ASSIGNMENT 13: POTENTIAL ENERGY SURFACES

E. ARNOLD, M-S. LIU, R. PRABHU & C.S. RICHARDS
THE UNIVERSITY OF CAMBRIDGE

Written in XƎLATEX

POTENTIAL ENERGY SURFACES

THEORETICAL BACKGROUND

Corrugation Function as a Fourier Series

In previous tutorials, we have demonstrated that a hard-wall corrugation function can be constructed for a close-packed plane. This is achieved by expanding the surface potential as a two-dimensional Fourier series, which can be fitted to the lattice; the properties of the fitted series must match the real lattice, and so we must specify several conditions: the corrugation function must match the periodicity of the underlying lattice, and thus must take the form:

$$\zeta(\mathbf{R}) = \sum_{\mathbf{G}} A(\mathbf{G})\, e^{i\mathbf{G}\cdot\mathbf{R}},$$

where $\mathbf{G}$ is any reciprocal lattice vector, and $A(\mathbf{G})$ is its corresponding Fourier coefficient.

The hard-wall potentials tutorial gives the following strategy for constructing a corrugation function:

1. Determine the Bravais lattice of the surface.
2. Find the set of basis lattice vectors for the aforementioned lattice, denoted $\mathbf{a}_1$ and $\mathbf{a}_2$.
3. Find the corresponding basis reciprocal lattice vectors, denoted $\mathbf{b}_1$ and $\mathbf{b}_2$.
4. Use the basis vectors of the reciprocal lattice to find as many reciprocal lattice vectors as necessary of the form:

$$\mathbf{G} = h\mathbf{b}_1 + k\mathbf{b}_2,$$

   where h and k are integers.
5. Use the reciprocal lattice vectors to construct a truncated Fourier series for the corrugation function. The coefficients for the terms should be chosen such that the corrugation has local maxima at the positions of the atoms composing the surface.

Number of Fourier Components

Different numbers of Fourier components are required, dependent on circumstance, to construct the desired features of a lattice. In this assignment, we consider the example of a close-packed plane. This is because it has remarkable symmetry properties.

In the previous tutorial it was shown that two reciprocal lattice vectors were insufficient to represent the rotational symmetry of a close-packed plane. Three vectors successfully represented the symmetry, but failed to distinguish between the three types of bridge site, and the two types of hollow site. A labelled corrugation function for a close=packed plane is given in the margin to act as a memory aid.

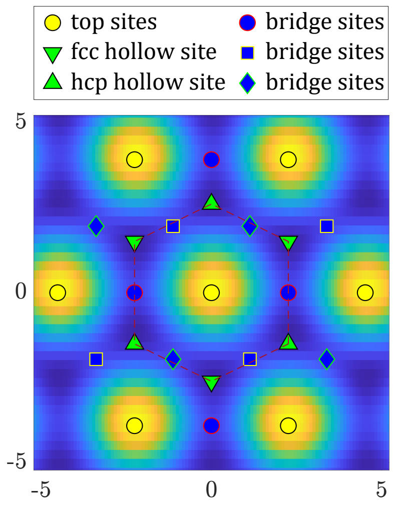

Figure 1 | A labelled plot of the corrugation as computed in the hard-wall corrugation assignment.

The Wigner-Seitz Unit cell

The dashed line in the figure denotes the *Wigner-Seitz* unit cell of the lattice. The Wigner-Seitz cell around a lattice point is defined as the locus of points that are closer to that lattice point than any others. This cell is primitive, and tesselates all of real space. This concept is used in the discussion of generating advanced potential energy surfaces.

Computing Higher Fourier Components

In principle, it is possible to compute the corrugation function exactly for each point in the Wigner-Seitz (WS) unit cell of the lattice; and then this function can then be tessellated over all of real space, giving the complete corrugation function. This method is impractical: it would require sampling an infinite number of points in space, which cannot be done. How do we avoid this problem? A common approach is to discretise the WS unit cell. The number of samples required can be further reduced by exploiting the symmetry of the lattice; thus we only sample a portion of the WS cell, at the expense of losing the distinction between some sites on the potential surface.

A common approach is to calculate the corrugation across a grid of points in the so-called *irreducible triangle*. This is defined to be the smallest triangle on which the FCC and HCP hollow sites differ. Seen in the figure 2 in the margin, the large circles are sufficient to differentiate between hollow sites. However, these large circles form a hexagonal grid rotated at 30° to the cartesian basis vectors. This makes operations such as the Fourier transform more problematic; extra points are included to avoid a rotated grid, and to improve the resolution of the simulated functions. The figure contains 31 discrete points, which form a hexagonal grid; this grid has the same orientation as the lattice, but a different spacing. The lattice defined by this grid of points is known as the "grid lattice". The number 31 is not arbitrary.

Whilst a 31 point grid cannot distinguish between bridge sites, it is the minimum number of points required to represent the bridge sties as local minima (as they should be).

Once the potential has been calculated across the grid, the triangle can be tessellated across real space using symmetry operations. This produces a corrugation function. However, since our irreducible triangle as appearing in the figure only contains one bridge site, tessellating in the manner described below will produce a corrugation function that does not distinguish between bridge sites (according to whether they represent FCC or HCP structures). For all sites to be distinguished, three irreducible triangles would be required. Fortunately the distinction between bridge sites is not important for most adsorbates; but the distinction between hollow sites is of interest in the study of adsorption and adsorbate motion on the surface, because adsorbate positions at hollow sites are usually more stable.

The process of tiling the surface exploits the hexagonal symmetry of the lattice, and in our case is:

1. Reflect the irreducible triangle across one face.
2. Rotate the diamond formed about one of the points common to both the original and reflected triangle to tile a hexagonal unit cell. Points on opposite sides of the hexagon are equivalent since they are one lattice vector apart.
3. The unit cell produced by this process is a WS cell in the basic lattice, and can be used to tessellate all of real space by translating the hexagonal grid by basic lattice vectors.

In this construction the top-hcp distance is 1/3 of the basic lattice nearest-neighbour distance in the same direction, and the top-bridge distance is 1/2 the basic lattice separation. Thus the separation of the grid lattice is 1/12th that of the basic lattice. The reciprocal grid lattice is therefore 12 times the spacing of the reciprocal basic lattice, and the WS cell of the reciprocal basic space (the 1st BZ) is correspondingly 12 times the size of the WS cell of the reciprocal grid lattice.

The 1st BZ contains all information required to describe every reciprocal lattice vector corresponding to the grid in real space. To describe points outside the WS cell we can add an integer multiple of a unique reciprocal lattice vector in the 1st BZ (we denote such reciprocal lattice vector by $\mathbf{G}_1$) to get back to the first BZ. Thanks to the properties of complex exponentiation, when such a sum has its inner product taken with any of the real space vectors in the expression for the corrugation, the corrective $\mathbf{G}_1$ component reduces to a factor $\exp(2\pi i) = 1$. The Fourier coefficients of the reciprocal lattice vectors that lie outside the 1st BZ will also be equivalent to those inside.

Considering only the unique reciprocal vectors within the 1st BZ,

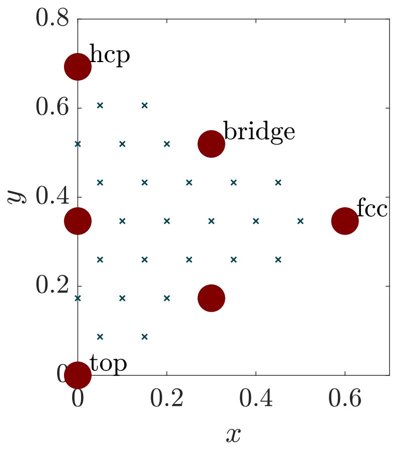

Figure 2 | The large red circles are the only points required to differentiate between hollow sites. However, additional points have been included to provide a more general procedure, giving a total of 31 points.

