Supplementary information

for the manuscript

Peptides@mica: From affinity to adhesion mechanism

A. Gladytz[1,2], T. John[1,2], T. Gladytz[2], R. Hassert[3], M. Pagel[3], H.J. Risselada[1], S. Naumov[1], A.G. Beck-Sickinger[3], B. Abel[1,2]

[1] *Leibniz Institute of Surface Modification (IOM), Permoserstrasse 15, 04318 Leipzig, Germany*
[2] *Wilhelm-Ostwald-Institute for Physical and Theoretical Chemistry, University of Leipzig, Linnéstrasse 3, 04103 Leipzig, Germany*
[3] *Institute of Biochemistry, University of Leipzig, Brüderstrasse 34, 04103 Leipzig, Germany*

Definition of the autocorrelation function Autocor (S1) and the convolution Autocor*Autocor (S2)

The 2D spatial autocorrelation function of every image with a height profile h(x,y) is defined as the sum over all products between an image and its copy translated by the vector $\vec{r} = \binom{u}{v}$ according to equation S1.

$$Autocor(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y)h(x+u,y+v)dx\,dy \quad \text{(S1)}$$

The convolution of two (autocorrelation) functions f(x,y) and g(x,y) is defined by equation S2:

$$(f * g)(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)g(u-x,v-y)dx\,dy \quad \text{(S2)}$$

