ASSIGNMENT 9: SOLUTION

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

Written in XeLaTeX

ADSORBATES

PREREQUISITES

The chapter *Adsorbates* from the *Theory Handbook*.

INTRODUCTION

The study of particle adsorption to surfaces is of great importance to atom scattering experiments. As discussed in the theory handbook, the presence of defects on a surface, in the form of adsorbates, has the effect of distorting the surface potential; thus the scattering amplitude from the surface is altered by them. Both the scattered intensity distribution and the intermediate scattering factor (for scanning helium microscopy [SHeM] and helium spin echo spectroscopy [HeSE] respectively) are very sensitive to these perturbations in the surface potential. Adsorption of some kind is inevitable in these experiments, even in high vacuum.

As stated in the *Theory Handbook*, and demonstrated in the tutorial on the Eikonal approximation, information can be extracted about a clean suface for complete adlayers of adsorbates. In general, the surface is not clean: thus we must use a more in-depth study. Futhermore, adsorbates cannot be studied in isolation in practice, as modelled in the Monte Carlo and Molecular Dynamics tutorials.

In this tutorial, we employ a Monte Carlo simulation of Ag adsorbates on a Pt(111) surface, using a very simple 'nearest-neighbours' interaction between adsorbates. We calculate the intensity distribution from such a surface and observe how this changes with both momentum transfer and adsorbate coverage.

Many of the simulation techniques used in this tutorial are taken from [1], which also provides some interesting applications of this code if you are interested.

THEORETICAL BACKGROUND

In this tutorial, we will mainly consider two coverage limits. As a prerequisite, we denote the coverage of the adsorbate as Θ. The first of

the coverage limits is the low-coverage limit, where the coverage Θ is much smaller than 1; thus adsorbates are isolated and randomly distributed on the surface. The second limit is the high-coverage limit, where the coverage Θ is much larger than 1. For this regime, successive adlayers are formed on the surface.

Analytic Models for Low Coverage

The theory of adsorbates has been discussed in the *Theory Handbook*. To revise it here, in the low coverage limit, the scattering cross-section of an adsorbate on a flat surface can be expressed as:

$$\Sigma = -\frac{1}{n_s I_0}\frac{\mathrm{d}I}{\mathrm{d}\Theta}\bigg|_{\Theta=0},$$

leading to the scattered intensity being expressed as:

$$\frac{I}{I_0} \approx \exp\left(-n_s \Sigma \Theta\right),$$

where:
1. n_s is the number of substrate atoms/lattice sites per unit area,
2. Θ is the coverage of adsorbates, which is the number of adsorbates per substrate;
3. I is the specular ($\Delta\mathbf{K} = 0$) intensity with coverage Θ,
4. I_0 is the specular intensity with zero coverage.

This equation assumes adsorbates are non-interacting, randomly distributed on the surface, and have small cross-sections that do not overlap.

For small cross-sectional overlap, we may apply the 'overlap approach'; this assumes that the overlap is purely geometical, giving the equation for the intensity as

$$\frac{I}{I_0} = (1-\Theta)^{n_s \Sigma}.$$

At higher coverages of the adsorbate, there is no existing analytic model; at sufficiently high coverages, adsorbates generally from 'adlayers' of atoms: this generates a quasihard-wall potential, similar to the substrates hard-wall potential. Thus, the specular intensity begins to increase again. This leads to a cycling of specular intensity - the adsorbates form successive adlayers.

Monte Carlo Simulation

In this exercise, we wish to model the scattering of helium from silver adatoms on a Pt(111) surface using a Monte Carlo simulation. Why this surface? The choice of the Pt(111) surface stems from the fact that

the He/Pt(111) surface is rather smooth (modelled as flat, to the first order of the potential); thus the majority of scattering on the surface is due to the Ag adsorbates. Additionally, in the literature there is a reliable potential for the He/Pt(111) surface. This will be useful to us.

How do we model the adsorbates? We could distribute them randomly on the surface. While this is trivial, we need to simulate interactions between adsorbates. In our model we assume that only nearest-neighbour interactions are non-negligible, and that the interaction between two adsorbates depends only on the *number* of their nearest neighbours n. Thus, the energy of a particle in a given configuration is:

$$E(n) = E_0 + n\epsilon,$$

where $E_0 = 5.2\,\text{kcal}$ is the activation energy for diffusion at zero coverage, and $\epsilon = 5.0\,\text{kcal}$ is the nearest-neighbour interaction energy. These values are qualitatively valid as determined by experiment.