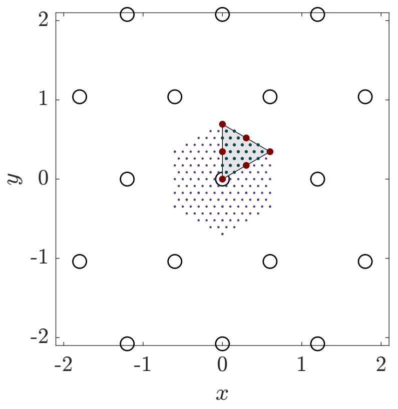

Figure 3 | Depiction of the real space lattice along with the irreducible triangle (shaded) and the set of points encompassed by the WS cell.

the corrugation at the grid point $\mathbf{R}_j$ (for $j \in \{1, 2, ..., n_R\}$) is given by:

$$\zeta(\mathbf{R}_j) = \sum_{\mathbf{G}_1} A(\mathbf{G}_1)\, e^{i\mathbf{G}_1 \cdot \mathbf{R}_j}.$$

In general, the process for computation is as follows:

1. Take a grid of points in real space which have the same hexagonal structure as the basic lattice but shrunk by an integer factor N (31 in the previous example).
2. Compute the unique reciprocal lattice vectors, $\mathbf{G}_1$, inside the 1st Brillouin zone as determined by a reciprocal space lattice that has N times the dimensions of the basic reciprocal lattice.
3. Calculate the corrugation at each point in the real space WS cell, where the coefficients $A(\mathbf{G}_1)$ are free variables in the model. It is necessary to give a fractional 'weight' to each point so that the potential at edges/corners is not double/triple counted respectively during tessellation.

Anthraquinone

The task below uses the motivating example of the potential energy surface for an Anthraquinone (AQ) molecule adsorbed on the Cu(111) plane, so it would be prudent to establish a good phenomenological understanding of this system.

Anthraquinone is composed of three benzene rings fused in a linear arrangement with oxygens bonded at its midpoint, often called the 'linkers' of the AQ molecule. The molecule has two-fold rotational symmetry about any axis through its centre-of-mass and perpendicular to its main surface.

Since AQ is a large rod-like molecule it can't be modelled as occupying a single site on the substrate lattice. On the Cu(111) surface, both of the oxygen linkers interact strongly with the substrate whereas the central body of the molecule remains somewhat raised above the surface (in fact, the ends of the rod bend away from the surface whilst the centre remains close to it [1]). As such, in constructing a potential energy surface for the molecule we must consider not only the position of its centre-of-mass, but also its polar angle with respect to the basis of the substrate lattice.

Modelling using Density Functional Theory shows that AQ molecules preferentially adsorb on hcp hollow sites, which are the global minima of the surface. There are local maxima at fcc sites and local minima at bridge sites. The molecules adsorb on top sites.

This unusual mode of adsorption lends itself to some unusual properties. Anthraquinone can diffuse uniaxially along the close-packed directions of the surface [2,3]. Additionally, AQ molecules may 'pick up' light adsorbates (e.g. CO_2) that pass close to its oxygen linkers, carrying them along the surface and thereby acting as a cargo-carrying agent. The unusual adsorption also causes AQ adsorbates to form honeycomb structures at high coverages, but that is beyond the scope of this assignment.

However, the AQ molecule does not move in a straight line as one might expect. Instead, it performs four rotations of $\pm 15°$ about one of its linkers to move one substrate lattice spacing, producing a characteristic 'walking' motion.

Note that each stage is named after the type of site above which the centre-of-mass of the molecule lies. In modelling the potential surface it is possible to consider only the position of the molecule's centre-of-mass, as well as its orientation. Its orientation is given by the angle θ, defined as the anticlockwise angle of its length from the [1 0] azimuth (positive x-direction).

Rotations of greater than $\pm 15°$ are possible but have a much higher energetic barrier so are less common. Thus, AQ motion the Cu(111) surface is characterised by long periods of linear walking motion interspersed with rotations through larger angles.

TASK

1. The first task is to investigate the existing code for producing the potential energy surface. To represent all bridge sites as local minima, which is required in modelling the moition of AQ on Cu(111), 31 points are required in the irreducible triangle. **However**, since AQ is a rod-shaped molecule it is not rotationally-symmetric, and as such one must compute the potential for each allowed orientation of the AQ molecule.
   (a) To start, run `file1.m`. This file constructs an initial estimate of the potential based on experimentally-determined values. It generates a file `Potential_AQ_base_file.mat`, which stores the following values:
      - `xypos`, a 157×2 vector of all the (x, y)-positions of grid points in the WS cell. 157 points is the regular number of points obtained from tiling a 31-point irreducible triangle across the WS cell.
      - `thetapos`, a 1×24 vector of all the allowed orientations of the AQ molecule with respect to the surface lattice. Note that θ is the angle of the main body of the AQ molecule, measured anticlockwise from the [1 0] azimuth on the surface.
      - `weight`, a 1×157 vector of weights for each x, y-position to prevent double/treble counting for edges/corners of the WS cell respectively.
      - `V3Dinterp_AQ`, a 157×24 vector of potential values for each

point (rows) with given AQ orientation (columns).

Since the values in this file are purely experimental it is simply given in the appendix.

(b) Note that in neither `file1.m` nor in `file2.m` is there any mention of flipping or rotation of an irreducible triangle - the entire WS cell (in fact, a rectangle encompassing said cell) is constructed and then different values set for the Fourier components at each angle. Why was the irreducible triangle method **not** used here? (Note you do not have to fully understand the code in each file - use the comments to get a general understanding).

2. (a) Second, run `file2.m`. This should produce:
   - A plot of all the points in the WS cell. Note the first 31 points are in the irreducable triangle.
   - A plot of all the points in the WS cell after interpolation. This plot is performed within the `halvespacing.m` function, given in the appendices.
   - 13 plots of the corrugation function for varying angles of the AQ molecule in the range 0° to 360°.

   (b) Having produced the above plots of the potential, note that the shape of the potential changes for each orientation of the rod-like molecule. Why is the change in potential periodic, with a period of 180°?
3. With reference to the discussion of AQ motion in the theoretical background, explain the shape of the potential energy surface at each orientation.
4. (a) Take the generated file, `loadPES.mat`, and run it in PIGLE to simulate a trajectory of a single AQ molecule on the Cu(111) surface using MD methods. The motion should match that described in the theoretical background.

   The required code for the PIGLE UI files is given below:

   The first file, `pigle_ui.m`, sets up the preferences for several options, as well as the scattering parameters and the length of the simulation.

```
%% parameters for config_model.m

z_enabled = 0; % Enable/Disable (1/0) motion
    perpendicular to the surface
dKz_include_in_isf = 0; % Enable/Disable (1/0)
    calculation of the ISF for perpendicular momentum
    transfer
theta_enabled = 1; % Enable/Disable (1/0) rotation of
    rigid body (rotation axis perpendicular to the
    surface)
zero_p_init = 0; % set initial momentum be set to zero? (
    if set to 0, p_init will correspond to thermal
    distribution)
interactions_active = 0; % Enable/Disable (1/0)
```

```
    interadsorbate interactions
N_runs = 1; % Number of runs
run_parallel = 1; % Enable/Disable (1/0) of parallel
    computing. If parallel computing toolbox is not
    installed - set this option to zero.

% Specify dK as a 2D vector, 3rd dim is azimuths.
dK = [0.05 0.1 0.15 0.2:0.1:5];
azim_1 = [0 1];
azim_2 = [1 1];

% specify beam parameters and geometrical parameters for
    scatering calculations
theta_tot = 44.4; % Degrees
beam_ki = 3.3977; % Angstrom ^{-1}

% Specify simulation time parameters
% (those will be adjusted by the program, see below if
    interested)
sample_time = 2e-3;
sample_time_clist = 1e-2;
isf_sample_time = 5e-2; % time step for which the
    position will be recorded, for later use in the ISF
    calculations
thermalizing_time = 20; % during this initial period, the
     trajectory won't be recorded.
stop_time = 1024*10;% Total simulation time.

% N_steps and N_ISF_steps are calculated after PIGLE
    adjusts the requested time parameters
max_N_steps = 1e10; % The simulation won't start if the
    calculated number of steps is greater than
    max_N_steps. Prevents too-long runs.
max_N_ISF_steps = 6e5; % The simulation won't start if
    the calculated number of steps is greater than
    max_N_ISF_steps. Prevents insensible memory
    exploitation
```

