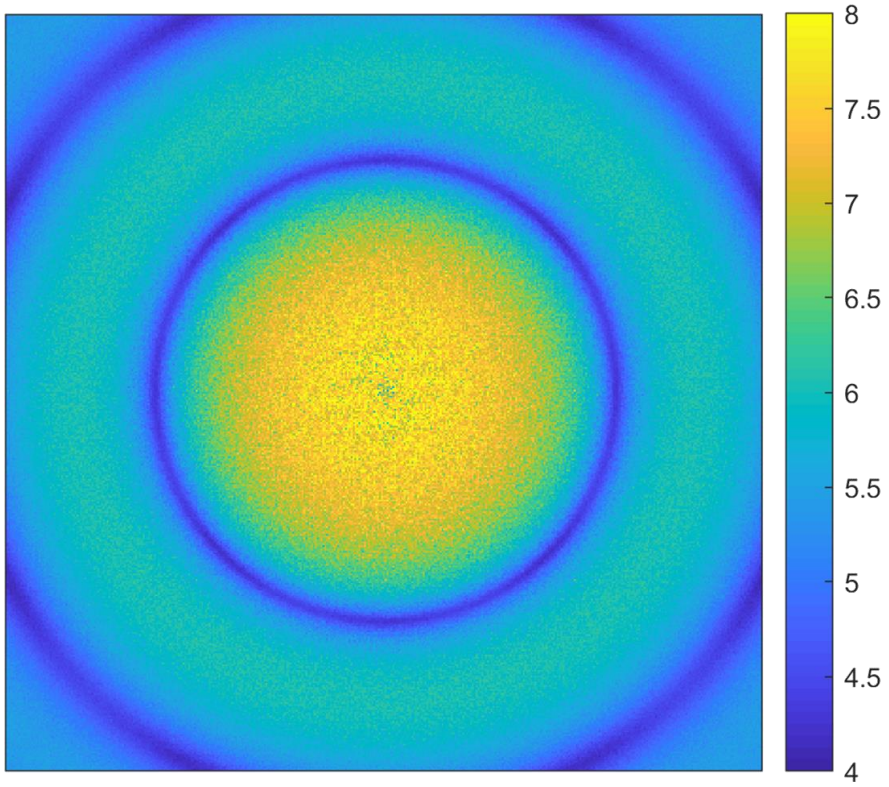




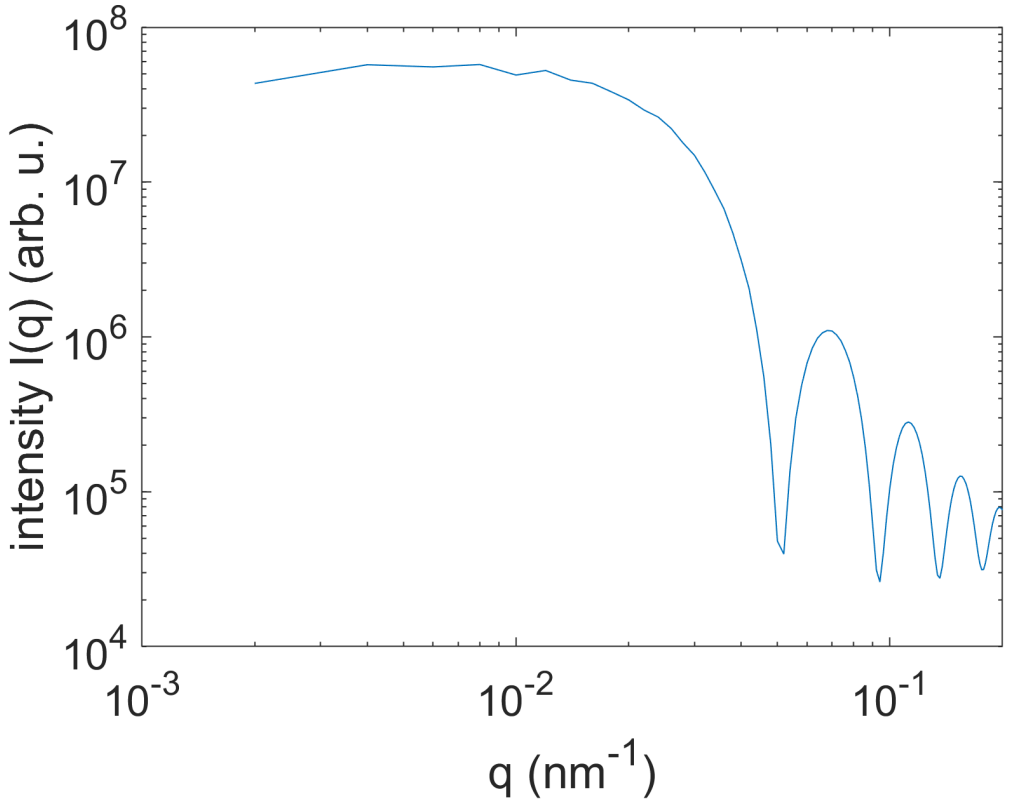
SAXS and XPCS from 2D