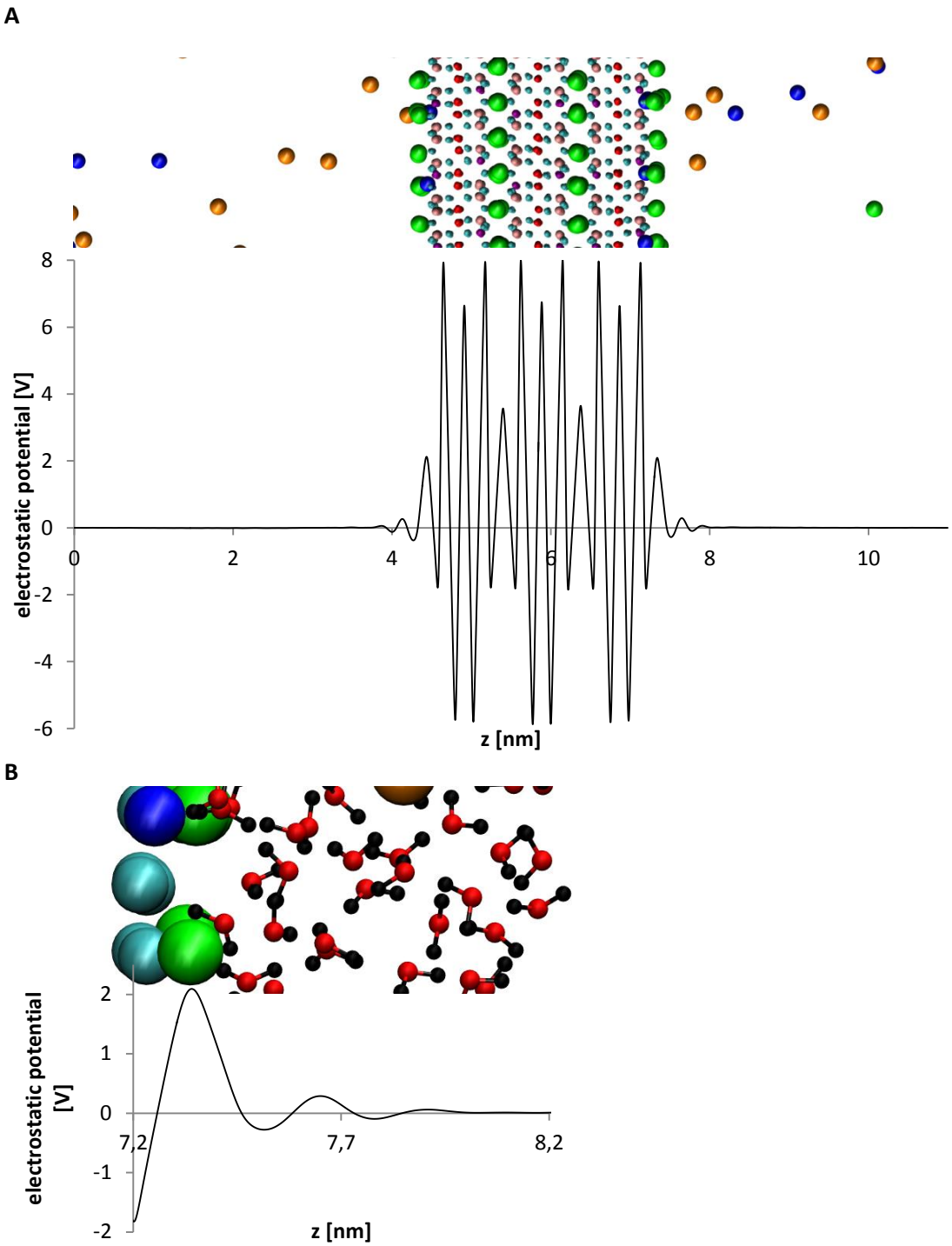

Figure S1: Electrostatic potential along the z-direction of the box (after translation of the symmetric mica surface to the middle of the box.) **A:** Potential through the whole box with illustrations of the differently charged mica layers and the NaCl ions (K^+: green, Si^{4+}: pink, Al^{3+} at octahedral site: red, O^{2-}: cyan, Na^+: blue, Cl^-: orange). Each mica layer with its corresponding charge can be seen. **B:** Zoom in to demonstrate the development of the potential in the vicinity of the surface. The negative surface charge of the O^{2-} (cyan) anions is efficiently shielded by K^+ (green) and Na^+(blue). The remaining charge is quickly damped by the polarization of water which is evident from the damped oscillations of the potential. After 1.15 nm (z=8.2 nm) behind the surface (position of K^+: Z= 7.35 nm), the potential is negligible.

