ASSIGNMENT 13: SOLUTION

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

Written in XeLaTeX

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.

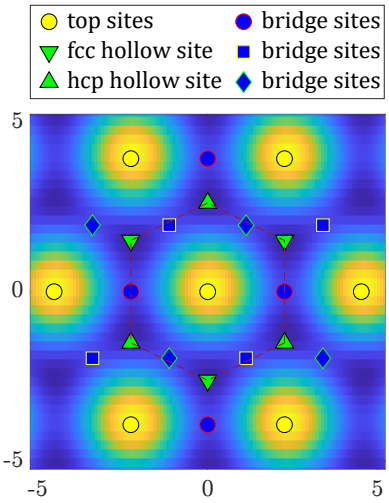

Figure 1 | A labelled plot of the corrugation as computed in the hard-wall corrugation assignment.

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,

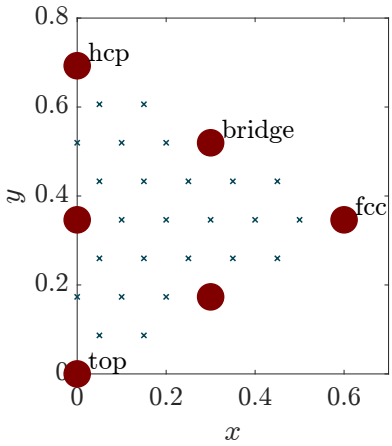

Figure 2 | The large red circles are the only points required to differentiate between hollow sites. However, additional points have been included to provide a more general procedure, giving a total of 31 points.

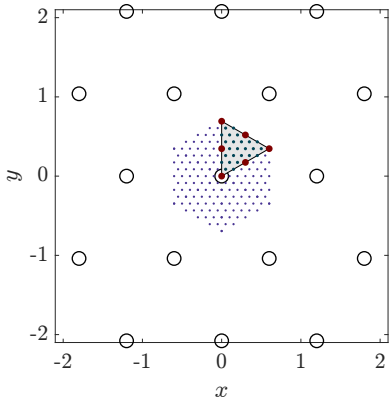

Figure 3 | Depiction of the real space lattice along with the irreducible triangle (shaded) and the set of points encompassed by the WS cell.

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 ±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 ±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 motion 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.

```
1  %% parameters for config_model.m
2
3  z_enabled = 0; % Enable/Disable (1/0) motion
       perpendicular to the surface
4  dKz_include_in_isf = 0; % Enable/Disable (1/0)
       calculation of the ISF for perpendicular momentum
       transfer
5  theta_enabled = 1; % Enable/Disable (1/0) rotation of
       rigid body (rotation axis perpendicular to the
       surface)
6  zero_p_init = 0; % set initial momentum be set to zero? (
       if set to 0, p_init will correspond to thermal
       distribution)
7  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};
```

```
55  A_dw   = {1./tau};
56  A_eta  = {eta};
57  A_tau  = {tau};
58  A_spatial_depended_friction = {0};
59
60  %%%%%%%%%%%%%%%%%%%%
61  % Angular Friction %
62  %%%%%%%%%%%%%%%%%%%%
63  % Depending on the case (see calc_A function in
        calculate_sim_params.m),
64  % different fields of the structure A_struct are expected
        .
65  % Each parameter needs to be stored either as a single
        cell (which will be distributed to all
66  % populations), or as a cell-array (with the i'th element
         distributed to the i'th
67  % population)
68  %
69  A_theta_case = {1};
70  A_theta_w0   = {eta_theta};
71  A_theta_dw   = {1};
72  A_theta_eta  = {eta_theta};
73  A_theta_tau  = {1};
74  A_spatial_depended_theta_friction = {0};
75
76  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
77  % Surface-Adsorbate Potential %
78  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
79
80  % For each population, PIGLE generate a potential of up
        to 4D (which will be called here PES, for potential
        energy surface).
81  % PIGLE will generate the potential using a function
        which is defined by
82  % the user, with parameters to that function defined by
        the user as well.
83
84  % Pointers to the population specific functions for
        generating the PES
85  PES_func_list = {@loadPES};
86
87  % Define variables to hold the arguments which are stored
         in PES_arg_list (see below)
88  params_for_function_prepare_potential
89
90  % Define the arguments for PES generation, i.e. arguments
        to @loadPES
91  PES_arg_list = {'loadPES.mat', 'PES4D_AQ'};
92
93  %% Parameters for Interactions
94
95  prepare_params_for_interactions
96
97  % assign functions to species:
98  % For each pair in f_perm, a function case is defined in
        f_func. This
99  % function case is taken from f_interaction.m - and
        f_func_params contain
100 % the arguments for each function.
101 f_perm = [1 1; 1 2; 2 2];
102 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;


```

```
17
18 %% POT 2
19
20 %pot_strct(2) = pot_strct(1);
21 %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:

```
1  % extract positional and orientational data from pigle
2  r = data.prtcl.r;
3
4  % reshape positional data into rows
5  rx = reshape(r(1,:,:), [1 length(r)]);
6  ry = reshape(r(2,:,:), [1 length(r)]);
7  rz = zeros(size(rx));
8
9  % reshape orientational data into a row
10 rtheta = reshape(r(3,:,:), [1 length(r)]) .* 180/pi;
11
12 % plot figure
13 figure();
14
15 % label axes
16 xlabel('$x$ /\AA'); ylabel('$y$ /\AA');
17
18 % use trick with surface() to plot a continuous gradient
       on the trajectory where the colours correspond to
       angles theta
19 surface([rx;rx], [ry;ry], [rz;rz], [rtheta;rtheta], '
       FaceColor', 'none', 'EdgeColor', 'interp');
20
21 % default 2-D view
22 view(2);
23
24 % add colourbar for theta
25 c = colorbar;
26
27 % format colourbar
28 c.Title.String = '$\theta$ /$^\circ$'; c.Title.
       Interpreter = 'latex';
29 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.

```
1  function [ V3Dnew, xyposnew, weightnew, npnew, nscalenew ] =
       halvespacing( V3D, xypos, weight, np,nscale,a,nneb)
2  % For a hexagonal cell it makes sense to sample it with a grid
       of points
3  % of the same symmetry as the basic lattice - and this will be
       computed
4  % originally on a lattice with spacing a/nscale if the original
       lattice
5  % constant is a. This rouine interpolates between this original
       grid,
6  % doubling the point spacing, using a cubic interpolation scheme
7  % nneb is the number of neighbours to use in the intepolation
8  %find list of new points needed
9  %clear xyposreq weightreq
10 dum10=a;
11 dumvec0=size(V3D);
12 nz=dumvec0(2);
13 del=0.0001*a*a;
14 a2=a*a/nscale/nscale;
15 annn2=a2*3;
16 npnew=0;
17 radius=a*(1/sqrt(3.0)+2/nscale);
18 V3Dnew=V3D;
19 V3Dextra=V3D;
20 for ip1=1:np-1
21     for ip2=ip1+1:np
22         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
```

```
76          % left
77          npextra=npextra+1;
78          xyposextra(npextra,1)=xypos(ip,1)-a;
79          xyposextra(npextra,2)=xypos(ip,2);
80          if(xyposextra(npextra,1)^2+xyposextra(npextra,2)^2 >
                radius^2)
81              npextra=npextra-1;
82              else
83              V3Dextra(npextra,:)=V3D(ip,:);
84          end
85      end
86      if(weight(ip)==1)
87          % the point is not on an edge or corner, so copy other 4
88          npextra=npextra+1;
89          xyposextra(npextra,1)=xypos(ip,1)+a/2;
90          xyposextra(npextra,2)=xypos(ip,2)+a*sqrt(3)/2;
91                  if(xyposextra(npextra,1)^2+xyposextra(npextra,2)
                        ^2 >radius^2)
92              npextra=npextra-1;
93              else
94              V3Dextra(npextra,:)=V3D(ip,:);
95                  end
96
97          npextra=npextra+1;
98          xyposextra(npextra,1)=xypos(ip,1)+a/2;
99          xyposextra(npextra,2)=xypos(ip,2)-a*sqrt(3)/2;
100                 if(xyposextra(npextra,1)^2+xyposextra(npextra,2)
                        ^2 >radius^2)
101             npextra=npextra-1;
102             else
103             V3Dextra(npextra,:)=V3D(ip,:);
104                 end
105
106         npextra=npextra+1;
107         xyposextra(npextra,1)=xypos(ip,1)-a/2;
108         xyposextra(npextra,2)=xypos(ip,2)+a*sqrt(3)/2;
109                 if(xyposextra(npextra,1)^2+xyposextra(npextra,2)
                        ^2 >radius^2)
110             npextra=npextra-1;
111             else
112             V3Dextra(npextra,:)=V3D(ip,:);
113                 end
114
115
116         npextra=npextra+1;
117         xyposextra(npextra,1)=xypos(ip,1)-a/2;
118         xyposextra(npextra,2)=xypos(ip,2)-a*sqrt(3)/2;
119                 if(xyposextra(npextra,1)^2+xyposextra(npextra,2)
                        ^2 >radius^2)
120             npextra=npextra-1;
121             else
122             V3Dextra(npextra,:)=V3D(ip,:);
123                 end
124
125     end
126     if((abs(xypos(ip,1)-a/2)<del) && weight(ip)>0.4)
127         % the point is  on the right row of the original cell so
                copy
128         % and is not in a corner - so copy up and down
                diagonally
129         npextra=npextra+1;
```

```
130         xyposextra(npextra,1)=xypos(ip,1)-a/2;
131         xyposextra(npextra,2)=xypos(ip,2)+a*sqrt(3)/2;
132                 if(xyposextra(npextra,1)^2+xyposextra(npextra,2)
                        ^2 >radius^2)
133             npextra=npextra-1;
134             else