diffusing discs

Results from xpcs.m

Summed 2D pattern



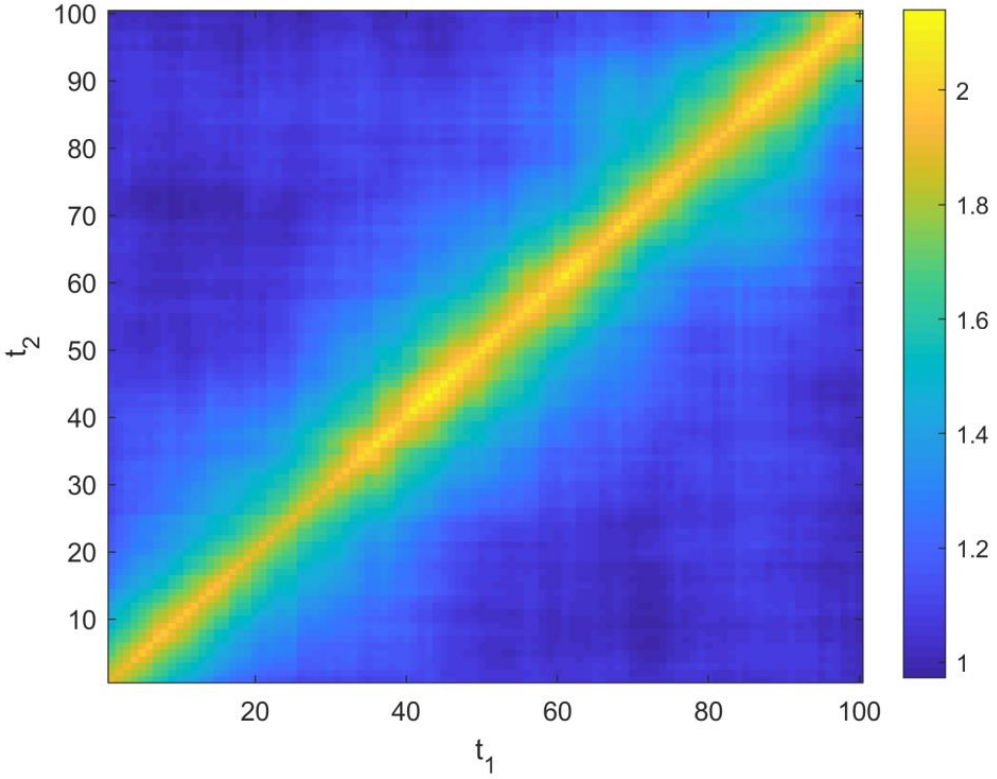
I(q)