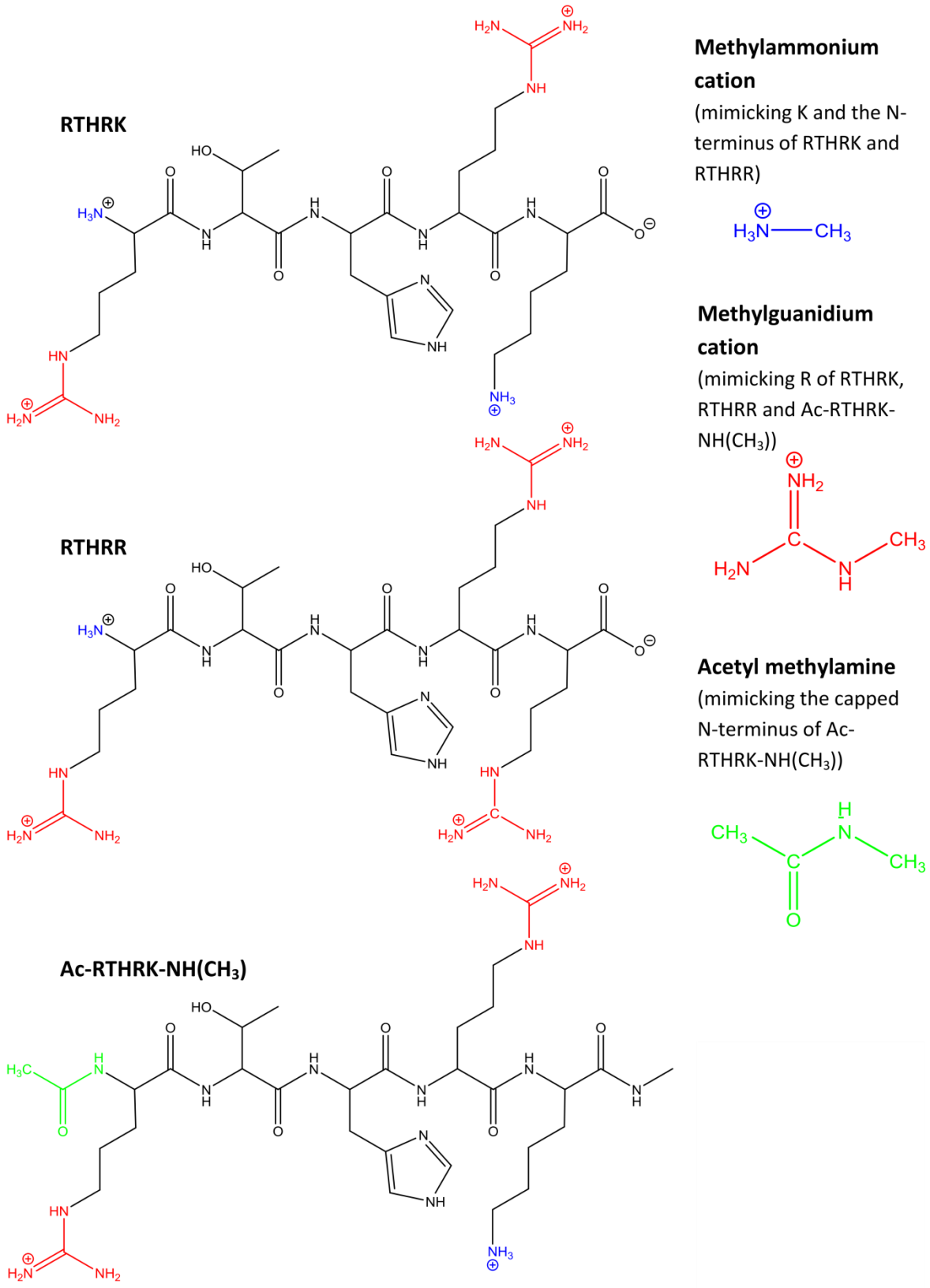

Figure S2: Nomenclature and chemical structures of the peptides used for MD simulations (left) and the analogues of their adsorbing functional groups as used for DFT calculations (right).