The next file, `pigle_ui_surface_params.m`, sets up the parameters of the surface as well as the adsorbates on it. It also sets up the potential by calling @loadPES

```
%% params for surface_params.m
T=200; % Temperature in Kelvin
Nprtcl_total = 1; % will be rounded to justify the number
     density
mass_list = [208]; % vector of masses for all species (
    one mass per species ...)
radius = 6; % for calculating angular mass, however, one
    can define angular mass directly (see below)
number_density = [0.05]; % relative number density for
    each mass.
eta = 2; % helps to define variables down below, but the
    user can define it directly or using any other method
eta_theta = 100; % "
tau = [1]; % "

a1=2.5560;                               % Copper 111
    lattice constant in Angstrom
x0 = 0; nx = 120; xdim = a1;             % x dimention
    params of the unitcell/PES
```

```
y0 = 0; ny = 200; ydim = a1*sqrt(3.0);    % y dimention params of the unitcell/PES
z0 = 0; nz = 20; zdim = 10;               % z dimention params of the unitcell/PES
theta0 = 0; ntheta = 24; thetadim = 2*pi; % theta dimention params of the unitcell/PES
numOfPrmtvCells = [1 1]; % How many primitive cells exist in the XY potential (with periodic boundary conditions)

unitcell = prepFuncs.make_unitcell([nx xdim x0],[ny ydim y0],'z',[nz zdim z0 z_enabled],'theta',[ntheta thetadim theta0 theta_enabled],'numOfPrmtvCells', numOfPrmtvCells);

angular_mass_list = mass_list.*radius.^2;

Nmass = length(mass_list);

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Adsorbate configuration %
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% See prepare_configuration.m.
% If new adsorbate_conf_case_num (in addition to '1') are implemented in
% prepare_configuration.m, surface_params.m needs to be updated.
% Case '1' is for top symmetric molecule
% Each parameter needs to be stored either as a single cell (which will be distributed to all
% populations), or as a cell-array (with the i'th element distributed to the i'th
% population)
%
r_conf_case_num = {1};
r_conf_radius   = {0};
r_conf_Natoms   = {1};
CoM_form_factor_case_num = {1}; % The configuration case refer to form_factor.m.
CoM_form_factor_hemisphere_radius = {0.5};
form_factor_case_num = {repmat(1,r_conf_Natoms{1},1)}; % The configuration case refer to form_factor.m.
form_factor_hemisphere_radius = {repmat(0.5,r_conf_Natoms{1},1)};

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Translational Friction %
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Depending on the case (see calc_A function in calculate_sim_params.m),
% different fields of the structure A_struct are expected. For new cases in
% calculate_sim_params, ammend also surface_params.m
% Each parameter needs to be stored either as a single cell (which will be distributed to all
% populations), or as a cell-array (with the i'th element distributed to the i'th
% population)
%
A_case = {1};
A_w0   = {eta};
```

```
A_dw   = {1./tau};
A_eta  = {eta};
A_tau  = {tau};
A_spatial_depended_friction = {0};

%%%%%%%%%%%%%%%%%%%%
% Angular Friction %
%%%%%%%%%%%%%%%%%%%%
% Depending on the case (see calc_A function in calculate_sim_params.m),
% different fields of the structure A_struct are expected.
% Each parameter needs to be stored either as a single cell (which will be distributed to all
% populations), or as a cell-array (with the i'th element distributed to the i'th
% population)
%
A_theta_case = {1};
A_theta_w0   = {eta_theta};
A_theta_dw   = {1};
A_theta_eta  = {eta_theta};
A_theta_tau  = {1};
A_spatial_depended_theta_friction = {0};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Surface-Adsorbate Potential %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For each population, PIGLE generate a potential of up to 4D (which will be called here PES, for potential energy surface).
% PIGLE will generate the potential using a function which is defined by
% the user, with parameters to that function defined by the user as well.

% Pointers to the population specific functions for generating the PES
PES_func_list = {@loadPES};

% Define variables to hold the arguments which are stored in PES_arg_list (see below)
params_for_function_prepare_potential

% Define the arguments for PES generation, i.e. arguments to @loadPES
PES_arg_list = {'loadPES.mat', 'PES4D_AQ'};

%% Parameters for Interactions

prepare_params_for_interactions

% assign functions to species:
% For each pair in f_perm, a function case is defined in f_func. This
% function case is taken from f_interaction.m - and f_func_params contain
% the arguments for each function.
f_perm = [1 1; 1 2; 2 2];
f_func = [repmat(1,1,3)];
```

```
f_func_params = {[fparam1 4],[fparam2 4],[fparam3 4]};

% Define the boundaries for interactions:
% out_cutoff_r - The supercell must be larger than that
    number (see calculate_sim_params.m).
%                TODO: include in connection lists, once
    implemented
% in_cutoff_r -  the force between particles will be
    calculated for r >= r_in
out_cutoff_r = norm(unitcell.celldim)*10*0+49;
in_cutoff_r = 0.1;

% x_interactions - the points in which the force is to be
     calculated
x_min = in_cutoff_r/10; % in Angstrom
x_max = out_cutoff_r + 1; % in Angstrom
numOfPoints_interactions = 500;
x_interactions = linspace(x_min, x_max,
    numOfPoints_interactions); %x_min/max in Angstrom
```

The file pigle_wrapper_params.m deals with the main options that determine what the program should output.

```
%% Wrapper parameters

isISF        = 1; % 0 - just run the simulation (single
    run), 1 - calculate an averaged ISF
isSave       = 0; % save results (on/off)
ISF2save     = [1 2]; % which ISF to save? 1-Incoherent,
    2-Coherent (needs isSave to be turned-on)
toPlot       = 0; %
reduceData   = 2; % 0 - don't reduce. 1 - remove p and
    r_supercell. 2 - Leave only one trajectory for each
    population. 3 - Remove all trajectories from all
    populations
clearParams  = 1; % Clear the structure containing all
    the model-configuration parameters from previous runs
```

For simulating only one particle one must also change the following file by commenting out the second entry:

```
% define a structure to be passed as argument to the
    function which creates the potential

%% POT 1

pot_strct(1).ref_De = [1600 1600 1600 1600 1600 1600]*0;

%pot_strct(1).V = [192 168 ; 0.5 0.5; 0.5 0.5; 18 62 ;
    112 104 ; 156 118 ]*1; % top, slope1,slop2,bridge,hcp
    ,fcc
%pot_strct(1).V = [0 0 ; 0.5 0.5; 0.5 0.5; 80 80 ; 80 80
    ; 80 80 ]*1; % top, slope1,slop2,bridge,hcp,fcc
pot_strct(1).V = [250 ; 0.5 ; 0.5 ; 110 ; 22 ; 0]; % top,
     slope1,slop2,bridge,hcp,fcc

pot_strct(1).is_potval = [1 0 0 1 1 1];
pot_strct(1).a = [1 NaN NaN 1 1 1]*1;
pot_strct(1).r_e = [2 NaN NaN 2 2 2];
pot_strct(1).f_2D = @hexagonal6interp;


```

```

%% POT 2

%pot_strct(2) = pot_strct(1);
%pot_strct(2).V = [exp(15*number_density(2))-1 ; 0.5 ;
    0.5 ; 240 ; 270 ; 280]; % top, slope1,slop2,bridge,
    hcp,fcc
```

With these files edited, one can place the file `loadPES.mat` in the root directory of the PIGLE folder and run `run_pigle.m`. This will run the simulation with the above parameters. To view the motion, one may either run the command `make_movie(params, data)` to produce an animation of the motion or run the following code to plot the full trajectory of the molecule over the simulation:

```
% extract positional and orientational data from pigle
r = data.prtcl.r;

% reshape positional data into rows
rx = reshape(r(1,:,:), [1 length(r)]);
ry = reshape(r(2,:,:), [1 length(r)]);
rz = zeros(size(rx));

% reshape orientational data into a row
rtheta = reshape(r(3,:,:), [1 length(r)]) .* 180/pi;

% plot figure
figure();

% label axes
xlabel('$x$ /\AA'); ylabel('$y$ /\AA');

% use trick with surface() to plot a continuous gradient
    on the trajectory where the colours correspond to
    angles theta
surface([rx;rx], [ry;ry], [rz;rz], [rtheta;rtheta], '
    FaceColor', 'none', 'EdgeColor', 'interp');

% default 2-D view
view(2);

% add colourbar for theta
c = colorbar;

% format colourbar
c.Title.String = '$\theta$ /$^\circ$'; c.Title.
    Interpreter = 'latex';
c.TickLabelInterpreter = 'latex';
```

(b) Having produced this visualisation, do the trajectories fit with the qualitative predictions in the theoretical background? Are the angles at each point along the trajectory as you would expect?