135             V3Dextra(npextra,:)=V3D(ip,:);
136                 end
137
138                npextra=npextra+1;
139         xyposextra(npextra,1)=xypos(ip,1)-a/2;
140         xyposextra(npextra,2)=xypos(ip,2)-a*sqrt(3)/2;
141                 if(xyposextra(npextra,1)^2+xyposextra(npextra,2)
                        ^2 >radius^2)
142             npextra=npextra-1;
143             else
144             V3Dextra(npextra,:)=V3D(ip,:);
145                 end
146
147     end
148 end
149 xyposextrasize=size(xyposextra);
150 V3Dextrasize=size(V3Dextra);
151 plot(xyposextra(:,1),xyposextra(:,2),'bs')
152 % we now have a candidate neighbours - npextra of them,
        xyposextra in
153 % position with values of potential stored in V3Dextra and we
        are looking
154 % to find potential values at the npnew points listed in
        xyposreq. V3Dnew
155 % already has the existing np points loaded into it.
156 %
157 % now find nearest neighbours of all required points
158 distance2=zeros(npnew,npextra);
159 distance=zeros(npnew,npextra,2);
160 dumvec=linspace(0,0,npextra);
161 ipextraindex=linspace(0,0,npextra);
162 distance0=linspace(0,0,nneb);
163 distancetemp=linspace(0,0,nneb);
164 D=linspace(0,0,nneb);
165 sigma=linspace(1,1,nneb);
166 sigma(5:nneb)=100;
167 tol=a/nscale*0.01;
168 iflag=0;
169 ni=10;
170 f=zeros(ni,nneb);
171 for ipnew=1:npnew
172     for ipextra=1:npextra
173         distance(ipnew,ipextra,:)=xyposextra(ipextra,:)-xyposreq
                (ipnew,:);
174         distance2(ipnew,ipextra)=distance(ipnew,ipextra,1)^2+
                distance(ipnew,ipextra,2)^2;
175     end
176     [dumvec,ipextraindex]=sort(squeeze(distance2(ipnew,:)));
177     distance2(ipnew,:)=dumvec;
178     distance(ipnew,:,1)=squeeze(distance(ipnew,ipextraindex,1));
179     distance(ipnew,:,2)=squeeze(distance(ipnew,ipextraindex,2));
180     % so now have points in order - ipextraindex has indices of
            the
181     % ipextras
182     % now calculate
```

```
183      distancetemp=sqrt(squeeze(distance2(ipnew,1:nneb)));
184       if(iflag==0)
185           iflag=1;
186           distance0=distancetemp;
187       else
188           if (sum(abs(distancetemp-distance0))>tol)
189               xohhelp=1
190           end
191       end
192  % pause on
193  % figure(2000)
194  % clf
195  % hold on
196  % plot(xyposextra(:,1),xyposextra(:,2),'rx')
197  % plot(xyposreq(:,1),xyposreq(:,2),'go')
198  % plot(xyposreq(ipnew,1),xyposreq(ipnew,2),'ro')
199  % for ij=1:nneb
200  %     plot(xyposextra(ipextraindex(ij),1),xyposextra(
         ipextraindex(ij),2),'bs')
201  % end
202  % pause;
203  % pause off
204       for ij=1:nneb
205           dx=distance(ipnew,ij,1);
206           dy=distance(ipnew,ij,2);
207           % index of neighbour
208           f(1,ij)=1;
209           f(2,ij)=dx;
210           f(3,ij)=dy;
211           f(4,ij)=dx*dx;
212           f(5,ij)=dx*dy;
213           f(6,ij)=dy*dy;
214           f(7,ij)=dx*dx*dx;
215           f(8,ij)=dx*dx*dy;
216           f(9,ij)=dx*dy*dy;
217           f(10,ij)=dy*dy*dy;
218       end
219
220       M=zeros(ni,ni);
221       for ik=1:ni
222           for ii=1:ni
223               for ij=1:nneb
224                   M(ik,ii)=M(ik,ii)+f(ik,ij)*f(ii,ij)/sigma(ij)
                          ^2;
225               end
226           end
227       end
228       pause on
229       for iz=1:nz
230  % figure(2001)
231  % clf
232  % hold on
233  % % xlim([xyposreq(ipnew,1)-2*a/nscale,xyposreq(ipnew,1)+2*a/
         nscale])
234  % % ylim([xyposreq(ipnew,2)-2*a/nscale,xyposreq(ipnew,2)+2*a/
         nscale])
235  % plot3(xyposextra(:,1),xyposextra(:,2),V3Dextra(:,iz),'rx')
236  %     plot(xyposreq(ipnew,1),xyposreq(ipnew,2),'go')
237  %
238  %     for ij=1:nneb
239  %     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;
```

```
34  nearhcphighmax=100;
35
36  hcpmin=0;
37  hcpmax=60;
38  hcphighmax=80;
39
40  fccmin=75;
41  fccmax=210;
42
43  bridgemin=35;
44  bridgemax=300;
45
46  bridgebarmin=90;
47  bridgebarmax=340;
48
49  nearbridgebarmin=65; % positions near bridge sites
50  nearbridgebarmax=360; % positions near bridge sites
51
52  otherpoints=190;
53
54  %% positions
55
56  % top1pos=[1];
57  % top2pos=[2,6,136,114,91,84,62,32];
58  % top3pos=[3,7,10,13,140,110,118,124,100,95,88,58,66,39,36,33];
59  % 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];

60  %
61  % 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];