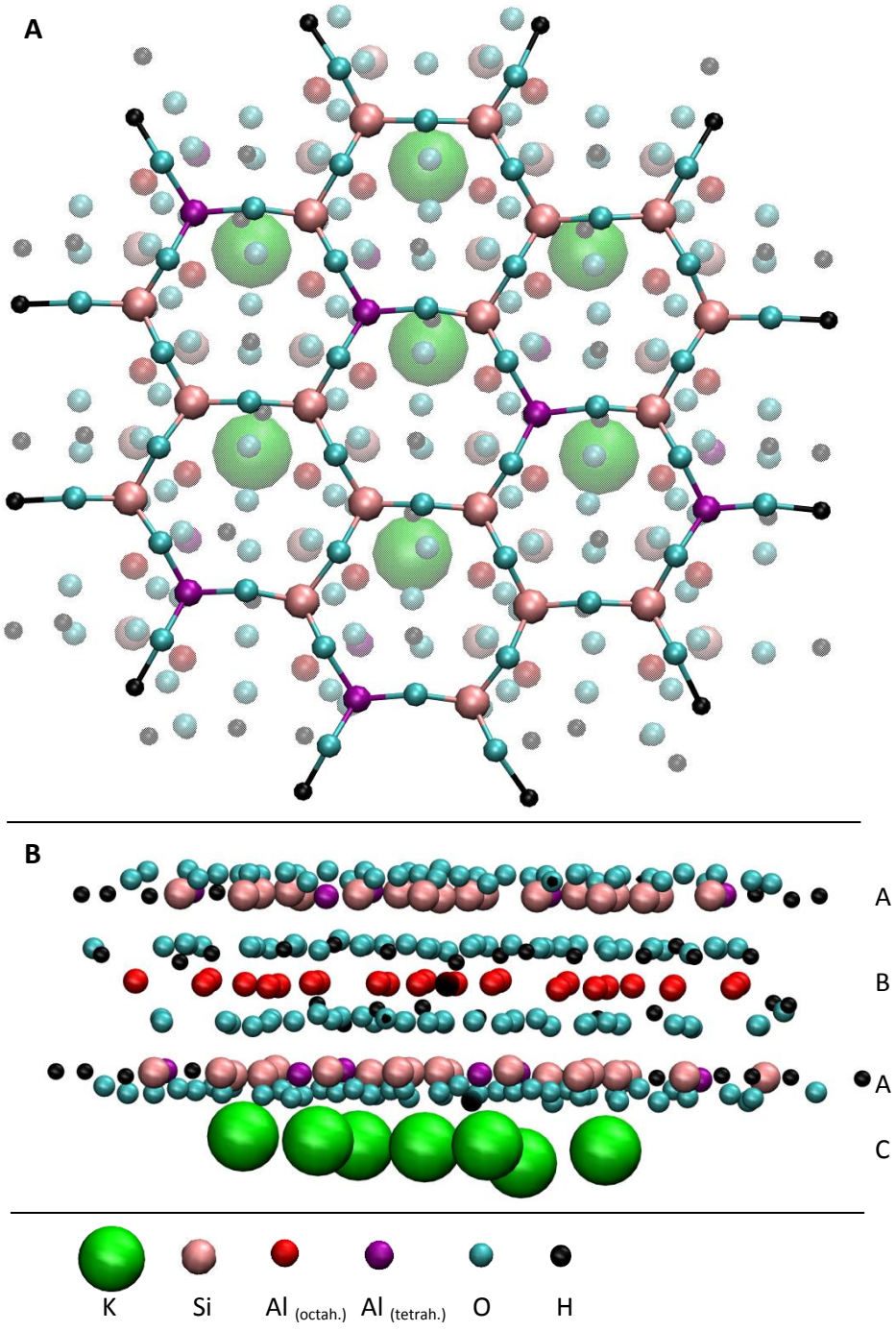

Figure S3: Mica model as used in DFT calculations **A:** View along the c-axis. **B:** View along the a-axis. One repetition of the A-B-A-C layer motif was simulated. For further details see the main text.

Function for the evaluation of the autocorrelation of an AFM image

In the following part, the function for the evaluation of the autocorrelation of an AFM image using GNU Octave 3.6.4. is given.

Workflow

```
## Copyright (C) 2015 Thomas & Anika
##
## This program is free software; you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation; either version 3 of the License, or
## (at your option) any later version.
##
## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
## GNU General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with Octave; see the file COPYING.  If not, see
## <http://www.gnu.org/licenses/>.

## workflow

%%% exemplified workflow of the evaluation of the grain shapes using
the autocorrelation of the AFM images

clear;

[data, meta,files] = loaddata('bc5*.txt'); % import all AFM image
autocorrelations
pkg load optim % loading needed octave packages
pkg load struct

i=1; % go through all individual pictures. Not to miss any artifacts in
the images no loop is used
files{i} % prints the filename of the current autocorrelation
imagesc(data{i}); % Plot the autocorrelation. Check for artifacts.
[widths,fit,coeff,result(3:9,i)] =
fitconv((10^24)*data{i}(250:262,250:262),[3.4; 3.6; 1.13; -
0.1],meta.imgsize(i),1); % fit a model to the autocorrelations.
result(11,i) = prod((size(data{i})+1)./512)*(meta.imgsize(i).^2); %
write the image area to the output
result(1:2,i) = [meta.conc(i); meta.imgsize(i)]; % write the current
concentration value and the image dimensions to the output

i=2;
files{i}
imagesc(data{i}); % leichte Zeilenfehler
```

```
[widths,fit,coeff,result(3:9,i)] =
fitconv((10^24)*data{i}(247:265,247:265),[3.4; 3.6; 1.13; -
0.1],meta.imgsize(i),1);
result(11,i) = prod((size(data{i})+1)./512)*(meta.imgsize(i).^2);
result(1:2,i) = [meta.conc(i); meta.imgsize(i)];

% ... continue with all pictures

% export to ascii format
save -ascii widths.txt result
```