(c) Try varying the following parameters and observe how the trajectory is affected:

- `mass`, the molecular mass

- `radius`, the approximate molecular radius
- `eta`, the translational frictional coupling constant
- `eta_theta`, the rotational frictional coupling constant

(d) Try disabling rotations and observe how this affects the motion. This can be done by setting `theta_enabled=0;` on line 5 of `pigle_ui.m`.

EXTENSION

1. Try plotting the ISF and fitting it to an exponential decay.
2. Plot the dephasing rate of the fit against momentum transfer for both azimuths $\langle 0\,1 \rangle$ and $\langle 1\,1 \rangle$

REFERENCES

1. J. Chem. Phys. 142, 101907 (2015); https://doi.org/10.1063/1.4906048
2. Science 315 (5817), 1391-1393; 10.1126/science.1135302
3. Phys. Rev. Lett. 95, 166101; 10.1103/PhysRevLett.95.166101

APPENDIX

HALVESPACING.M

The function `halvespacing()` is given below. It takes as arguments:

- `V3D`, the computed potential values across the grid defined by `xypos`,
- `xypos`, a grid of positions for an entire WS cell,
- `weight`, the weights of each position in `xypos`,
- `np`, the number of grid points in `xypos`,
- `nscale`, the integer factor N in the theoretical background which determines the grid lattice spacing,
- `a`, the basic lattice spacing,
- `nneb`, the number of neighbours to use in the interpolation.

The function then uses a cubic interpolation scheme to calculate the potential across a grid of half the spacing than the input. This gives us twice the resolution at a much cheaper cost. Given this context the outputs are relatively self-explanatory.

```
function [ V3Dnew, xyposnew, weightnew, npnew, nscalenew ] = halvespacing( V3D, xypos, weight, np,nscale,a,nneb)
% For a hexagonal cell it makes sense to sample it with a grid of points
% of the same symmetry as the basic lattice - and this will be computed
% originally on a lattice with spacing a/nscale if the original lattice
% constant is a. This rouine interpolates between this original grid,
% doubling the point spacing, using a cubic interpolation scheme
% nneb is the number of neighbours to use in the intepolation
%find list of new points needed
%clear xyposreq weightreq
dum10=a;
dumvec0=size(V3D);
nz=dumvec0(2);
del=0.0001*a*a;
a2=a*a/nscale/nscale;
annn2=a2*3;
npnew=0;
radius=a*(1/sqrt(3.0)+2/nscale);
V3Dnew=V3D;
V3Dextra=V3D;
for ip1=1:np-1
    for ip2=ip1+1:np
        dum=(xypos(ip1,1)-xypos(ip2,1))^2+(xypos(ip1,2)-xypos(
            ip2,2))^2;
        if( abs(dum-a2)<del)
            npnew=npnew+1;
            xyposreq(npnew,1)=0.5*(xypos(ip1,1)+xypos(ip2,1));
            xyposreq(npnew,2)=0.5*(xypos(ip1,2)+xypos(ip2,2));
            weightreq(npnew)=1;
        end
        if(abs(dum-annn2)<del) && (weight(ip1)+weight(ip2)<1.1)
            % we have a new edge point
            npnew=npnew+1;
            xyposreq(npnew,1)=0.5*(xypos(ip1,1)+xypos(ip2,1));
            xyposreq(npnew,2)=0.5*(xypos(ip1,2)+xypos(ip2,2));
            weightreq(npnew)=0.5;
        end
    end
end
xyposnew=xypos;
weightnew=weight;
xyposnew(np+1:np+npnew,:)=xyposreq(:,:);
weightnew(np+1:np+npnew)=weightreq(:,:);
nscalenew=nscale*2;
figure(2000)
clf
hold on
plot(xypos(:,1),xypos(:,2),'rx')
plot(xyposreq(:,1),xyposreq(:,2),'go')
sumofrealspaceweightinhalfspacingcell=sum(weight)+sum(weightreq)
title(['orig. pts redx.  new pts green o, interpolation grid
    blue square'])

% this total should be equal to nscalenew^2
%
% we need points outside the original unit cell - so need to add
     points
% from copies of that cell along the 6 sides of the original
    ones
%
% first translate to left and right - one needs all the points
    except the
% corner points along y=0
xyposextra=xypos;
npextra=np;
del=a/nscale*0.01;
for ip=1:np
   if(abs(xypos(ip,1)+a/2)>del)
        % the point is not on the left row of the original cell
            so copy
        % right
        npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)+a;
        xyposextra(npextra,2)=xypos(ip,2);
        if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >
            radius^2)
            npextra=npextra-1;
        else
            V3Dextra(npextra,:)=V3D(ip,:);
        end
    end
    if(abs(xypos(ip,1)-a/2)>del)
        % the point is not on the right row of the original cell
             so copy
        % left
        npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)-a;
        xyposextra(npextra,2)=xypos(ip,2);
        if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 > radius^2)
            npextra=npextra-1;
            else
            V3Dextra(npextra,:)=V3D(ip,:);
        end
    end
    if(weight(ip)==1)
        % the point is not on an edge or corner, so copy other 4
        npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)+a/2;
        xyposextra(npextra,2)=xypos(ip,2)+a*sqrt(3)/2;
                if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >radius^2)
            npextra=npextra-1;
            else
            V3Dextra(npextra,:)=V3D(ip,:);
                end

        npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)+a/2;
        xyposextra(npextra,2)=xypos(ip,2)-a*sqrt(3)/2;
                if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >radius^2)
            npextra=npextra-1;
            else
            V3Dextra(npextra,:)=V3D(ip,:);
                end

        npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)-a/2;
        xyposextra(npextra,2)=xypos(ip,2)+a*sqrt(3)/2;
                if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >radius^2)
            npextra=npextra-1;
            else
            V3Dextra(npextra,:)=V3D(ip,:);
                end


        npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)-a/2;
        xyposextra(npextra,2)=xypos(ip,2)-a*sqrt(3)/2;
                if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >radius^2)
            npextra=npextra-1;
            else
            V3Dextra(npextra,:)=V3D(ip,:);
                end

    end
    if((abs(xypos(ip,1)-a/2)<del) && weight(ip)>0.4)
        % the point is  on the right row of the original cell so copy
        % and is not in a corner - so copy up and down diagonally
        npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)-a/2;
        xyposextra(npextra,2)=xypos(ip,2)+a*sqrt(3)/2;
                if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >radius^2)
            npextra=npextra-1;
            else
            V3Dextra(npextra,:)=V3D(ip,:);
                end

               npextra=npextra+1;
        xyposextra(npextra,1)=xypos(ip,1)-a/2;
        xyposextra(npextra,2)=xypos(ip,2)-a*sqrt(3)/2;
                if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >radius^2)
            npextra=npextra-1;
            else
            V3Dextra(npextra,:)=V3D(ip,:);
                end

    end
end
xyposextrasize=size(xyposextra);
V3Dextrasize=size(V3Dextra);
plot(xyposextra(:,1),xyposextra(:,2),'bs')
% we now have a candidate neighbours - npextra of them, xyposextra in
% position with values of potential stored in V3Dextra and we are looking
% to find potential values at the npnew points listed in xyposreq. V3Dnew
% already has the existing np points loaded into it.
%
% now find nearest neighbours of all required points
distance2=zeros(npnew,npextra);
distance=zeros(npnew,npextra,2);
dumvec=linspace(0,0,npextra);
ipextraindex=linspace(0,0,npextra);
distance0=linspace(0,0,nneb);
distancetemp=linspace(0,0,nneb);
D=linspace(0,0,nneb);
sigma=linspace(1,1,nneb);
sigma(5:nneb)=100;
tol=a/nscale*0.01;
iflag=0;
ni=10;
f=zeros(ni,nneb);
for ipnew=1:npnew
    for ipextra=1:npextra
        distance(ipnew,ipextra,:)=xyposextra(ipextra,:)-xyposreq(ipnew,:);
        distance2(ipnew,ipextra)=distance(ipnew,ipextra,1)^2+distance(ipnew,ipextra,2)^2;
    end
    [dumvec,ipextraindex]=sort(squeeze(distance2(ipnew,:)));
    distance2(ipnew,:)=dumvec;
    distance(ipnew,:,1)=squeeze(distance(ipnew,ipextraindex,1));
    distance(ipnew,:,2)=squeeze(distance(ipnew,ipextraindex,2));
    % so now have points in order - ipextraindex has indices of the
    % ipextras
    % now calculate
    distancetemp=sqrt(squeeze(distance2(ipnew,1:nneb)));
     if(iflag==0)
         iflag=1;
         distance0=distancetemp;
     else
         if (sum(abs(distancetemp-distance0))>tol)
             xohhelp=1
         end
     end
% pause on
% figure(2000)
% clf
% hold on
% plot(xyposextra(:,1),xyposextra(:,2),'rx')
% plot(xyposreq(:,1),xyposreq(:,2),'go')
% plot(xyposreq(ipnew,1),xyposreq(ipnew,2),'ro')
% for ij=1:nneb
%     plot(xyposextra(ipextraindex(ij),1),xyposextra(
    ipextraindex(ij),2),'bs')
% end
% pause;
% pause off
     for ij=1:nneb
         dx=distance(ipnew,ij,1);
         dy=distance(ipnew,ij,2);
         % index of neighbour
         f(1,ij)=1;
         f(2,ij)=dx;
         f(3,ij)=dy;
         f(4,ij)=dx*dx;
         f(5,ij)=dx*dy;
         f(6,ij)=dy*dy;
         f(7,ij)=dx*dx*dx;
         f(8,ij)=dx*dx*dy;
         f(9,ij)=dx*dy*dy;
         f(10,ij)=dy*dy*dy;
     end

     M=zeros(ni,ni);
     for ik=1:ni
         for ii=1:ni
             for ij=1:nneb
                 M(ik,ii)=M(ik,ii)+f(ik,ij)*f(ii,ij)/sigma(ij)
                     ^2;
             end
         end
     end
     pause on
     for iz=1:nz
% figure(2001)
% clf
% hold on
% % xlim([xyposreq(ipnew,1)-2*a/nscale,xyposreq(ipnew,1)+2*a/
    nscale])
% % ylim([xyposreq(ipnew,2)-2*a/nscale,xyposreq(ipnew,2)+2*a/
    nscale])
% plot3(xyposextra(:,1),xyposextra(:,2),V3Dextra(:,iz),'rx')
%     plot(xyposreq(ipnew,1),xyposreq(ipnew,2),'go')
%
%     for ij=1:nneb
%     plot3(xyposreq(ipnew,1)+distance(ipnew,ij,1),xyposreq(
    ipnew,2)+distance(ipnew,ij,2),V3Dextra(ipextraindex(ij),iz)
    ,'bs')
%     end
% pause
         for ij=1:nneb
             D(ij)=V3Dextra(ipextraindex(ij),iz);
         end
         B=linspace(0,0,ni);
         for ik=1:ni
             for ij=1:nneb
                 B(ik)=B(ik)+D(ij)*f(ik,ij)/sigma(ij)^2;
             end
         end
         Avec=B/M;
         V3Dnew(np+ipnew,iz)=Avec(1);
     end
     pause off
end
npnew=npnew+np;


end
```