The transition probability, per unit time, for a particle to move from site i to a site f is written as:

$$\omega_{i\to f} = \nu \exp\left(-E(n_i)/(kT)\right),$$

where ν is a constant. Note we have assumed that the transition rate depends only on the initial state. This is reasonable because the adsorbate must overcome a potential barrier in hopping between different sites.

Model of Surface Potential

In modelling the potential energy of an impinging helium atom onto a surface at position a $\mathbf{r} = (\mathbf{R}, z)$, we assume the following form for the surface potential:

$$V_{\text{He}\to\text{Ag/Pt(111)}}(\mathbf{r}) = V_{\text{He}\to\text{Pt(111)}}(z) + \sum_i V_{\text{He}\to\text{Ag}_i}(\mathbf{r}, \mathbf{r}_i),$$

where the sum is over all adsorbates (at positions $\mathbf{r}_i$). We have assumed pairwise interactions for the He with the Ag atoms. We have assumed these are linear.

For the surface, we use a He-Pt Morse potential:

$$V_{\text{He}\to\text{Pt(111)}}(z) = D_e\left[e^{-2\alpha(z-z_m)} - 2e^{-\alpha(z-z_m)}\right],$$

where $D_e = 2.89\times10^{-4}\,\text{a.u.}$, $\alpha = 0.52\,\text{a.u.}$ and $z_m = 11.46\,\text{a.u.}$.

For the adsorbates we use a He-Ag Lennard-Jones potential:

$$V_{\text{He}\to\text{Ag}}(r) = \frac{C_{12}}{r^{12}} - \frac{C_6}{r^6},$$

where r is the distance between the He and Ag atoms. Additionally we use $C_{12} = 6.33\times10^6\,\text{a.u.}$ and $C_6 = 29.0\,\text{a.u.}$.

All the above parameters are fitted from experimental results in the literature, where a.u. just means 'arbitrary units' in this context. Don't worry too much about the actual values and units of these constants, they have been chosen to provide a good fit with experiment in our simulation. Feel free to tweak these and observe how they affect your results.

Calculation of the Scattering Intensity

In calculating the intensity, we utilise the so-called *sudden approximation*; this requires that the momentum transfer parallel to the surface is small compared to the momentum transfer normal to the surface. Under this assumption, the intensity distribution from a disordered configuration of N adsorbates with positions $\mathbf{r}_i$, denoted by $\beta = \{\mathbf{r}_1, ..., \mathbf{r}_N\}$, can be written analytically. This takes the form:

$$I(\Delta\mathbf{K}) = \frac{1}{S^2}\left\langle \left| \int_S \exp\left(i\Delta\mathbf{K}\cdot\mathbf{R} + 2i\eta_\beta(\mathbf{R})\right) \mathrm{d}^2\mathbf{R} \right|^2 \right\rangle,$$

where:

1. S is is the area of the surface over which integration is performed (ideally all of space),
2. $\langle \dots \rangle$ denotes the average over all possible configurations,
3. $\eta_\beta(\mathbf{R})$ denotes the phase shift for He scattering from a configuration β.

In the so-called *WKB approximation*, a common approximation in quantum physics, the phase shift is given by:

$$\eta_\beta(\mathbf{R}) = \int_{\zeta_\beta(\mathbf{R})}^{\infty} \left(\sqrt{k_{iz}^2 - 2mV_\beta(\mathbf{R}, z)/\hbar^2} - k_{iz} \right) \mathrm{d}z - k_{iz}\zeta_\beta(\mathbf{R}),$$

where $z = \zeta_\beta(\mathbf{R})$ is the *corrugation function* as seen previously. This is the locus of classical turning points for incident He atoms, given by:

$$(\hbar k_{iz})^2 - 2mV_\beta(\mathbf{R}, \zeta_\beta(\mathbf{R})) = 0.$$

Assuming that only the $z = \zeta_\beta(\mathbf{R})$ contribution to the integral is non-negligible, this expression reduces to

$$\eta_\beta(\mathbf{R}) = -2k_{iz}\zeta_\beta(\mathbf{R}).$$

This is equivalent to applying both the Eikonal and Sudden approximations.