Loaddata

```
function [data, meta, files] = loaddata(filemask, anzahl) % filemask =
'RTHRK*'
%LOADDATA reads AFM autocorrelation images from text files
%  some additional information (output: meta) like concentration and
image size is additionally read from the filenames
%  files is a cell array containing all filenames read
format compact;
files=dir(filemask); % look for files
numdat = length(files);
files={files.name};

% --- set the number of images to be read either to the number of
images found or the optionally predefined argument anzahl.
if ~exist('anzahl'), anzahl = numdat; else anzahl = min(numdat,anzahl);
end;
if anzahl == 0;
    warning(['Did not find any data files.'])
end;

for i=1:anzahl % loop through files
    filename = files{i};
    disp(['loading file ' int2str(i) ' of ' int2str(anzahl) ': '
files{i}]) % display progress
    data{i} = dlmread(files{i});%, ' ', 0, 0); % load autocorrelation
image
    % ---- read some additional meta data from the filenames
(specificly adapted to the nomenclature of the files)
    pos = find(files{i}=='_'); % find space in filenames
    meta.conc(i) = sscanf(files{i}(pos(1)+1:pos(2)-1), '%f',1); % read
concentration information between first and second spacer.
    meta.imgsize(i) = sscanf(files{i}(pos(end-4)+1:pos(end-3)-1),
'%d',1); % read imagesize information
endfor
endfunction
```

Fitconf

```
function [widths,fit,coeff,result] = fitconv(data, inival, imgsize)

result = zeros(6,1); % predefine results variable
widths = bfgsmin('evalconvgauss2',{inival,data}); % minimize deviation
of data from model defined in evalconvgauss2
widths(1:min(length(widths),3)) = abs(widths(1:min(length(widths),3)));
% constrain widths to be positive (no change as widths are squared in
the model)
switch length(widths) % switch through different number of optimizable
parameters
  case 0
    error('widths should not be empty.')
  case 1
    triangFWHM = widths(1); % only tip diameter is used and optimized
 %   xy_rat=1;
 %   theta=0;
  case 2
    triangFWHM = widths(1);
    gausswidth = widths(2); % additionally objects with a gaussian size
distribution are considered and optimized
%    xy_rat = 1;
%    theta=0;
  case 3
    triangFWHM = widths(1);
    gausswidth = widths(2);
    xy_rat = widths(3); % elliptic shape of objects is considered and
the x-y-ratio of the ellipse is optimized
%    theta=0;
  case 4
    triangFWHM = widths(1);
    gausswidth = widths(2);
    xy_rat = widths(3);
    theta=widths(4); % rotation of the elliptic shape is considered and
the rotation angle is optimized
endswitch

% --- evaluate and save fit results and print them to the console.
triangFWHM % print tip autocorrelation width
tipdiam_nm = triangFWHM*imgsize*1000./512 % compute and print tip
diameter
result(1) = tipdiam_nm; % save as the first element of the results
vector

% recompute contributions of the individual autocorrelation models to
the overall autocorrelation.
[deviation, fit, coeff] = evalconvgauss2(widths, data);
triangheight_pm2 = coeff(1)./(imgsize.^2) % tip autocorrelation height
in pm²
result(4) = triangheight_pm2;
```

```
bgheight_pm2 = coeff(2) % autocorrelation background height
diracheight_pm2 = coeff(3)./(imgsize.^2) % noise autocorrelation height
result(5) = diracheight_pm2;

if exist('gausswidth') % only if objects are considered
    gausswidth % object autocorrelation width
    objdiam_nm = sqrt(log(2))*gausswidth*imgsize*1000./512 % mean
object diameter
    result(2) = objdiam_nm;
    totalwidth = sqrt((triangFWHM.^2)./6+(gausswidth.^2)./2)
    totalFWHM_nm = totalwidth*2.4*imgsize*1000./512 % total FWHM of the
autocorrelation
    result(3) = totalFWHM_nm;
    gaussheight_pm2 = coeff(4)./(imgsize.^2) % height of the object
autocorrelation
    result(6) = gaussheight_pm2;
endif
if exist('xy_rat') % if elliptic shape of the objects are considered
   xy_rat = xy_rat.^2
   result(7) = xy_rat; % x-y-ratio of the mean object ellipse
endif
if exist('theta') % if rotation of the autocorrelation ellipse is
considered
   theta
   theta = mod(theta*180./pi,360) % rotation angle of the mean object
ellipse
endif

figure(1); imagesc(fit); % plot figures showing fit and original data
for comparison
figure(2); imagesc(data);
deviation % print fit deviation to the console
endfunction
```