INHEXRECIP.M

The function `InHexRecip()` is given below. It takes as arguments:

- `G`, any given reciprocal lattice vector,
- `G1`, the spacing of the reciprocal lattice,
- `n`, the integer factor N in the theoretical background which determines the grid lattice spacing.

It then assigns a weight to the reciprocal lattice vector `G` based on its position in the WS cell. Bulk points are assigned a weight of 1, edge points are assigned 1/2 and corner points 1/3.

```
function weight=InHexRecip(G,G1,n)
x=G(1);
y=G(2);
del=G1/1e4;
r3=sqrt(3);
weight=0;
if ( (y+del>-n*G1/2) && (y-del<n*G1/2) && (y-del<n*G1-r3*x) && (
    y+del>r3*x-n*G1) && (y+del>-r3*x-n*G1) && (y-del<r3*x+n*G1))
% we have a point inside the Wigner-Seitz cell
    weight=1;
% now need to check if its on an edge or corner

    dum=0;
    if( abs(y+n*G1/2)<del)
        dum=dum+1;
    end

    if( abs(y-n*G1/2)<del)
        dum=dum+1;
    end

    if( abs(y-n*G1+r3*x)<del)
        dum=dum+1;
    end

    if( abs(y-r3*x+n*G1)<del)
        dum=dum+1;
    end

    if( abs(y+r3*x+n*G1)<del)
        dum=dum+1;
    end

    if( abs(y-r3*x-n*G1)<del)
        dum=dum+1;
    end
    if(dum==1)
        % its on an edge
        weight=1/2;
    end
    if(dum==2)
        % its at a corner
        weight=1/3;
    end

end
```