TASK

1. Our first task is to implement a class representing the state of adsorbates on a surface. We can represent the surface as a close-packed

plane of $n \times n$ unit cells (with lattice parameter $a = 3.912\,\text{Å}$) and periodic boundary conditions (PBCs).

(a) Create a `Grid` class containing a meshgrid of lattice sites on the surface. It should additionally store adsorbate positions on the lattice, along with a function which converts input coordinates to their PBC equivalents. This class should be able to process adsorbates occupying multiple layers above one another (up to a non-periodic upper limit or otherwise).

(b) Now we have a way to store adsorbate positions, we need methods to add adsorbates to the surface. Write a function `addParticle(x,y)` which places a particle at a horizontal position on the lattice (at the lowest available z-coordinate). Additionally implement a function denoted `addRandParticle()`, which places a particle randomly on the surface. Take care not to forget the PBCs!

(c) We now wish to have some way to allow our adsorbates to settle into an equilibrium arrangement. Implement a `timestep()` function that uses the model discussed in the theoretical background to propagate the system by one timestep. Additionally, make sure that any adsorbates that end up above a vacant site will 'fall' down into it to reduce their potential energy.

(d) Experiment with the model. Draw some figures using the function `addParticle()` and observe how they evolve. You may notice that large islands of adsorbates are more stable than individual adsorbates. Where does this effect come from in our model?

2. Now we have a model for adsorbates on a surface, we wish to model scattering from the surface with adsorbates present. To do so, we must define an equation for the potential energy of an impinging helium atom using the model discussed in the theoretical background.

   To calculate scattering from a surface:

   - Simulate Ag atoms randomly adsorbing to the surface (we suggest starting with the length `len` as 5 and the height `hgt` as 1 for the Grid).
   - Sum the potential functions across the surface and all the adsorbates thereon to obtain the total surface potential.
   - By solving the equation:

     $$V(\mathbf{R}, z) = (\hbar k_{iz})^2/2m,$$

     obtain the classical turning point for each lateral displacement $\mathbf{R}$ on the surface. This will return a set of values $z = \zeta(\mathbf{R})$ - which is the corrugation function.
   - Calculate the intensity for a range of momentum transfers $\Delta\mathbf{K}$

using the Eikonal/Sudden approximations.
Try varying the coverage of the surface and observing how this affects the intensity distribution.

3. (a) The final task in this tutorial is to plot an uptake curve of reflectivity against coverage for the surface. To achieve this, we consider only the specular beam, however we must calculate its intensity at a range of coverages (*i.e.* as we add adsorbates to the surface). Alter your code to allow for such a computation by wrapping your existing code in a loop.
   (b) Considering only small coverages, plot an uptake curve and try fitting the two analytic equations for the reflectivity given in the theoretical background to your data. You will get a higher-resolution curve if the number of unit cells is larger (*i.e.* each adsorbate contributes less to the coverage) and the curve will deviate less if you repeat your calculation over multiple cycles of adsorption and take an average over the cycles (*i.e.* average over more adsorbate configurations as in the theoretical background).
   For what coverage range is your fit reasonable? How might this be influenced by the assumptions in our model?
   (c) Estimate the Ag cross-section from both of these models. Compare this value with the geometric cross-section.
   As a crude approximation, the geometric cross-section can be calculated from the equation for the area of a circle, using the classical turning point in the Lennard-Jones He→Ag potential as the effective radius of the Ag atoms, since it can be assumed that the He is pointlike in comparison the the Ag atoms (due to their large difference in atomic mass).
   (d) Repeat the above but allow for the formation of multiple layers. In particular, try plotting uptake curves for with coverages ranging from 0 to integers larger than 1. Is the resulting curve as you would expect?

Tips

There are two main options here. The first is to represent the positions of each adsorbate in an array of 0s and 1s. Alternatively, one can implement an Adsorbate class containing properties x,y and z for each adsorbate's position. Then the Grid class can simply contain a collection of adsorbates. Whilst the latter of these options will be faster to iterate over, ask yourself which will be simpler to code - don't object-orient your code until it is undecipherable!

You may find it useful to implement a separate `neighbour()` function that returns the state of neighbouring sites. You will also need to

define new properties. Some of these (e.g. the Boltzmann constant k_B) will be constants for all simulations, and should be given in a 'properties(Constant)...end' wrapper. Be careful with units!