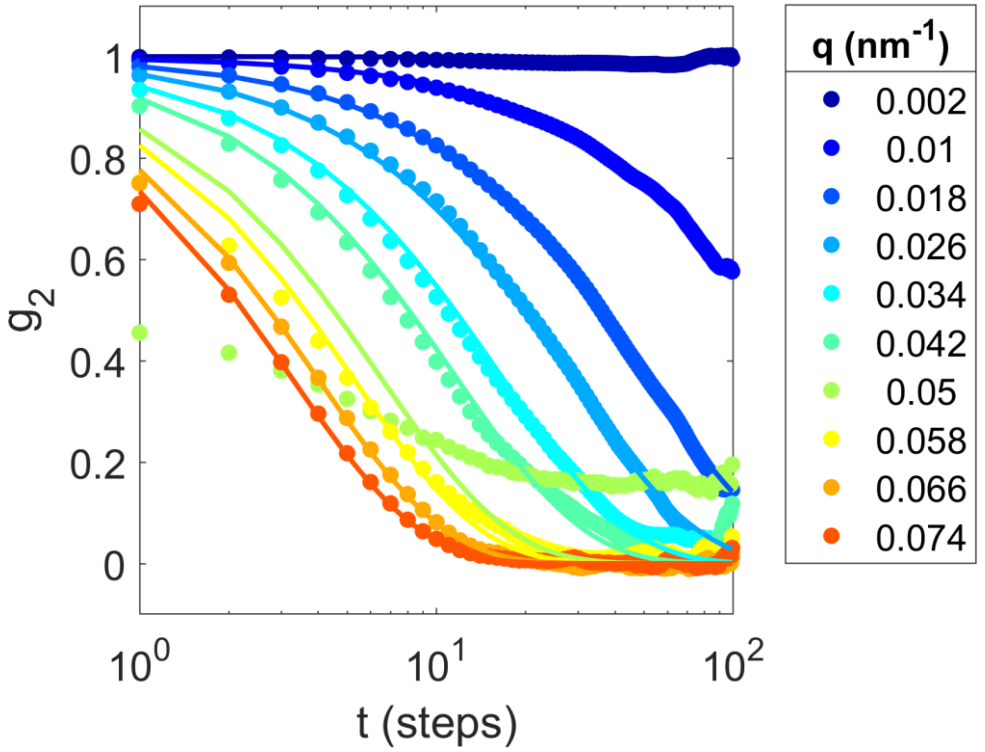
SAXS and XPCS from 2D diffusing discs

Results from xpcs.m

Two time correlation function



g_2 functions



SAXS and XPCS from 2D diffusiong discs

Results from xpcs.m

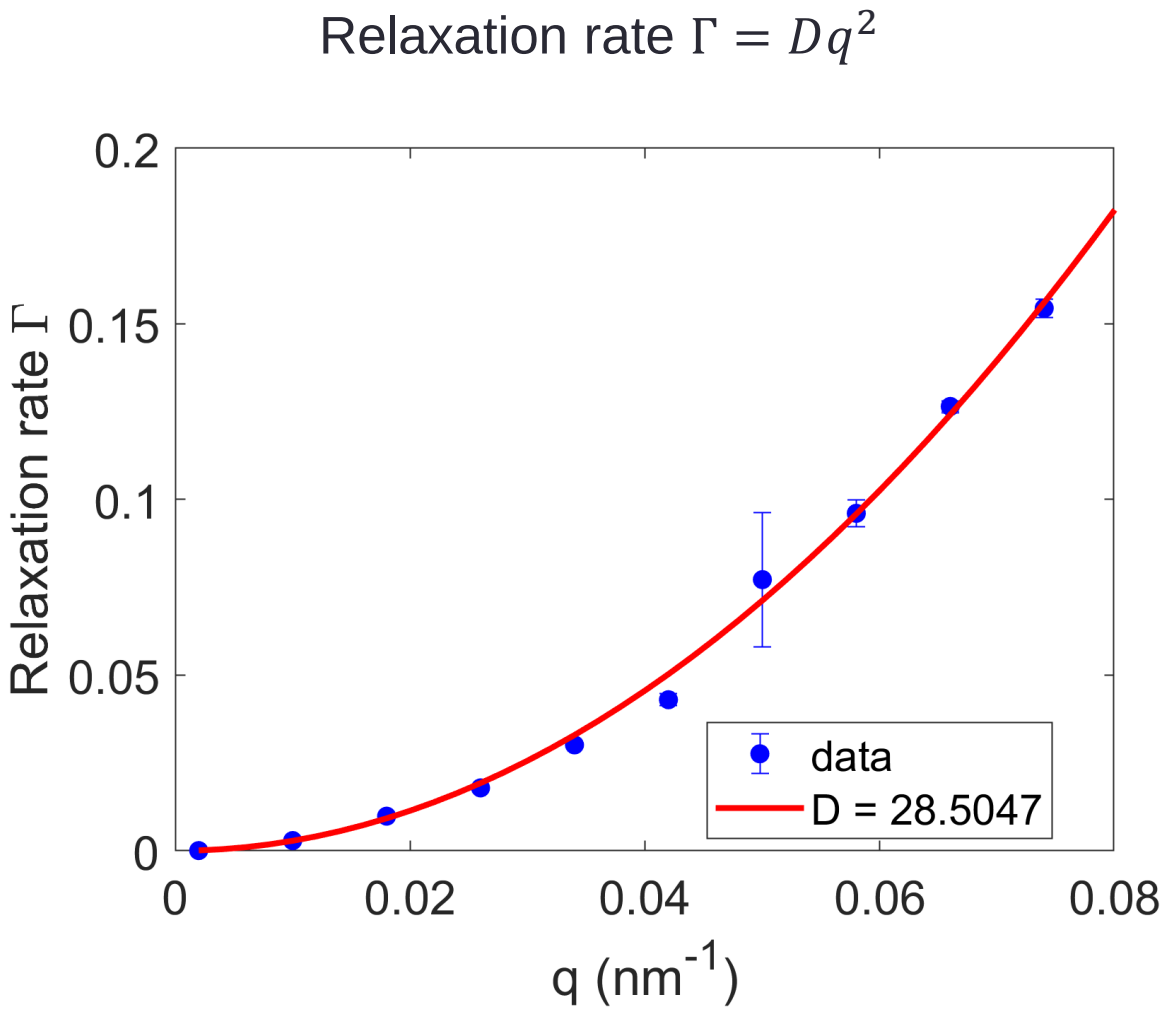


Table of Contents

clean up

```
close all
clear all
```

some parameters

```
matsize=100;    % voronoi parameters
matdist=10;
radius=5;
faktor=10;

steps=100;      % times steps for XPCS
qint=0.002;     % steps for integration
q=0.002:qint:0.2;
qpick=1:4:40;   % indices of q-values for XPCS analysis
```

place particles -- use function voronoiplacing

```
n=voronoiplacing(matsize, matdist, 5,0);
n=n*faktor;
```

define qmap

```
center=[matsize*faktor matsize*faktor]/2+1;     % beam position on
 pattern

pix=50e-6;                                      % pixel size in m
sdd=5;                                          % sample-detector
 distance in m
lambda=1.5e-10;                                 % wavelength in m

[x,y]=meshgrid(1:matsize*faktor,1:matsize*faktor);
r=hypot(x-center(1),y-center(2))*pix;
qmap=4*pi/lambda*sin(atan(r/sdd)/2)*1e-9;       % in nm^-1
```

```
qind=cell(length(q),1);                          % list indices for
 particular q-values
for j=1:length(q)
    qind{j}=find(qmap>=q(j)-qint/2 & qmap<q(j)+qint/2);
end
```