FILE1.M

```
% Potential for 157 pt. script
%% parameters

num_ang=24; % number of angles
num_pos=157; % number of potential positions
potential_mat=zeros(num_pos,num_ang);
new_file='Potential.mat';

%% position values

% top1=600;
% top2=470;
% top3=370;
% top4=270;

top1new=1000;
top2new=900;
top3new=800;
top4new=700;
top5new=600;
top6new=550;
top7new=500;
top8new=450;
top9new=400;
% top10new=600;
% top11new=600;
% top12new=600;

nearfccmin=85;
nearfccmax=250;

nearhcpmin=5;
nearhcpmax=65;
```

```
nearhcphighmax=100;

hcpmin=0;
hcpmax=60;
hcphighmax=80;

fccmin=75;
fccmax=210;

bridgemin=35;
bridgemax=300;

bridgebarmin=90;
bridgebarmax=340;

nearbridgebarmin=65; % positions near bridge sites
nearbridgebarmax=360; % positions near bridge sites

otherpoints=190;

%% positions

% top1pos=[1];
% top2pos=[2,6,136,114,91,84,62,32];
% top3pos=[3,7,10,13,140,110,118,124,100,95,88,58,66,39,36,33];
% top4pos
    =[4,8,11,14,17,146,143,137,115,121,127,131,106,103,98,92,85,63,69,72,43,40,37,34];

%
% topnewpos=[1,   6,136,114,84,62,32,   2,13,110,91,58,39,
    10,140,118,88,66,36,
    7,17,146,137,115,124,95,85,63,72,43,33,
    3,14,22,143,111,121,100,92,59,69,48,40,
    11,20,149,141,119,127,98,89,67,75,46,37,
    8,18,25,153,147,138,116,125,131,103,96,86,64,73,79,51,44,34,

    4,15,23,28,151,144,112,122,129,106,101,93,60,70,77,54,49,41,

    12,21,27,155,150,142,120,128,133,105,99,90,68,76,81,53,47,38,

    9,19,26,30,157,154,148,139,117,126,132,135,108,104,97,87,65,74,80,83,56,52,45,35,

    5,16,24,29,31,42,50,55,57,61,71,78,82,94,102,107,109,113,123,130,134,145,152,156];


topnewpos1=[1];
topnewpos2=[6,136,114,84,62,32];
topnewpos3=[2,13,110,91,58,39];
topnewpos4=[10,140,118,88,66,36];
topnewpos5=[7,17,146,137,115,124,95,85,63,72,43,33];
topnewpos6=[3,14,22,143,111,121,100,92,59,69,48,40];
topnewpos7=[11,20,149,141,119,127,98,89,67,75,46,37];
topnewpos8
    =[8,18,25,153,147,138,116,125,131,103,96,86,64,73,79,51,44,34];

topnewpos9
    =[4,15,23,28,151,144,112,122,129,106,101,93,60,70,77,54,49,41];

% topnewpos10
    =[12,21,27,155,150,142,120,128,133,105,99,90,68,76,81,53,47,38];
```

```
% topnewpos11
    =[9,19,26,30,157,154,148,139,117,126,132,135,108,104,97,87,65,74,80,83,56,52,45,35];
% topnewpos12
    =[5,16,24,29,31,42,50,55,57,61,71,78,82,94,102,107,109,113,123,130,134,145,152,156];

nearhcppos=[35,9,65,87,117,139];
nearfccpos=[30,157,83,56,108,135];

hcppos=[5,61,113];
fccpos=[31,57,109];

bridge1pos=[50,130];
bridge2pos=[24,102];
bridge3pos=[78,152];

bridge1barpos=[42,55,123,134];
bridge2barpos=[16,29,94,107];
bridge3barpos=[71,82,145,156];

nearbridge1pos=[45,47,52,126,128,132];
nearbridge2pos=[19,21,26,97,99,104];
nearbridge3pos=[74,76,80,148,150,154];

for i=1:num_ang/2
    for j=1:size(topnewpos1,2)
        pos_temp=topnewpos1(j);
        potential_mat(pos_temp,i)=top1new;
    end
    for j=1:size(topnewpos2,2)
        pos_temp=topnewpos2(j);
        potential_mat(pos_temp,i)=top2new;
    end
    for j=1:size(topnewpos3,2)
        pos_temp=topnewpos3(j);
        potential_mat(pos_temp,i)=top3new;
    end
    for j=1:size(topnewpos4,2)
        pos_temp=topnewpos4(j);
        potential_mat(pos_temp,i)=top4new;
    end
    for j=1:size(topnewpos5,2)
        pos_temp=topnewpos5(j);
        potential_mat(pos_temp,i)=top5new;
    end
    for j=1:size(topnewpos6,2)
        pos_temp=topnewpos6(j);
        potential_mat(pos_temp,i)=top6new;
    end
    for j=1:size(topnewpos7,2)
        pos_temp=topnewpos7(j);
        potential_mat(pos_temp,i)=top7new;
    end
    for j=1:size(topnewpos8,2)
        pos_temp=topnewpos8(j);
        potential_mat(pos_temp,i)=top8new;
    end
    for j=1:size(topnewpos9,2)
        pos_temp=topnewpos9(j);
        potential_mat(pos_temp,i)=top9new;
    end
%     for j=1:size(topnewpos10,2)
%         pos_temp=topnewpos10(j);
%         potential_mat(pos_temp,i)=top10new;
%     end
%     for j=1:size(topnewpos11,2)
%         pos_temp=topnewpos11(j);
%         potential_mat(pos_temp,i)=top11new;
%     end
%     for j=1:size(topnewpos12,2)
%         pos_temp=topnewpos12(j);
%         potential_mat(pos_temp,i)=top12new;
%     end
end

% topnewpos=[1,   6,136,114,84,62,32,   2,13,110,91,58,39,
    10,140,118,88,66,36,
    3,14,22,143,111,121,100,92,59,69,48,40,
    11,20,149,141,119,127,98,89,67,75,46,37,
    8,18,25,153,147,138,116,125,131,103,96,86,64,73,79,51,44,34,

    4,15,23,28,151,144,112,122,129,106,101,93,60,70,77,54,49,41,

    12,21,27,155,150,142,120,128,133,105,99,90,68,76,81,53,47,38,

    9,19,26,30,157,154,148,139,117,126,132,135,108,104,97,87,65,74,80,83,56,52,45,35];
     otherpoints=[];
% for i=1:157
%     if sum(ismember([i],topnewpos))==1;
%         continue
%     else
%         sizeotherpoints=size(otherpoints,2);
%         otherpoints(sizeotherpoints+1)=i;
%     end
% end

%% angles

% all angles - top sites same for all angles

% for i=1:num_ang/2
%     for j=1:size(top1pos,2)
%         pos_temp=top1pos(j);
%         potential_mat(pos_temp,i)=top1;
%     end
%     for j=1:size(top2pos,2)
%         pos_temp=top2pos(j);
%         potential_mat(pos_temp,i)=top2;
%     end
%     for j=1:size(top3pos,2)
%         pos_temp=top3pos(j);
%         potential_mat(pos_temp,i)=top3;
%     end
%     for j=1:size(top4pos,2)
%         pos_temp=top4pos(j);
%         potential_mat(pos_temp,i)=top4;
%     end
% end

%%%%% 0, 60, 120 degrees (first inputting 0 degrees values)