You may also find it beneficial to repeat your calculation over multiple cycles of adsorption and take an average over the cycles, since this will cause your data to converge to the analytical solution. The analytical solution in fact requires an average across all possible configurations, however a close approximation is provided for only a few configurations (5 is a good starting point).

You should use similar scattering parameters to those given in previous tutorials, but it is really up to you - what do you wish to model? Many of the parameters in the worked solutions are merely chosen to improve readability of figures more than to model a system of interest (though they are usually of the correct order of magnitude).

EXTENSION

1. Change your code to use the more general form of the WKB approximation, as opposed to the Eikonal/sudden approximation. When does the former approximation give more accurate answers?
2. (a) Modify your simulation such that adsorbates cannot approach within one space of each other. This will require a redefinition of nearest neighbours in your code, as well as when adding particles to the surface. An uptake curve plotted under such a simulation should follow the relation:

$$\frac{I}{I_0} = (1 - m\Theta)^{n_s \Sigma},$$

   as described in the Theory Handbook. Recall that $1/m$ is the maximum allowed coverage of the surface under this model.

   (b) You may wish to make the closest distance a variable of your simulation so that you can experiment with different values m.

REFERENCES

1. J. Chem. Phys. 106, 4228 (1997); https://doi.org/10.1063/1.473513

SOLUTIONS

ADSORBATES: MAIN TASKS

Task 1: Question

Our first task is to implement a class representing the state of adsorbates on a surface. We can represent the surface as a close-packed plane of $n \times n$ unit cells (with lattice parameter $a = 3.912\,\text{Å}$) and periodic boundary conditions (PBCs).

(a) Create a `Grid` class containing a meshgrid of lattice sites on the surface. It should additionally store adsorbate positions on the lattice, along with a function which converts input coordinates to their PBC equivalents. This class should be able to process adsorbates occupying multiple layers above one another (up to a non-periodic upper limit or otherwise).

(b) Now we have a way to store adsorbate positions, we need methods to add adsorbates to the surface. Write a function `addParticle(x,y)` which places a particle at a horizontal position on the lattice (at the lowest available `z`-coordinate). Additionally implement a function denoted `addRandParticle()`, which places a particle randomly on the surface. Take care not to forget the PBCs!

(c) We now wish to have some way to allow our adsorbates to settle into an equilibrium arrangement. Implement a `timestep()` function that uses the model discussed in the theoretical background to propagate the system by one timestep. Additionally, make sure that any adsorbates that end up above a vacant site will 'fall' down into it to reduce their potential energy.

(d) Experiment with the model. Draw some figures using the function `addParticle()` and observe how they evolve. You may notice that large islands of adsorbates are more stable than individual adsorbates. Where does this effect come from in our model?

Task 1: Solution

We begin by defining a handle class `Grid`:

```
1 classdef Grid < handle
```

The key input variables of the class will be the lattice parameter, the simulation temperature and the dimensions of the grid. We choose to store the positions of all possible adsorbate sites (in orthogonal `(x,y,z)`-coordinates) in the 3D meshgrid `Rxy`. The positions of adsorbates are then stored in the matrix `rs` such that if `rs(u,v,w)=1` then there is an adsorbate at position `(x,y,z)` such that `x=Rxyz(u,1), y=Rxyz(v,2)` and `z=Rxyz(w,3)`. The `(u,v,w)`-basis in fact corresponds to the hexagonal basis of the lattice, but with the origin shifted to $(1,1)$ and the dimensions of the unit cells normalised. We refer to this as index space.

```
2     properties
3         % substrate lattice parameter /A
4         a = 3.912;
5
6         % adsorbate adlayer depth /A
7         c = 1;
8
9         % temperature /K
10        T = 300;
11
12        % MC grid horizontal dimensions, same for u and v
13        len = 100;
14
15        % MC grid vertical dimensios, w
16        hgt = 100;
17
18        % x,y,z grid of lattice sites
19        Rxyz = Grid.xyGrid(3.912, 1, 100, 100);
20
21        % INDEX (u,v,w) positions of adsorbates (1 if present, 0
              if not)
22        rs = zeros([100 100 100]);
23    end
```