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

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

72  % 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;
```

```
130      end
131  %     for j=1:size(topnewpos10,2)
132  %         pos_temp=topnewpos10(j);
133  %         potential_mat(pos_temp,i)=top10new;
134  %     end
135  %     for j=1:size(topnewpos11,2)
136  %         pos_temp=topnewpos11(j);
137  %         potential_mat(pos_temp,i)=top11new;
138  %     end
139  %     for j=1:size(topnewpos12,2)
140  %         pos_temp=topnewpos12(j);
141  %         potential_mat(pos_temp,i)=top12new;
142  %     end
143  end
144
145  % 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=[];
146  % for i=1:157
147  %     if sum(ismember([i],topnewpos))==1;
148  %         continue
149  %     else
150  %         sizeotherpoints=size(otherpoints,2);
151  %         otherpoints(sizeotherpoints+1)=i;
152  %     end
153  % end
154
155  %% angles
156
157  % all angles - top sites same for all angles
158
159  % for i=1:num_ang/2
160  %     for j=1:size(top1pos,2)
161  %         pos_temp=top1pos(j);
162  %         potential_mat(pos_temp,i)=top1;
163  %     end
164  %     for j=1:size(top2pos,2)
165  %         pos_temp=top2pos(j);
166  %         potential_mat(pos_temp,i)=top2;
167  %     end
168  %     for j=1:size(top3pos,2)
169  %         pos_temp=top3pos(j);
170  %         potential_mat(pos_temp,i)=top3;
171  %     end
172  %     for j=1:size(top4pos,2)
173  %         pos_temp=top4pos(j);
174  %         potential_mat(pos_temp,i)=top4;
175  %     end
176  % end
177
178  %%%%% 0, 60, 120 degrees (first inputting 0 degrees values)
179
```

```
%%% 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)
```

```
241     potential_mat(nearbridge3pos(i),1)=nearbridgebarmax;
242 end
243 
244 %%% Same values for 60, 120 degrees
245 
246 potential_mat(:,5)=potential_mat(:,1);
247 potential_mat(:,9)=potential_mat(:,1);
248 
249 %%%%% 30, 90, 150 degrees (first inputting 30 degrees values)
250 
251 %%% hcp, fcc
252 
253 for i=1:size(hcppos,2)
254     potential_mat(hcppos(i),3)=hcphighmax;
255 end
256 
257 for i=1:size(fccpos,2)
258     potential_mat(fccpos(i),3)=fccmax;
259 end
260 
261 %%% near hcp, fcc
262 
263 for i=1:size(nearhcppos,2)
264     potential_mat(nearhcppos(i),3)=nearhcphighmax;
265 end
266 
267 for i=1:size(nearfccpos,2)
268     potential_mat(nearfccpos(i),3)=nearfccmax;
269 end
270 
271 %%% bridge sites, barriers, near bridge sites
272 
273 % bridge 1
274 
275 for i=1:size(bridge1pos,2)
276     potential_mat(bridge1pos(i),3)=bridgemax;
277 end
278 
279 for i=1:size(bridge1barpos,2)
280     potential_mat(bridge1barpos(i),3)=bridgebarmax;
281 end
282 
283 for i=1:size(nearbridge1pos,2)
284     potential_mat(nearbridge1pos(i),3)=nearbridgebarmax;
285 end
286 
287 % bridge 2
288 
289 for i=1:size(bridge2pos,2)
290     potential_mat(bridge2pos(i),3)=bridgemax;
291 end
292 
293 for i=1:size(bridge2barpos,2)
294     potential_mat(bridge2barpos(i),3)=bridgebarmax;
295 end
296 
297 for i=1:size(nearbridge2pos,2)
298     potential_mat(nearbridge2pos(i),3)=nearbridgebarmax;
299 end
300 
301 % bridge 3
```

```
302
303 for i=1:size(bridge3pos,2)
304     potential_mat(bridge3pos(i),3)=bridgemax;
305 end
306
307 for i=1:size(bridge3barpos,2)
308     potential_mat(bridge3barpos(i),3)=bridgebarmax;
309 end
310
311 for i=1:size(nearbridge3pos,2)
312     potential_mat(nearbridge3pos(i),3)=nearbridgebarmax;
313 end
314
315 %%% Same values for 90, 150 degrees
316
317 potential_mat(:,7)=potential_mat(:,3);
318 potential_mat(:,11)=potential_mat(:,3);
319
320 %%%%% 15, 45 degrees (first inputting 15 degrees values)
321
322 %%% hcp, fcc
323
324 for i=1:size(hcppos,2)
325     potential_mat(hcppos(i),2)=hcpmax;
326 end
327
328 for i=1:size(fccpos,2)
329     potential_mat(fccpos(i),2)=fccmax;
330 end
331
332 %%% near hcp, fcc
333
334 for i=1:size(nearhcppos,2)
335     potential_mat(nearhcppos(i),2)=nearhcpmax;
336 end
337
338 for i=1:size(nearfccpos,2)
339     potential_mat(nearfccpos(i),2)=nearfccmax;
340 end
341
342 %%% bridge sites, barriers, near bridge sites
343
344 % bridge 1
345
346 for i=1:size(bridge1pos,2)
347     potential_mat(bridge1pos(i),2)=bridgemin;
348 end
349
350 for i=1:size(bridge1barpos,2)
351     potential_mat(bridge1barpos(i),2)=bridgebarmin;
352 end
353
354 for i=1:size(nearbridge1pos,2)
355     potential_mat(nearbridge1pos(i),2)=nearbridgebarmin;
356 end
357