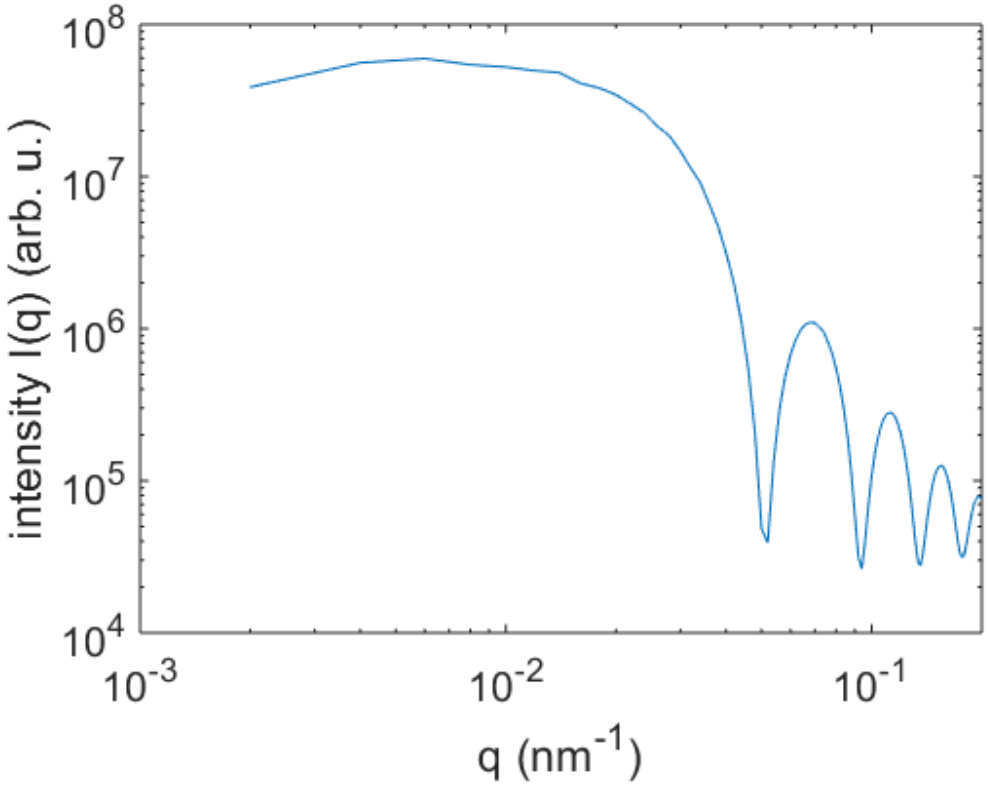
Brownian motion and XPCS

```
imgall=zeros(matsize*faktor,matsize*faktor);    % pre-define summed
 image

for j=1:steps
    patt=zeros(matsize*faktor,matsize*faktor);  % pre-define real
 space pattern

    vec=normrnd(0,radius/10,length(n),2);       % vector defining
 moving steps
    sig=round(rand(length(n),2));
    sig(sig==0)=-1;
    n=n+vec.*sig;

    for k=1:length(n)                           % placing particles of
 type "circ" -- use function getcircle.m
        circ=getcircle(n(k,1),n(k,2),5,matsize*faktor,matsize*faktor);
        patt=patt+circ;
    end

%    figure(1111)
%    imagesc(patt)
%    xlim([300 600])
%    ylim([300 600])
%    title(num2str(j))


    img=abs(fftshift(fft2(patt))).^2;           % "diffraction" --
 absolute square of 2D fft
    imgall=imgall+img;                          % sum up for
 integration

    for k=1:length(qpick)
        inten{qpick(k)}(:,j)=img(qind{qpick(k)});      % intensity at
 q-rings for XPCS
    end
end
```

2D integration: SAXS

```
for jj=1:length(q)
    iq(jj)=mean(imgall(qmap>=q(jj)-qint/2 & qmap<q(jj)+qint/2));
end
```

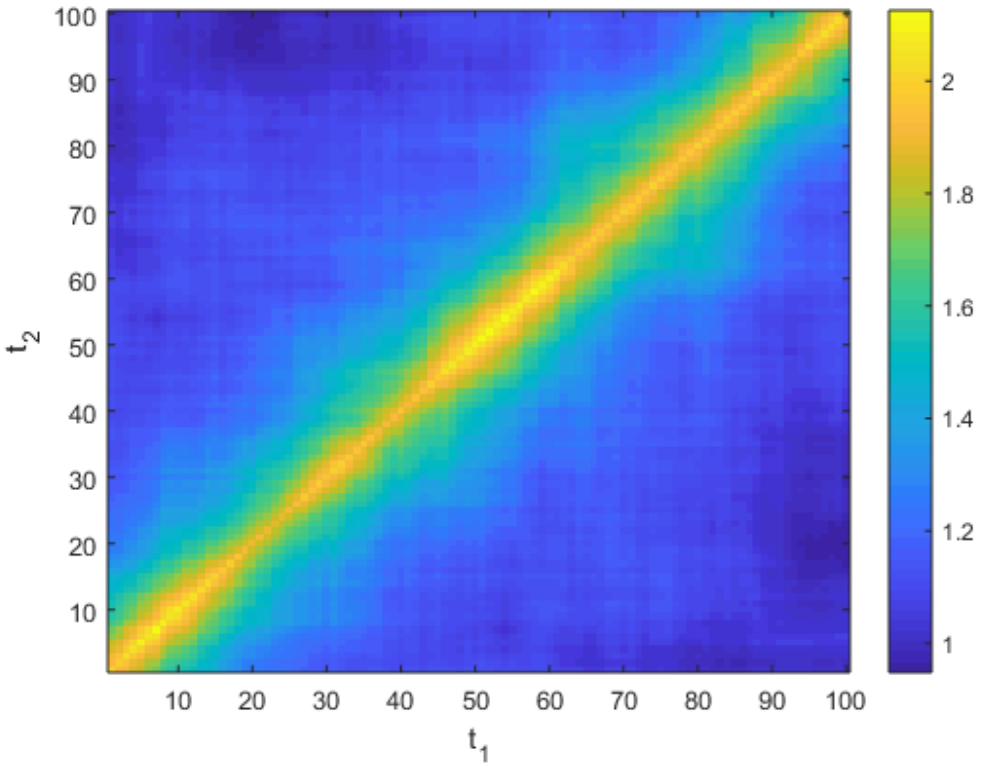
```
figure(2)
loglog(q,iq)
set(gca,'Fontsize',16)
xlabel('q (nm^{-1})')
ylabel('intensity I(q) (arb. u.)')
```