In the default constructor for `Rxy` we called a (static) method `xyGrid`, so it would be prudent to define this before moving on. `xyGrid` is intended to generate a meshgrid of all the sites which adsorbates could occupy.

```
24    methods(Static)
25        function Rxyz = xyGrid(a, c, len, hgt)
26            % generates a hexagonal grid in orthonormal x,y
                  coords
27
28            % Coordinates in hexagonal basis:
29            [U,V,W] = meshgrid(a.*[0:1:len-1], a.*[0:1:len-1], c
                  .*[0:1:hgt-1]);
30
31            % Coordinates in orthonormal rectangular basis:
32            X = U+V.*1/2; Y = V.*sqrt(3)/2; Z = W;
33            Rxyz = [X(:) Y(:) Z(:)];
34        end
35    end
```

where the coordinate transform from line 31 follows from trigonometrical considerations.

The next step is to write a constructor for the grid, the code for which is given below. Note that ease of use the surface is always initialised as clean (i.e. no adsorbates), so `rs` is defined as an array of zeros.

```
36      methods
37          function this = Grid(a, c, T, len, hgt)
38          %
39
40
41              % if the number of arguments is correct, run the
                    following, else just use the default values
42              if nargin > 0
43
44                  % substrate lattice parameter /A
45                  this.a = a;
46
47                  % adsorbate adlayer depth /A
48                  this.c = c;
49
50                  % temperature /K
51                  this.T = T;
52
53                  % MC grid horizontal dimensions, same for u and
                        v
54                  this.len = len;
55
56                  % MC grid vertical dimensios, w
57                  this.hgt = hgt;
58
59                  % x,y,z grid of lattice sites
60                  this.Rxyz = Grid.xyGrid(a, c, len, hgt);
61
62                  % INDEX (u,v,w) positions of adsorbates (1 if
                        present, 0 if not)
63                  this.rs = zeros([this.len this.len this.hgt]);
64              end
65          end
```

The final step is to implement the periodic boundary conditions. The easiest way to do this is to define a method `pbc()` that applies PBCs to any input. We will only be working in index space in what follows, so we design our method to take any integer `z` to the range [1,`len`], corresponding to a horizontal coordinate in index space (We never apply PBCs along the axis normal to the plane since symmetry is broken in this direction). This can be conveniently implemented using the modulo function, however some additional shifting of indices is required thanks to Matlab's from-one indexing. The function given below function can take as its argument not only vectors but also matrices of any dimension by exploiting so-called *linear indexing*.

```
66          function fz = pbc(this,z)
67              % applies PBCs to any scalar/vector/matrix input of
                    indices
68              fz = zeros(size(z));
69
70              for i = 1:length(z)
```

```
71                  % indices in range [1, len]
72                  fz(i) = mod(z(i) - 1, this.len) + 1;
73              end
74          end
```

The crucial step in `addParticle()` is to enforce the PBCs. In our implementation we expect the input coordinates to be in index space. Then a simple call to `pbc()` on the inputs is all that is required. We then simply locate the lowest space on the vertical `w`-axis that is available (i.e. the particle adsorbs to the existing surface) and insert a particle there.

```
75          function addParticle(this, u, v)
76              % apply PBCs
77              u = this.pbc(u); v = this.pbc(v);
78
79              % find lowest empty site:
80              w=1;
81              while this.rs(u,v,w)>0 && w<this.hgt
82                  w = w+1;
83              end
84
85              % i.e. run out of vertical space
86              if (w==this.hgt && this.hgt~=1) || (this.rs(u,v,this
                    .hgt)==1 && this.hgt==1)
87                  fprintf('Location already fully occupied.');
88
89              % space left -> add it
90              else
91                  this.rs(u,v,w) = 1;
92              end
93          end
```

Adding particles to the surface randomly function is more nuanced since in principle one may randomly attempt to place an adsorbate in a site that is already full. In our implementation we choose to try multiple random locations to find an empty one on each single call to the function. We set an arbitrary limit of 100 attempts since this will cover most cases. The number of empty spaces remaining, `len^2-sum(rs, 'all')`, could also be a reasonable limit, and would likely lead to fewer overall probes at the cost of being less likely to find spaces when the grid is very full.