358 % bridge 2
359
360 for i=1:size(bridge2pos,2)
361     potential_mat(bridge2pos(i),2)=bridgemax;
362 end
```

```
363
364 for i=1:size(bridge2barpos,2)
365     potential_mat(bridge2barpos(i),2)=bridgebarmax;
366 end
367
368 for i=1:size(nearbridge2pos,2)
369     potential_mat(nearbridge2pos(i),2)=nearbridgebarmax;
370 end
371
372 % bridge 3
373
374 for i=1:size(bridge3pos,2)
375     potential_mat(bridge3pos(i),2)=bridgemax;
376 end
377
378 for i=1:size(bridge3barpos,2)
379     potential_mat(bridge3barpos(i),2)=bridgebarmax;
380 end
381
382 for i=1:size(nearbridge3pos,2)
383     potential_mat(nearbridge3pos(i),2)=nearbridgebarmax;
384 end
385
386 %%% Same values for 45 degrees
387
388 potential_mat(:,4)=potential_mat(:,2);
389
390 %%%%% 135, 165 degrees (first inputting 135 degrees values)
391
392 %%% hcp, fcc
393
394 for i=1:size(hcppos,2)
395     potential_mat(hcppos(i),10)=hcpmax;
396 end
397
398 for i=1:size(fccpos,2)
399     potential_mat(fccpos(i),10)=fccmax;
400 end
401
402 %%% near hcp, fcc
403
404 for i=1:size(nearhcppos,2)
405     potential_mat(nearhcppos(i),10)=nearhcpmax;
406 end
407
408 for i=1:size(nearfccpos,2)
409     potential_mat(nearfccpos(i),10)=nearfccmax;
410 end
411
412 %%% bridge sites, barriers, near bridge sites
413
414 % bridge 1
415
416 for i=1:size(bridge1pos,2)
417     potential_mat(bridge1pos(i),10)=bridgemax;
418 end
419
420 for i=1:size(bridge1barpos,2)
421     potential_mat(bridge1barpos(i),10)=bridgebarmax;
422 end
423
```

```
424 for i=1:size(nearbridge1pos,2)
425     potential_mat(nearbridge1pos(i),10)=nearbridgebarmax;
426 end
427
428 % bridge 2
429
430 for i=1:size(bridge2pos,2)
431     potential_mat(bridge2pos(i),10)=bridgemin;
432 end
433
434 for i=1:size(bridge2barpos,2)
435     potential_mat(bridge2barpos(i),10)=bridgebarmin;
436 end
437
438 for i=1:size(nearbridge2pos,2)
439     potential_mat(nearbridge2pos(i),10)=nearbridgebarmin;
440 end
441
442 % bridge 3
443
444 for i=1:size(bridge3pos,2)
445     potential_mat(bridge3pos(i),10)=bridgemax;
446 end
447
448 for i=1:size(bridge3barpos,2)
449     potential_mat(bridge3barpos(i),10)=bridgebarmax;
450 end
451
452 for i=1:size(nearbridge3pos,2)
453     potential_mat(nearbridge3pos(i),10)=nearbridgebarmax;
454 end
455
456 %%% Same values for 165 degrees
457
458 potential_mat(:,12)=potential_mat(:,10);
459
460 %%%%% 75, 105 degrees (first inputting 75 degrees values)
461
462 %%% hcp, fcc
463
464 for i=1:size(hcppos,2)
465     potential_mat(hcppos(i),6)=hcpmax;
466 end
467
468 for i=1:size(fccpos,2)
469     potential_mat(fccpos(i),6)=fccmax;
470 end
471
472 %%% near hcp, fcc
473
474 for i=1:size(nearhcppos,2)
475     potential_mat(nearhcppos(i),6)=nearhcpmax;
476 end
477
478 for i=1:size(nearfccpos,2)
479     potential_mat(nearfccpos(i),6)=nearfccmax;
480 end
481
482 %%% bridge sites, barriers, near bridge sites
483
484 % bridge 1
```

```
485
486 for i=1:size(bridge1pos,2)
487     potential_mat(bridge1pos(i),6)=bridgemax;
488 end
489
490 for i=1:size(bridge1barpos,2)
491     potential_mat(bridge1barpos(i),6)=bridgebarmax;
492 end
493
494 for i=1:size(nearbridge1pos,2)
495     potential_mat(nearbridge1pos(i),6)=nearbridgebarmax;
496 end
497
498 % bridge 2
499
500 for i=1:size(bridge2pos,2)
501     potential_mat(bridge2pos(i),6)=bridgemax;
502 end
503
504 for i=1:size(bridge2barpos,2)
505     potential_mat(bridge2barpos(i),6)=bridgebarmax;
506 end
507
508 for i=1:size(nearbridge2pos,2)
509     potential_mat(nearbridge2pos(i),6)=nearbridgebarmax;
510 end
511
512 % bridge 3
513
514 for i=1:size(bridge3pos,2)
515     potential_mat(bridge3pos(i),6)=bridgemin;
516 end
517
518 for i=1:size(bridge3barpos,2)
519     potential_mat(bridge3barpos(i),6)=bridgebarmin;
520 end
521
522 for i=1:size(nearbridge3pos,2)
523     potential_mat(nearbridge3pos(i),6)=nearbridgebarmin;
524 end
525
526 %%% Same values for 105 degrees
527
528 potential_mat(:,8)=potential_mat(:,6);
529
530
531 %%%%% All other points on matrix
532
533 for i=1:num_pos
534     for j=1:num_ang
535         if potential_mat(i,j)==0
536             potential_mat(i,j)=otherpoints;
537         end
538     end
539 end
540
541 potential_mat(:,((num_ang/2)+1):num_ang)=potential_mat(:,1:(
        num_ang/2));
542
543 %%%%% Create new file
544
```

```
545 V3Dinterp_AQ=potential_mat;
546 general_file='Potential_AQ_base_file.mat';
547 copyfile(general_file, new_file);
548 save(new_file, 'V3Dinterp_AQ', '-append');
549 clear;
```