```

```
%%% hcp, fcc

for i=1:size(hcppos,2)
    potential_mat(hcppos(i),1)=hcpmin;
end

for i=1:size(fccpos,2)
    potential_mat(fccpos(i),1)=fccmin;
end

%%% near hcp, fcc

for i=1:size(nearhcppos,2)
    potential_mat(nearhcppos(i),1)=nearhcpmin;
end

for i=1:size(nearfccpos,2)
    potential_mat(nearfccpos(i),1)=nearfccmin;
end

%%% bridge sites, barriers, near bridge sites

% bridge 1

for i=1:size(bridge1pos,2)
    potential_mat(bridge1pos(i),1)=bridgemax;
end

for i=1:size(bridge1barpos,2)
    potential_mat(bridge1barpos(i),1)=bridgebarmax;
end

for i=1:size(nearbridge1pos,2)
    potential_mat(nearbridge1pos(i),1)=nearbridgebarmax;
end

% bridge 2

for i=1:size(bridge2pos,2)
    potential_mat(bridge2pos(i),1)=bridgemax;
end

for i=1:size(bridge2barpos,2)
    potential_mat(bridge2barpos(i),1)=bridgebarmax;
end

for i=1:size(nearbridge2pos,2)
    potential_mat(nearbridge2pos(i),1)=nearbridgebarmax;
end

% bridge 3

for i=1:size(bridge3pos,2)
    potential_mat(bridge3pos(i),1)=bridgemax;
end

for i=1:size(bridge3barpos,2)
    potential_mat(bridge3barpos(i),1)=bridgebarmax;
end

for i=1:size(nearbridge3pos,2)
```

```
    potential_mat(nearbridge3pos(i),1)=nearbridgebarmax;
end

%%% Same values for 60, 120 degrees

potential_mat(:,5)=potential_mat(:,1);
potential_mat(:,9)=potential_mat(:,1);

%%%%% 30, 90, 150 degrees (first inputting 30 degrees values)

%%% hcp, fcc

for i=1:size(hcppos,2)
    potential_mat(hcppos(i),3)=hcphighmax;
end

for i=1:size(fccpos,2)
    potential_mat(fccpos(i),3)=fccmax;
end

%%% near hcp, fcc

for i=1:size(nearhcppos,2)
    potential_mat(nearhcppos(i),3)=nearhcphighmax;
end

for i=1:size(nearfccpos,2)
    potential_mat(nearfccpos(i),3)=nearfccmax;
end

%%% bridge sites, barriers, near bridge sites

% bridge 1

for i=1:size(bridge1pos,2)
    potential_mat(bridge1pos(i),3)=bridgemax;
end

for i=1:size(bridge1barpos,2)
    potential_mat(bridge1barpos(i),3)=bridgebarmax;
end

for i=1:size(nearbridge1pos,2)
    potential_mat(nearbridge1pos(i),3)=nearbridgebarmax;
end

% bridge 2

for i=1:size(bridge2pos,2)
    potential_mat(bridge2pos(i),3)=bridgemax;
end

for i=1:size(bridge2barpos,2)
    potential_mat(bridge2barpos(i),3)=bridgebarmax;
end

for i=1:size(nearbridge2pos,2)
    potential_mat(nearbridge2pos(i),3)=nearbridgebarmax;
end

% bridge 3
```

```

for i=1:size(bridge3pos,2)
    potential_mat(bridge3pos(i),3)=bridgemax;
end

for i=1:size(bridge3barpos,2)
    potential_mat(bridge3barpos(i),3)=bridgebarmax;
end

for i=1:size(nearbridge3pos,2)
    potential_mat(nearbridge3pos(i),3)=nearbridgebarmax;
end

%%% Same values for 90, 150 degrees

potential_mat(:,7)=potential_mat(:,3);
potential_mat(:,11)=potential_mat(:,3);

%%%%% 15, 45 degrees (first inputting 15 degrees values)

%%% hcp, fcc

for i=1:size(hcppos,2)
    potential_mat(hcppos(i),2)=hcpmax;
end

for i=1:size(fccpos,2)
    potential_mat(fccpos(i),2)=fccmax;
end

%%% near hcp, fcc

for i=1:size(nearhcppos,2)
    potential_mat(nearhcppos(i),2)=nearhcpmax;
end

for i=1:size(nearfccpos,2)
    potential_mat(nearfccpos(i),2)=nearfccmax;
end

%%% bridge sites, barriers, near bridge sites

% bridge 1

for i=1:size(bridge1pos,2)
    potential_mat(bridge1pos(i),2)=bridgemin;
end

for i=1:size(bridge1barpos,2)
    potential_mat(bridge1barpos(i),2)=bridgebarmin;
end

for i=1:size(nearbridge1pos,2)
    potential_mat(nearbridge1pos(i),2)=nearbridgebarmin;
end

% bridge 2

for i=1:size(bridge2pos,2)
    potential_mat(bridge2pos(i),2)=bridgemax;
end
```

```

for i=1:size(bridge2barpos,2)
    potential_mat(bridge2barpos(i),2)=bridgebarmax;
end

for i=1:size(nearbridge2pos,2)
    potential_mat(nearbridge2pos(i),2)=nearbridgebarmax;
end

% bridge 3

for i=1:size(bridge3pos,2)
    potential_mat(bridge3pos(i),2)=bridgemax;
end

for i=1:size(bridge3barpos,2)
    potential_mat(bridge3barpos(i),2)=bridgebarmax;
end

for i=1:size(nearbridge3pos,2)
    potential_mat(nearbridge3pos(i),2)=nearbridgebarmax;
end

%%% Same values for 45 degrees

potential_mat(:,4)=potential_mat(:,2);

%%%%% 135, 165 degrees (first inputting 135 degrees values)

%%% hcp, fcc

for i=1:size(hcppos,2)
    potential_mat(hcppos(i),10)=hcpmax;
end

for i=1:size(fccpos,2)
    potential_mat(fccpos(i),10)=fccmax;
end

%%% near hcp, fcc

for i=1:size(nearhcppos,2)
    potential_mat(nearhcppos(i),10)=nearhcpmax;
end

for i=1:size(nearfccpos,2)
    potential_mat(nearfccpos(i),10)=nearfccmax;
end

%%% bridge sites, barriers, near bridge sites

% bridge 1

for i=1:size(bridge1pos,2)
    potential_mat(bridge1pos(i),10)=bridgemax;
end

for i=1:size(bridge1barpos,2)
    potential_mat(bridge1barpos(i),10)=bridgebarmax;
end

```

```
for i=1:size(nearbridge1pos,2)
    potential_mat(nearbridge1pos(i),10)=nearbridgebarmax;
end

% bridge 2

for i=1:size(bridge2pos,2)
    potential_mat(bridge2pos(i),10)=bridgemin;
end

for i=1:size(bridge2barpos,2)
    potential_mat(bridge2barpos(i),10)=bridgebarmin;
end

for i=1:size(nearbridge2pos,2)
    potential_mat(nearbridge2pos(i),10)=nearbridgebarmin;
end

% bridge 3

for i=1:size(bridge3pos,2)
    potential_mat(bridge3pos(i),10)=bridgemax;
end

for i=1:size(bridge3barpos,2)
    potential_mat(bridge3barpos(i),10)=bridgebarmax;
end

for i=1:size(nearbridge3pos,2)
    potential_mat(nearbridge3pos(i),10)=nearbridgebarmax;
end

%%% Same values for 165 degrees

potential_mat(:,12)=potential_mat(:,10);

%%%%% 75, 105 degrees (first inputting 75 degrees values)

%%% hcp, fcc

for i=1:size(hcppos,2)
    potential_mat(hcppos(i),6)=hcpmax;
end

for i=1:size(fccpos,2)
    potential_mat(fccpos(i),6)=fccmax;
end

%%% near hcp, fcc

for i=1:size(nearhcppos,2)
    potential_mat(nearhcppos(i),6)=nearhcpmax;
end

for i=1:size(nearfccpos,2)
    potential_mat(nearfccpos(i),6)=nearfccmax;
end

%%% bridge sites, barriers, near bridge sites

% bridge 1
```

```

for i=1:size(bridge1pos,2)
    potential_mat(bridge1pos(i),6)=bridgemax;
end

for i=1:size(bridge1barpos,2)
    potential_mat(bridge1barpos(i),6)=bridgebarmax;
end

for i=1:size(nearbridge1pos,2)
    potential_mat(nearbridge1pos(i),6)=nearbridgebarmax;
end

% bridge 2

for i=1:size(bridge2pos,2)
    potential_mat(bridge2pos(i),6)=bridgemax;
end

for i=1:size(bridge2barpos,2)
    potential_mat(bridge2barpos(i),6)=bridgebarmax;
end

for i=1:size(nearbridge2pos,2)
    potential_mat(nearbridge2pos(i),6)=nearbridgebarmax;
end

% bridge 3

for i=1:size(bridge3pos,2)
    potential_mat(bridge3pos(i),6)=bridgemin;
end

for i=1:size(bridge3barpos,2)
    potential_mat(bridge3barpos(i),6)=bridgebarmin;
end

for i=1:size(nearbridge3pos,2)
    potential_mat(nearbridge3pos(i),6)=nearbridgebarmin;
end

%%% Same values for 105 degrees

potential_mat(:,8)=potential_mat(:,6);


%%%%% All other points on matrix

for i=1:num_pos
    for j=1:num_ang
        if potential_mat(i,j)==0
            potential_mat(i,j)=otherpoints;
        end
    end
end

potential_mat(:,((num_ang/2)+1):num_ang)=potential_mat(:,1:(
    num_ang/2));

%%%%% Create new file

```

```
V3Dinterp_AQ=potential_mat;
general_file='Potential_AQ_base_file.mat';
copyfile(general_file, new_file);
save(new_file, 'V3Dinterp_AQ', '-append');
clear;
```