Note we also use a do-while loop so that an initial attempt is performed *before* the check that the space is empty. Whilst the do-while loop isn't built into Matlab, it is useful in cases such as this for reducing verbosity in one's code.

```
94          function addRandParticle(this)
95              % tries=current xy attempt, w = current z attempt
96              tries=1; w=1;
97
98
99              while true
100                 % randomly pick horizontal position
101                 u = randi([1 this.len]);
102                 v = randi([1 this.len]);
```

```
103
104                 % start at bottom layer
105                 w=1;
106
107                 % Try all vertical sites
108                 while this.rs(u,v,w)>0 && w<this.hgt
109                     w = w+1;
110                 end
111
112                 % found empty slot or run out of attempts
113                 if tries>100 || this.rs(u,v,w)==0, break; end
114                 tries = tries+1;
115             end
116
117
118             % if run out of attempts, print error message,
                    otherwise add particle
119             if tries>100
120                 fprintf('No available site found in 100 random
                        probes.\n');
121             else
122                 this.rs(u,v,w) = 1;
123             end
124         end
```

The first step is to add some constants to our class:

```
1     properties(Constant)
2         % scaled Boltzmann const. /kCal K^{-1} so timestep =
              ~2.5 fs
3         k = 0.5;
4
5         % activation energy for diffusion at 0 coverage /kcal
              mol^{-1}
6         E0 = 5.2;
7
8         % nearest neighbour interaction energy /kcal mol^{-1}
9         e = 5.0;
10
11        % Arrhenius prefactor for hopping frequency
12        nu = 1;
13    end
```

It is important to remember that all these constants are arbitrary, however are of the correct order of magnitude so will produce accurate qualitative results if not quantitative ones.

Next is the `neighbour()` method, which simply explores the neighbouring cells in `rs` and checks which are empty. A 'neighbouring cell' is defined as any cell which can be reached by incrementing or decrementing one of the horizontal coordinates by one space.

```
125         function [nn, fs] = neighbours(this, u, v, w)
126             % nn = count of nearest neighbours of given point
127             % fs = matrix of empty neighbours (possible next
                    positions)
128
129
130             % PBCs
131             us = this.pbc([u-1 u u+1]);
132             vs = this.pbc([v-1 v v+1]);
```

```
133
134             fs = [];
135
136             % left
137             if this.rs(us(1),vs(2),w) == 0
138                 fs = [fs; [us(1) vs(2)]];
139             end
140
141             % right
142             if this.rs(us(3),vs(2),w) == 0
143                 fs = [fs; [us(3) vs(2)]];
144             end
145
146             % down
147             if this.rs(us(2),vs(1),w) == 0
148                 fs = [fs; [us(2) vs(1)]];
149             end
150
151             % up
152             if this.rs(us(2),vs(3),w) == 0
153                 fs = [fs; [us(2) vs(3)]];
154             end
155
156             % max neighbours - no. empties
157             nn = 4 - size(fs, 1);
158         end
```

Having defined the `neighbours()` function, the `timestep()` function is simple to implement. We iterate once over the entire grid, considering each layer before the next. Each time we encounter an adsorbate (`rs(u,v)=1`), we use the probability given in the theoretical background to determine whether it jumps:

```
159         function timestep(this)
160         % allows each particle to advance one timestep. The
                length of each timestep is determined by the
                arbitrary constant nu
161
162             % propagate the state of each layer sequentially,
                    from the bottom up
163             for w = 1:this.hgt
164
165                 % iterate over horizontal positions
166                 for u = 1:this.len
167
168                     % iterate over horizontal positions
169                     for v = 1:this.len
170
171                         % if an adsorbate is present, we must
                                consider whether it jumps
172                         if this.rs(u,v,w) > 0
173
174                             % calculates n_i, the number of
                                    nearest neighbours and fs, the
                                    index coordinates of the
                                    available empty sites
175                             [n_i, fs] = this.neighbours(u,v,w);
176
177                             % calculate total energy of
                                    adsorbate
```

```
178                             En_i = Grid.E0 + n_i*Grid.e;
179
180                             % calculate hopping probability
181                             w_if = Grid.nu * exp(-En_i/(Grid.k*
                                    this.T));
```