calculate two-time correlations

```
c=zeros(length(qpick),steps,steps);        % pre-define two-time
 correlation function

 for l=1:length(qpick)                      % calculate two-time
 correlation function
     ii=inten{qpick(l)};
     for j=1:steps
         for k=1:steps
             c(l,j,k)=mean(ii(:,j).*ii(:,k)/
(mean(ii(:,j))*mean(ii(:,k))));
         end
     end
 end

figure(3)
imagesc(squeeze(c(5,:,:)))
axis xy
colorbar
```

```
xlabel('t_1')
ylabel('t_2')
```



get g2 from two-time correlations

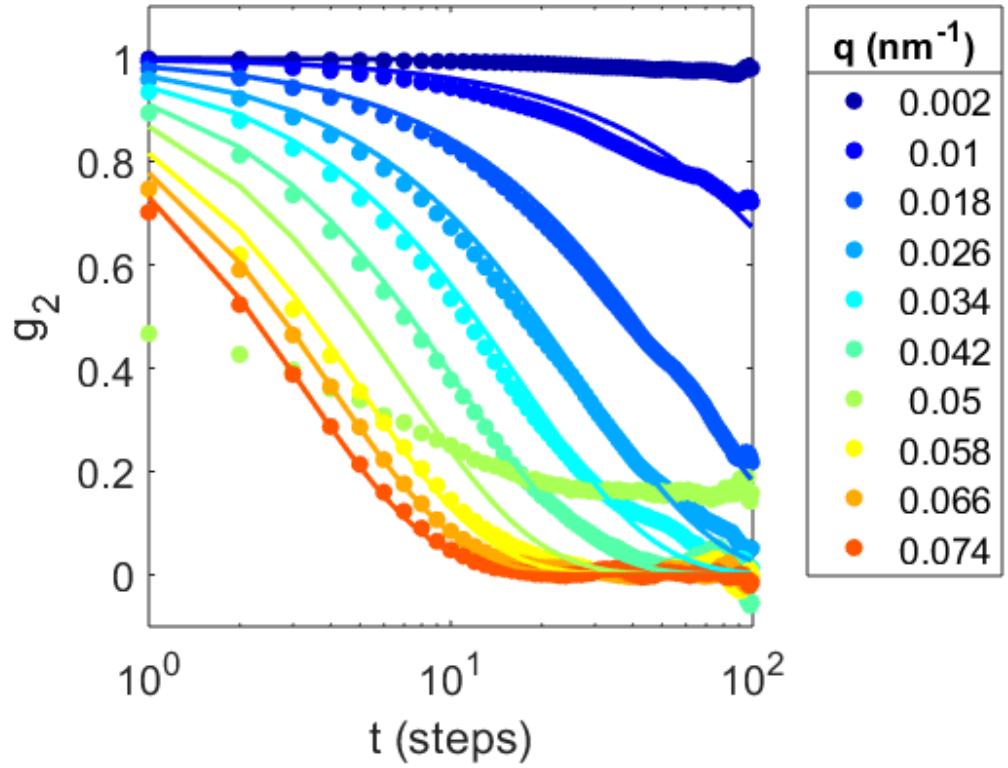
```
g2h=zeros(length(qpick),steps,steps);
for l=1:length(qpick)
    for j=1:steps
        g2h(l,j,1:end-j+1)=c(l,j,j:end);
    end
    g2hh=squeeze(g2h(l,:,:));
    g2(l,:)=sum(g2hh)./fliplr(1:steps);
end

for jj=1:steps
    g2norm(:,jj)=(g2(:,jj)-1)./(g2(:,1)-1);
end
```