FILE2.M

```
 1 clear;
 2 %
       ------------------------------------------------------------------------

 3 % load in Potential.mat
 4 %
       ------------------------------------------------------------------------

 5
 6 load('Potential.mat');
 7 % contains: - matrix of grid point (x,y)-positions
 8 %           - vector of AQ angles
 9 %           - vector of weights for each point
10 %           - matrix of sampled potentials (i.e. Fourier
       components at each point and each angle)
11
12 %
       ------------------------------------------------------------------------

13 % plot the locations of the 157 grid points
14 %
       ------------------------------------------------------------------------

15
16 % Plotting of 157 pts.
17 figure; hold on; axis equal;
18 x_157=xypos(:,1); y_157=xypos(:,2);
19 for i=1:size(xypos,1)
20     plot (x_157(i),y_157(i),'.')
21     text(x_157(i),y_157(i),num2str(i));
22 end
23
24
25
26 % Number of Interpolation Points for rectangular grid & angle
       interpolation
27 nx=120;
28 ny=200;
29 ntheta=25; % to request 30 theta vals., starting at 0, e.g., set
        ntheta=31
30
31 %% define range over which potential will be calculated
32 % potential is calculated across region (x,y) \in ([0,ax],[0,ay
       ])
33
34 % the maximum x-displacement of the WS cell is half the basic
       horizontal lattice spacing
35 ax=2*max(xypos(:,1));
36
37 % the maximum y-displacement of the WS cell is half the basic
       vertical lattice spacing
38 ay=3*max(xypos(:,2));
```

```
39
40  % axes are typically chosen such that dir(x) || dir(a1)
41  a=ax;
42
43  % number of points in non-interpolated WS cell
44  np=157;
45
46  V3Dinterp=V3Dinterp_AQ;
47
48  %% ------------------------------------------------------------------------

49  % interpolating in theta
50  % ------------------------------------------------------------------------

51
52  % intitial number of angles
53  ntheta0=length(thetapos);
54
55  % generate new linearly-spaced vector from [0,2*pi) with ntheta
        -1 entries
56  thetavec=linspace(0,2*pi,ntheta); thetavec=thetavec(1:(ntheta-1)
        );
57
58  ntheta=ntheta-1;
59
60  % use cubic interpolation with spline() function to interpolate
        in theta dimension for each position (x,y)
61  for ixy=1:(size(xypos,1))
62      V3Dinterp_temp(ixy,:)=spline(thetapos,V3Dinterp(ixy,:),
            thetavec); % spline() uses cubic interpolation
63  end
64  V3Dinterp=V3Dinterp_temp;
65
66  %% ------------------------------------------------------------------------

67  % interpolating in x and y
68  % ------------------------------------------------------------------------

69
70  % original integer factor N as defined in the theoretical bckgd
71  nscale=12;
72
73  % use halvespacing() method to double the resolution of
        V3Dinterp
74  [V3Dnew, xyposnew, weightnew, npnew, nscalenew] = halvespacing(
        V3Dinterp, xypos, weight, np,nscale,a,14);
75
76  %% ------------------------------------------------------------------------

77  % interpolating in x and y
78  %-----------------------------------------
79
80  % basic vertical reciprocal lattice spacing
81  G1=4*pi/sqrt(3)/a;
82
83  % 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);
183         end
184         VGnew(iGp,itheta)=dum/nscalenew/nscalenew;
185     end
186 end
187
188 %% ------------------------------------------------------------------------

189 % weight the potential
190 % ------------------------------------------------------------------------

191
192 pe3D=zeros(nx,ny,ntheta);
193
194 for itheta=1:ntheta
195
196 clear A
197     A=zeros(nx,ny);
198     for iGp=1:nGp
199         % weight the potential at each point to account for
                double/treble-counting
200         A(ix0Gp(iGp),iy0Gp(iGp))=VGnew(iGp,itheta)*weightp(iGp);
201     end
202     % take the real (cosine) part so that there is a top site at
           the origin
203     B=real(fft2(A));
204 pe3D(:,:,itheta)=B(:,:);
205 end
206
207 %% ------------------------------------------------------------------------

208 % produce grid of positions we wish to sample in our plot, as
      defined by nx and ny
209 % ------------------------------------------------------------------------

210
211 % Note we calculate it across a rectangle cell rather than a WS
      cell for ease of use
212 X=zeros(nx,ny);
213 Y=X;
214 for ix=1:nx
215     for iy=1:ny
216         X(ix,iy)=ax*(ix-1)/nx;
217         Y(ix,iy)=ay*(iy-1)/ny;
218     end
219 end
220
221 %% ------------------------------------------------------------------------

222 % plot potentials for each calculated theta in [0,180] degrees
223 % ------------------------------------------------------------------------

224
225 % define plot parameters
226 color_scale_min=0; color_scale_max=1050;
```

```
227 asp = [1 1 1/color_scale_max];
228 Lx = [0 max(X,[],'all')]; Ly = [0 max(Y,[],'all')];
229
230 % plot surface/contour map for each calculated angle theta in
        [0,180] degrees
231 for i = 1:24
232     figure(100+i);
233     xlim(Lx); ylim(Ly);
234
235     % surf plot
236     % stest = surf(X,Y,pe3D(:,:,i*3-2)); set(stest,'LineStyle','
            none'); daspect(asp); caxis([color_scale_min
            color_scale_max]); view([0 90]);
237
238     % contour plot
239     stest = contourf(X,Y,pe3D(:,:,i*3-2)); axis equal;
240
241     % export graphics to png files
242     exportgraphics(gcf,sprintf('deg%d.png',round(thetavec(i)
            *3*(180/pi))),'Resolution','300');
243 end
244
245 %% ------------------------------------------------------------------------

246 % prepare potential object for implementation in MD simulation
247 % ------------------------------------------------------------------------

248
249 % Including 3rd z dimension BUT only with length 1 because we
        defined a single corrugation surface z(x,y)
250 PES4D_AQ(:,:,1,:)=pe3D;
251
252 %permute axes t: y,x,z,theta
253 PES4D_AQ=permute(PES4D_AQ,[2 1 3 4]);
254
255 % save potential in file 'loadPES.mat'
256 save('loadPES', 'PES4D_AQ');
```