FILE2.M

```
clear;
% --------------------------------------------------------------------------
% load in Potential.mat
% --------------------------------------------------------------------------

load('Potential.mat');
% contains: - matrix of grid point (x,y)-positions
%           - vector of AQ angles
%           - vector of weights for each point
%           - matrix of sampled potentials (i.e. Fourier components at each point and each angle)

% --------------------------------------------------------------------------
% plot the locations of the 157 grid points
% --------------------------------------------------------------------------

% Plotting of 157 pts.
figure; hold on; axis equal;
x_157=xpos(:,1); y_157=xypos(:,2);
for i=1:size(xypos,1)
    plot (x_157(i),y_157(i),'.')
    text(x_157(i),y_157(i),num2str(i));
end



% Number of Interpolation Points for rectangular grid & angle interpolation
nx=120;
ny=200;
ntheta=25; % to request 30 theta vals., starting at 0, e.g., set ntheta=31

%% define range over which potential will be calculated
% potential is calculated across region (x,y) \in ([0,ax],[0,ay])

% the maximum x-displacement of the WS cell is half the basic horizontal lattice spacing
ax=2*max(xypos(:,1));

% the maximum y-displacement of the WS cell is half the basic vertical lattice spacing
ay=3*max(xypos(:,2));
```

```

% axes are typically chosen such that dir(x) || dir(a1)
a=ax;

% number of points in non-interpolated WS cell
np=157;

V3Dinterp=V3Dinterp_AQ;

%%
    ------------------------------------------------------------------------
% interpolating in theta
%
    ------------------------------------------------------------------------

% intitial number of angles
ntheta0=length(thetapos);

% generate new linearly-spaced vector from [0,2*pi) with ntheta
    -1 entries
thetavec=linspace(0,2*pi,ntheta); thetavec=thetavec(1:(ntheta-1)
    );

ntheta=ntheta-1;

% use cubic interpolation with spline() function to interpolate
    in theta dimension for each position (x,y)
for ixy=1:(size(xypos,1))
    V3Dinterp_temp(ixy,:)=spline(thetapos,V3Dinterp(ixy,:),
        thetavec); % spline() uses cubic interpolation
end
V3Dinterp=V3Dinterp_temp;

%%
    ------------------------------------------------------------------------
% interpolating in x and y
%
    ------------------------------------------------------------------------

% original integer factor N as defined in the theoretical bckgd
nscale=12;

% use halvespacing() method to double the resolution of
    V3Dinterp
[V3Dnew, xyposnew, weightnew, npnew, nscalenew] = halvespacing(
    V3Dinterp, xypos, weight, np,nscale,a,14);

%%
    ------------------------------------------------------------------------
% interpolating in x and y
%-------------------------------------------

% basic vertical reciprocal lattice spacing
G1=4*pi/sqrt(3)/a;

% basic vertical reciprocal lattice spacing
```

```
Gx0=2*pi/ax;

% basic vertical reciprocal lattice spacing
Gy0=2*pi/ay;

%this gives twice the point spacing in the original grid

% iGp is the number of mesh cells in real space to lattice cell along one direction
iGp=0;

% iterate through desired no of points
for ix=1:nx
    for iy=1:ny
        if(mod(ix+iy,2)==0)

            % place origin at centre of grid by limiting range to +/- nx or ny:

            % limit x
            ixind=ix-1;
            if(ixind>nx/2)
                ixind=ixind-nx;
            end

            % limit y
            iyind=iy-1;
            if(iyind>ny/2)
               iyind=iyind-ny;
            end

            % find corresponding reciprocal vector for given posn, Gtemp
            Gtemp(1)=ixind*Gx0; Gtemp(2)=iyind*Gy0;

            % weight that position based on its position within the range
            weighttemp=InHexRecip(Gtemp,G1,nscalenew);

            % if the point is within the basic reciprocal WS cell... (weight=1,1/2,1/3)
           if(weighttemp~=0)
                % add the point to the set of considered reciprocal vectors
                iGp=iGp+1;

                Gxp(iGp)=ixind*Gx0; Gyp(iGp)=iyind*Gy0;

                weightp(iGp)=weighttemp;
            end
        end
    end
end

% total no of reciprocal grid vectors in basic reciprocal WS cell
nGp=iGp;

% set of moduli of reciprocal grid lattice vectors
Gmag=(Gxp.*Gxp+Gyp.*Gyp);

```

```
% sort Gmag and store the permutatiion of the old order to the
    new order in IX
[Gmag, IX]=sort(Gmag);

% we can now use IX to reorder the points (Gxp,Gyp) by their
    modulus (reorder all their properties)
Gdum=Gxp(IX); Gxp=Gdum;
Gdum=Gyp(IX); Gyp=Gdum;

Gdum=weightp(IX); weightp=Gdum;

% Normalise the (ixGp,iyGp) to integers defining the reciprocal
    vectors
ixGp=round(Gxp./Gx0);
iyGp=round(Gyp./Gy0);

% Switch to 1-indexing for use in the Fourier transform
ix0Gp=ixGp+1;
iy0Gp=iyGp+1;

% add multiples of nx,ny to map all points (ix0Gp,iy0Gp)<1, i.e.
     points outside the range [1,nx] & [1,ny] to 1,2,...
for iGp=1:nGp
    if(ix0Gp(iGp)<1)
        ix0Gp(iGp)=ix0Gp(iGp)+nx;
    end
    if(iy0Gp(iGp)<1)
        iy0Gp(iGp)=iy0Gp(iGp)+ny;
    end
end

% Gxp,Gyp are the x and y components of the nGp reciprocal
    vectors used to describe the potential.

% ixGp and iyGp give these vectors as integer multiples of Gx0
    and Gy0 which are the basic G vectors of the rectangular
    unit cell

% ix0Gp and iy0Gp are these integers, but mapped on to the range
     1 to nx and 1 to ny where (1,1) is the (Gx,Gy)=(0,0) - i.e.
     these are the integers to use in the fourier transforms

% so now ix0Gp and iy0Gp have the integers that will locate each
     G vector in the matrix for a iff2'ed potential

%%
    ---------------------------------------------------------------------------

% calculate potential  Fourier components for interpolated
    potential
%
    ---------------------------------------------------------------------------

%VG(iGp,iz)
VGnew=zeros(nGp,ntheta);
for iGp=1:nGp
    for itheta=1:ntheta
        dum=0.0;
        for ip=1:npnew  % sum weighted contributions of each
            Fourier component
    dum=dum+V3Dnew(ip,itheta)*exp(-1i*(Gxp(iGp)*xyposnew(ip,1)+
        Gyp(iGp)*xyposnew(ip,2)))*weightnew(ip);
        end
        VGnew(iGp,itheta)=dum/nscalenew/nscalenew;
    end
end

%%
    ------------------------------------------------------------------------
% weight the potential
%
    ------------------------------------------------------------------------

pe3D=zeros(nx,ny,ntheta);

for itheta=1:ntheta

clear A
    A=zeros(nx,ny);
    for iGp=1:nGp
        % weight the potential at each point to account for
            double/treble-counting
        A(ix0Gp(iGp),iy0Gp(iGp))=VGnew(iGp,itheta)*weightp(iGp);
    end
    % take the real (cosine) part so that there is a top site at
         the origin
    B=real(fft2(A));
pe3D(:,:,itheta)=B(:,:);
end

%%
    ------------------------------------------------------------------------
% produce grid of positions we wish to sample in our plot, as
    defined by nx and ny
%
    ------------------------------------------------------------------------

% Note we calculate it across a rectangle cell rather than a WS
    cell for ease of use
X=zeros(nx,ny);
Y=X;
for ix=1:nx
    for iy=1:ny
        X(ix,iy)=ax*(ix-1)/nx;
        Y(ix,iy)=ay*(iy-1)/ny;
    end
end

%%
    ------------------------------------------------------------------------
% plot potentials for each calculated theta in [0,180] degrees
%
    ------------------------------------------------------------------------

% define plot parameters
color_scale_min=0; color_scale_max=1050;
```

```
asp = [1 1 1/color_scale_max];
Lx = [0 max(X,[],'all')]; Ly = [0 max(Y,[],'all')];

% plot surface/contour map for each calculated angle theta in [0,180] degrees
for i = 1:24
    figure(100+i);
    xlim(Lx); ylim(Ly);

    % surf plot
    % stest = surf(X,Y,pe3D(:,:,i*3-2)); set(stest,'LineStyle','none'); daspect(asp); caxis([color_scale_min color_scale_max]); view([0 90]);

    % contour plot
    stest = contourf(X,Y,pe3D(:,:,i*3-2)); axis equal;

    % export graphics to png files
    exportgraphics(gcf,sprintf('deg%d.png',round(thetavec(i)*3*(180/pi))),'Resolution','300');
end

%% --------------------------------------------------------------------------
% prepare potential object for implementation in MD simulation
% --------------------------------------------------------------------------

% Including 3rd z dimension BUT only with length 1 because we defined a single corrugation surface z(x,y)
PES4D_AQ(:,:,1,:)=pe3D;

%permute axes t: y,x,z,theta
PES4D_AQ=permute(PES4D_AQ,[2 1 3 4]);

% save potential in file 'loadPES.mat'
save('loadPES', 'PES4D_AQ');
```