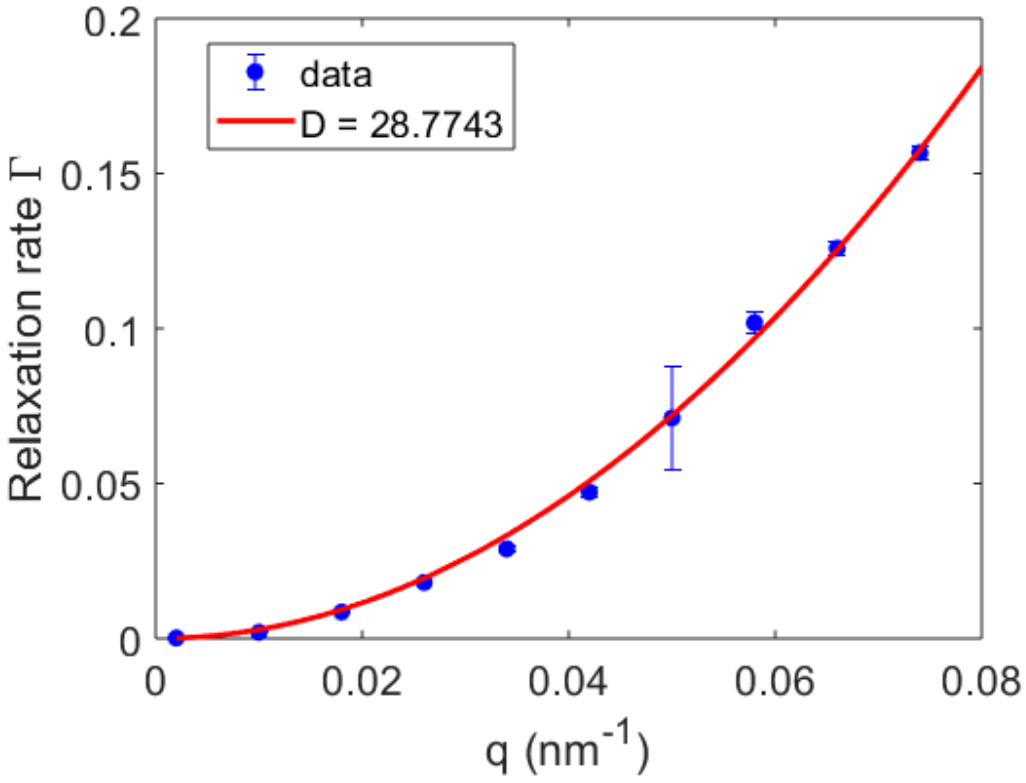
plotting & fitting

```
tau=1:99;                           % time steps
col=jet(length(qpick));             % define some colors
for jj=1:length(qpick)
    g2fit{jj}=fit(tau',g2norm(jj,2:end)','exp(-2*a*x)','StartPoint',
[0.1]);    % perform fit.
```

```
    gamma(jj)=g2fit{jj}.a;
    gamma_err_h=confint(g2fit{jj});
    gamma_err(jj)=gamma_err_h(1)-gamma(jj);

    figure(4)
    hold on

 semilogx(tau,g2norm(jj,2:end),'o','color',col(jj,:),'Markerfacecolor',col(jj,:))
end
for jj=1:length(qpick)
    plot(tau, feval(g2fit{jj},tau),'color',col(jj,:),'linewidth',2)
end

ylim([-0.1 1.1])
set(gca,'Xscale','log','Fontsize',16)
box on
xlabel('t (steps)')
ylabel('g_2')
leg4=legend(num2str(q(qpick)'));
leg4.Location='NorthEastOutside';
title(leg4,'q (nm^{-1})')

gamfit=fit(q(qpick)',gamma','D*x^2');
relrate=feval(gamfit,q);

figure(5)
errorbar(q(qpick),gamma,gamma_err,'ob','markerfacecolor','b')
hold on
plot(q,relrate,'-r','linewidth',2)
set(gca,'Fontsize',16)
xlim([0 0.08])
xlabel('q (nm^{-1})')
ylabel('Relaxation rate \Gamma')
legend('data',['D = ',num2str(gamfit.D)],'Location','best');
```