evalconvgauss2

```
function [deviation, fit, coeff] = evalconvgauss2(widths, data,
plotflag)

% --- read and constrain width input
widths(1:min(length(widths),3)) = abs(widths(1:min(length(widths),3)));
% constrain widths to be positive (no change as widths are squared in
the model)
switch length(widths) % switch through different number of optimizable
parameters
  case 0
    error('widths should not be empty.')
  case 1
    triangFWHM = widths(1); % only tip diameter is used and optimized
    xy_rat=1;
```

```
    theta=0;
  case 2
    triangFWHM = widths(1);
    gausswidth = widths(2); % additionally objects with a gaussian size
distribution are considered and optimized
    xy_rat = 1;
    theta=0;
    xy_rat_gauss = 1;
    theta_gauss = 0;
  case 3
    triangFWHM = widths(1);
    gausswidth = widths(2);
    xy_rat = widths(3); % elliptic shape of objects is considered and
the x-y-ratio of the ellipse is optimized
    theta=0;
    xy_rat_gauss = xy_rat;
    theta_gauss = 0;
  case 4
    triangFWHM = widths(1);
    gausswidth = widths(2);
    xy_rat = widths(3);
    theta=widths(4); % rotation of the elliptic shape is considered and
the rotation angle is optimized
    xy_rat_gauss = xy_rat;
    theta_gauss = theta;
endswitch

[m,n] = size(data);

% --- define coordinate system
x = -(n-1)/2:(n-1)/2;
y = (-(m-1)/2:(m-1)/2)';
% --- stretch and rotate coordinate system for elliptic autocorrelation
shapes (xy_rat is the square root of the x-y-ratio of the ellipse)
xrot = (ones(m,1)*(x*cos(theta)) + (y*sin(theta))*ones(1,n)).*xy_rat;
yrot = (ones(m,1)*(x*sin(-theta)) + (y*cos(theta))*ones(1,n))./xy_rat;
r = sqrt(y.^2*ones(1,n) + ones(m,1)*x.^2); % distance of each point
from the center
r_rot = sqrt(yrot.^2 + xrot.^2); % stretched and rotated distance of
each point from the center

% --- compute tip autocorrelation function
cone = zeros(m,n);
rsel = r_rot(r_rot<triangwidth);
tipdiam = triangwidth;
cone(r_rot<tipdiam) = (2./pi).*(acos(rsel./tipdiam) -
(rsel.*sqrt(tipdiam.^2-rsel.^2)./(tipdiam.^2)));

if length(widths)>=2
    % --- compute object autocorrelation function
    % same stretching and rotation of the coordinate system also for
the object shapes
```

```
    xrot = (ones(m,1)*(x*sin(theta_gauss)) +
(y*cos(theta_gauss))*ones(1,n)).*xy_rat_gauss;
    yrot = (ones(m,1)*(x*cos(-theta_gauss)) +
(y*sin(theta_gauss))*ones(1,n))./xy_rat_gauss;
    r_rot = sqrt(yrot.^2 + xrot.^2);
    gauss = exp(-((r_rot./gausswidth).^2)); % evaluate gauss function
    fit = conv2(cone,gauss,'same'); % convolute object autocorrelation
with tip autocorrelation
    if fit((m+1)/2,(n+1)/2)>0
        fit = fit./fit((m+1)/2,(n+1)/2); % normalize object
autocorrelation to a center value of 1.
    endif
else
    fit = [];
endif