SOLUTIONS

POTENTIAL ENERGY SURFACES: MAIN TASK

Task 1: Questions

(a) To start, run the file called `file1.m`. This file constructs an initial estimate of the potential based on experimentally-determined values. It generates a file named `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).

Given the values in this file are experimental, the file is given in the appendix.

(b) You may have noticed that there is no explicit mention of an irreducible triangle in either `file1.m` or in `file2.m`. The entire WS cell (a rectangle encompassing the cell, for ease of use) is constructed, and then different values set for the Fourier components at each angle. To what extent was the irreducible triangle method used here, and why?

Task 1: Solution

The AQ molecule cannot be modelled as a single particle since it has an extended, rodlike shape. In general this shape breaks the hexagonal symmetry of the interaction between the molecule and the surface;

therefore tiling the irreducible triangle across the WS cell in the manner described in the theoretical background is no longer reasonable for certain angles.

For angles of 0°, 30°, 60°, 90°, *etc.* the rod lies along the close-packed directions; the irreducible triangle method can be used to find the corrugation function. You can see this in the code where Fourier components of the PES are copied onto multiple points at these angles.

Task 2: Questions

(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 irreducible 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, explain the shapes of the potential seen for each orientation of the rod-like molecule. Why is the change in potential periodic, with a period of 180°?

Task 2: Solutions

When plotting the figures for the corrugation, you may find it more instructive to plot the surface using the `surf()` function and choose `view([0 90])` to see more detail in the corrugation (especially around the local minima).

We should first consider the static potential at 0°. There are local minima at both the fcc and bcc sites; however the deeper hcp sites have small local maxima at their centres. This demonstrates the fact that these sites can behave very differently; and as such it is often useful to distinguish between them in models. While the hcp sites are always minima, the fcc sites are only local minima for certain orientations. Therefore adsorption to the surface occurs preferentially at hcp sites.

We expect the change in potential to be periodic with period 180°. This is because anthraquinone is a rod-shaped molecule with 2-fold rotational symmetry about any axis parallel or perpendicular to its surface and through its centre (called a diad), and as so the interaction between the surface and either end of the molecule is equivalent. For a spherically-symmetric adsorbate we expect a period of 360°.

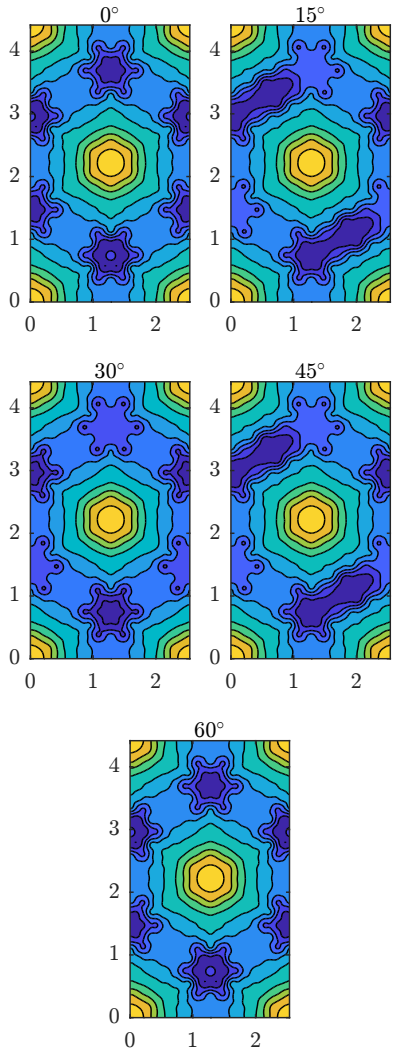

Figure 4 | Contours of the AQ/surface corrugation function

Task 3: Question

With reference to the discussion of AQ motion in the theoretical background, explain the shape of the potential energy surface at each orientation.

Task 3: Solution

The simplest motion to consider is that along the [1 0] azimuth on the surface, for which the crucial angles of interest are $\theta = 0°$ and $\pm 15°$.

Considering the position and orientation of the molecule at each step in the walking process should give a clear picture of how the potential guides the observed motion; each rotation of $\pm 15°$ changes the shape of the potential such that the next site in the walking process becomes the nearest local minimum.

Note that the corrugation only shows the potential of the centre-of-mass at each position - do not be fooled into thinking that the linkers occupy wells on this plot. The potential plotted here is what is 'seen' by the centre of mass due to the positioning of its linkers on the 'real' potential surface.

You may find it instructive to calculate the potential over a higher resolution in theta (by increasing `ntheta` in `file2.m`) and observe the gradual shift in potential that allows each transition to the next step in the walking motion.