Warning: Start point not provided, choosing random start point.


q (nm-1)
0.002
0.01
0.018
0.026
0.034
0.042
0.05
0.058
0.066
0.074
g2
1
0.8
0.6
0.4
0.2
0
10^0
10^1
10^2
t (steps)


data
D = 28.7743
Relaxation rate Γ
0.2
0.15
0.1
0.05
0
0
0.02
0.04
0.06
0.08
q (nm-1)

```
function newfinal=voronoiplacing(boxsize,stepsize,reps,ploton)

% Generates 2D randomly ordered, non-overlapping point arrangement
 using
% voronoi cells starting from a quadratic lattice. Open cells at edges
% avoided by additional points set on edges.
%
% Input:
% size        =   size of quadratic simulation box
% stepsize    =   lattice constant of starting round. Number of
 particle is
%                 then given by: (size/stepsize)^2.
% reps        =   number of iterations of voronoi placing.
% ploton      =   1, to show plots; other, don't show plots.
 ATTENTION:
%                 Figure handling adapted to Matlab R2014b, will
 likely not
%                 work for older version.
%
% Output: newfinal - coordinates of final positions.
%
% Version 1.0: FL, 07/11/2014

if nargin < 3
    error('Not enough input parameters! Provide box size, step size
 and number of repetitions.')
elseif nargin == 3
    ploton = 0;
elseif nargin > 4
    error('Too much input parameters...!')
end

xs=stepsize/2:stepsize:boxsize-stepsize/2;
ys=stepsize/2:stepsize:boxsize-stepsize/2;

[coox, cooy]=meshgrid(xs,ys);
startorg=[coox(:) cooy(:)];
fill=[0:round(stepsize/3):boxsize]';
zer=zeros(length(fill),1);
hun=ones(length(fill),1)*boxsize;
filler=[ [fill zer]; [zer(2:end) fill(2:end)]; [fill hun]; [hun
 fill] ];
start=[startorg; filler];

for cou=1:reps

    if ploton==1
        figure(1)
        clf
        title(['Rep. #',num2str(cou)])
        hold on
        voronoi(start(:,1),start(:,2))
```

```
    end


    [v,w]=voronoin(start);
    new=zeros(length(startorg),1);

    jj=0;
    while jj<length(startorg)
          jj=jj+1;
        try
            p=poly2mask(v(w{jj},1),v(w{jj},2),boxsize,boxsize);
        catch err
        end

        if ~exist('err')

            [ind1,ind2]=find(p==1);
            r=max([1, round(rand(1,1)*length(ind1))]);
            new(jj,2)=ind1(r);
            new(jj,1)=ind2(r);

            if ploton==1
                fig1=figure(1);
                fig1.Position=[40 650 560 420];

                plot(start(jj,1),start(jj,2),'ob')
                plot(new(jj,1),new(jj,2),'+r')

                fig2=figure(2);
                fig2.Position=[40 100 560 420];
                title('current Voronoi cell')
                imagesc(p)
                axis xy
                pause(0.02)
            end
        else
            clear err
        end
    end

    start=[new; filler];
end

if ploton==1
    fig3=figure(3);
    clf
    fig3.Position=[700 300 560 420];
    title('Final placement')
    plot(startorg(:,1),startorg(:,2),'+')
    hold on
    plot(new(:,1),new(:,2),'o')
    ylim([0 boxsize])
    xlim([0 boxsize])
    legend('start','final result')
```

```
end
newfinal=new;
```

```
Error using voronoiplacing (line 21)
Not enough input parameters! Provide box size, step size and number of
 repetitions.
```

Published with MATLAB® R2018b

```
function A=getcircle(CENx,CENy,Rout,Xdim,Ydim)

A=zeros(Xdim,Ydim);

for i=1:Xdim
    for j=1:Ydim
        %relative coordinates
        xi=i-CENx;
        yj=j-CENy;
        radius=sqrt(xi.^2+yj.^2);

        if (radius< Rout)
            A(i,j)=1;
        end

    end
end
```

```
Not enough input arguments.

Error in getcircle (line 3)
A=zeros(Xdim,Ydim);
```

Published with MATLAB® R2018b