dirac = zeros(m*n,1);
dirac((m*n+1)/2) = 1; % define noise autocorrelation
r((m+1)/2,(n+1)/2) = min(r(r~=r((m+1)/2,(n+1)/2))); % change the radius
value at the center to a non-zero value. (used for weighting)
if length(widths)==1
    M = [reshape(cone,m*n,1) ones(m*n,1) dirac]; % Matrix for linearly
fitting tip autocorrelation, background and dirac function to the data
    % weighting to balance the increasing number of data points at
larger distances from the center in the 2D autocorrelation
    M_sc = M./(reshape(r,m*n,1)*ones(1,3));
else
    M = [reshape(cone,m*n,1) ones(m*n,1) dirac reshape(fit,m*n,1)]; %
includes object autocorrelation
    M_sc = M./(reshape(r,m*n,1)*ones(1,4));
endif

% --- linear fit of the model to the data.
coeff = M_sc\reshape(data./r,m*n,1);

% --- check for negative contributions of individual model components
and constrain them to positive values.
if sum(coeff<0)>=1 & prod(coeff<0)<1 % there are negative contributions
and at least one positive contribution.
    % recompute the individual contributions ignoring the models that
gave a negative response
    coeff(coeff>=0) = M_sc(:,coeff>=0)\reshape(data./r,m*n,1);
    coeff(coeff<0) = 0; % set negative values to 0.
endif

% --- compute current fit
fitres = M*coeff;
fit = reshape(fitres,m,n);

% --- compute deviation between data and model; weighted by radius like
above; deviation is the first function output, which is minimized by
bfgsmin
deviation = sum(((reshape(data,m*n,1)-
fitres).^2)./reshape(r.^2,m*n,1));
```

```
% --- optional plot showing the individual contributions to the AFM
autocorrelation
if exist('plotflag')
  if plotflag
    figure(1); imagesc(fit);
    figure(2); imagesc(data);
    fitcompres = M.*(ones(m*n,1)*coeff');
    fitcone = reshape(fitcompres(:,1),m,n);
    fitbg = reshape(fitcompres(:,2),m,n);
    fitdirac = reshape(fitcompres(:,3),m,n);
    fitgauss = reshape(fitcompres(:,4),m,n);
    figure(3); plot((-(m-1)/2:(m-1)/2)', [data(:,(n+1)/2)
fitcone(:,(n+1)/2)+fitbg(:,(n+1)/2) fitbg(:,(n+1)/2)],'-
','linewidth',2, ...
        (-(m-1)/2:(m-1)/2)',
fitdirac(:,(n+1)/2)+fitbg(:,(n+1)/2),'.','linewidth',2, ...
        (-(m-1)/2:(m-1)/2)', fitgauss(:,(n+1)/2)+fitbg(:,(n+1)/2),'-
','linewidth',2);
    legend('AFM data', 'tip', 'offset', 'noise', 'object')
    axis('nolabel')%set(gca(),'xticklabel',[],)
    xlabel('cross section')
    ylabel('autocorrelation')
    print('autocorr_800.png', '-FHelvetica:10', '-r800', '-S3200,2400',
'-dpng');
    print('autocorr.png', '-FHelvetica:10', '-r1200', '-S4800,3600', '-
dpng');
    print('autocorr.eps', '-FHelvetica:10', '-r1200', '-S4800,3600', '-
tight', '-deps2');
    print('autocorr.pdf', '-FHelvetica:10', '-r1200', '-S4800,3600', '-
tight', '-dpdf');
  endif
endif
endfunction
```