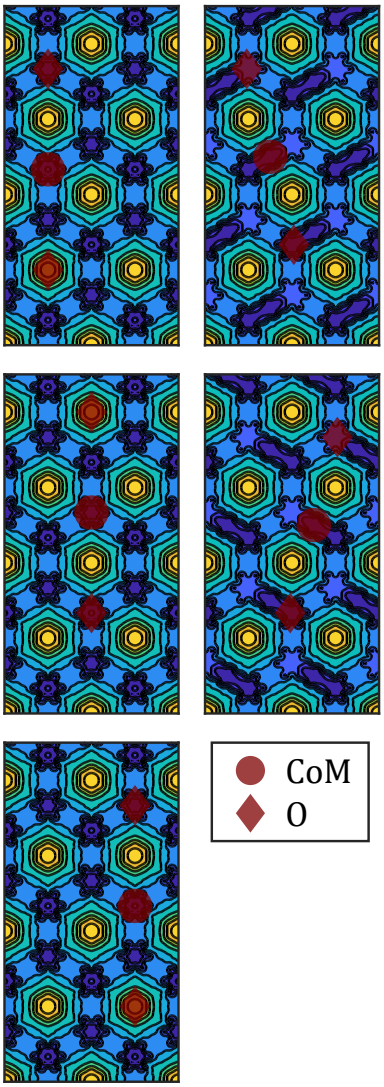

Figure 5 | The AQ/Cu(111) potential energy surface at each stage of the walking motion.

Task 4: Question

(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 in the original question.
With the pigle setup 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:

```
30  % extract positional and orientational data from pigle
31  r = data.prtcl.r;
32
33  % reshape positional data into rows
34  rx = reshape(r(1,:,:), [1 length(r)]);
35  ry = reshape(r(2,:,:), [1 length(r)]);
36  rz = zeros(size(rx));
37
```

```
38  % reshape orientational data into a row
39  rtheta = reshape(r(3,:,:), [1 length(r)]) .* 180/pi;
40
41  % plot figure
42  figure();
43
44  % label axes
45  xlabel('$x$ /\AA'); ylabel('$y$ /\AA');
46
47  % use trick with surface() to plot a continuous gradient on
        the trajectory where the colours correspond to angles
        theta
48  surface([rx;rx], [ry;ry], [rz;rz], [rtheta;rtheta], '
        FaceColor', 'none', 'EdgeColor', 'interp');
49
50  % default 2-D view
51  view(2);
52
53  % add colourbar for theta
54  c = colorbar;
55
56  % format colourbar
57  c.Title.String = '$\theta$ /$^\circ$'; c.Title.Interpreter =
        'latex';
58  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`.

Task 4: Solution

(a) The given ui files along with the existing pigle code of the correct version (v1.1.0-alpha1) should produce a figure similar to the following when the above code is run after pigle:

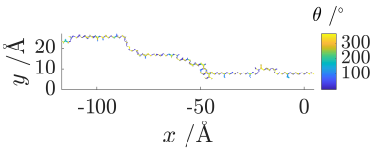

Figure 6 | Trajectory of an AQ molecule on the Cu(111) surface over 50 ps. The colours give the angle θ of the molecule at each position (in degrees).

(b) The trajectory does fit with the predictions of theory since it shows predominantly linear motion along the close-packed directions. In addition, the angle of the molecule at each point further supports theory, as it switches between blue (between 0° and 15°) and yellow (between 0° and −15 degree) with each step. At each hollow site the colour is either blue, green or yellow, corresponding to angles of 0°, 180° or 360° respectively.

(c) This is largely intended to be an exploratory exercise so no com-

plete solution is given, however there are some general trends one should note:

- For smaller masses, the linear inertia is reduced so large changes in motion due to random lattice impulses becomes more likely. This causes the motion to become less linear since the molecule is more free to move to higher-energy sites. In the low-mass limit the motion is Brownian
- For smaller radii, the moment of interia is reduced so large changes in angular motion due to random lattice impulses become more likely. This causes the motion to be less linear since the molecule is more free to change its walking direction. In the low-radius limit the particle can gain very large rotational kinetic energies and the model breaks down. This in turn makes it less likely for the molecule to perform walking motion since the time-scale of rotation is too fast compared to the time-scale for translation.

(d) With rotations disabled the motion is as would generally be expected for a pointlike particle, since at an angle of 0° the potential has local minima at both hollow sites. In fact, since the mass of the molecule is very large, without rotations the average time between jumps is very large, giving the following trajectory: which depicts only a single jump between an hcp (top) and fcc (bottom) site.

With a smaller mass of 75 u we get the more characteristic trajectory for jump diffusion between sites

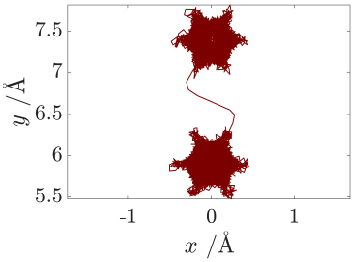

Figure 7 | Trajectory of an AQ molecule (of mass 208 u) on the Cu(111) surface with no rotations permitted. The trajectory is calculated over 50 ps.

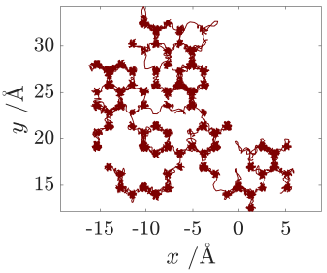

Figure 8 | Trajectory of a molecule of mass 75 u on the Cu(111) surface with no rotations permitted. The trajectory is calculated over 50 ps.