If the particle jumps, we then randomly sample the jump direction from the available neighbouring sites with `randperm()`. We then have to allow the particle to 'fall' down to the lowest possible site.

```
183                             if ~isempty(fs) && binornd(1,w_if)
184                                 % no. of empty sites
185                                 frows = size(fs,1);
186
187                                 % sample empty sites
188                                 f = fs(randperm(frows,1),:);
189
190                                 % Delete old position
191                                 this.rs(u,v,w) = 0;
192
193                                 % Try same layer @ new u,v posn
194                                 w_new = w;
195                                 while w_new~=1 && this.rs(f(1),f
                                        (2),w_new-1)==0
196                                     % If there's room, 'fall'
                                            down
197                                     w_new = w_new - 1;
198                                 end
199
200                                 % Concretely move to lower
                                        position @ new u,v
201                                 this.rs(f(1),f(2),w_new) = 1;
```

However, even if the particle doesn't move, on the previous layer the particle below it may have moved from beneath it. Thus we still have to allow the particle to 'fall' down to the lowest possible site.

```
202                             % If the site below is empty
203                             elseif w~=1 && this.rs(u,v,w-1)~=1
204                                 % Delete old position
205                                 this.rs(u,v,w) = 0;
206
207                                 % Start at current layer
208                                 w_new = w;
209                                 while w_new~=1 && this.rs(f(1),f
                                        (2),w_new-1)==0
210                                     % If there's room, 'fall'
                                            down
211                                     w_new = w_new - 1;
212                                 end
213
214                                 % Concretely move to lower
                                        position @ same u,v
215                                 this.rs(u,v,w_new) = 1;
216                             end
217                         end
218                     end
219                 end
220             end
221         end
```

We additionally define a `plot()` function for producing graphics of the surface, which you may find useful.

```
222         function plot(this)
223             % Reshape rs to match the meshgrid index convention
224             rs_flat = reshape(this.rs, [this.len^2*this.hgt 1]);
225
226             % Use find() to return all the indices for which
                    there is an adsorbate
227             ind = find(rs_flat > 0);
228             figure();
229
230             % Plot surface
231             fill(10.*[0 this.a 3*this.a/2 this.a/2]', 10.*[0 0
                    this.a*sqrt(3)/2 this.a*sqrt(3)/2]', [0.9 0.9
                    0.9], 'LineWidth', 0.01); hold on;
232
233             % Plot points at locations of adsorbates
234             scatter3(this.Rxyz(ind,1), this.Rxyz(ind,2), this.
                    Rxyz(ind,3), '.'); axis equal;
235
236
237             % Set limits of plot to encompass the entire surface
                     we're simulating, as well as all possible
                    adlayer heights
238             xlim([0 this.len*this.a*3/2]); ylim([0 this.len*this
                    .a*sqrt(3)/2]);
239             zlim([0 this.len*this.a]);
240         end
241     end
242 end
```

In our model large islands of adsorbates will typically be more stable than isolated adsorbates since we defined the transition probability such that it decays exponentially with the number of nearest neighbours. Thus, when adsorbates form an island they are less likely to leave. So although our model doesn't encode *long*-range interactions between adsorbates, it does encode their *short*-range attraction.

Note that adsorbates which adsorbed at higher levels fell down to lower ones as implemented in our model. The adsorbates also began to form islands as predicted.

The above simulation was carried out at a low temperature to speed up adsorbate motion. This could also be done by varying the Boltzmann constant (and thus the scale of the timestep) in the Grid class.

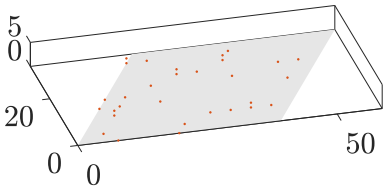

Figure 1 | Result of sudden deposition of 30 adsorbates.

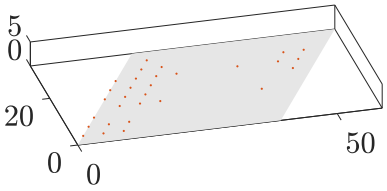

Figure 2 | Positions of adsorbates after 5000 timesteps.

Task 2: Question

Now we have a model for adsorbates on a surface, we wish to model scattering from the surface with adsorbates present. To do so, we must define an equation for the potential energy of an impinging helium atom using the model discussed in the theoretical background.

To calculate scattering from a surface:

- Simulate Ag atoms randomly adsorbing to the surface (we suggest starting with the length `len` as 5 and the height `hgt` as 1 for the

Grid).

- Sum the potential functions across the surface and all the adsorbates thereon to obtain the total surface potential.
- By solving the equation:

$$V(\mathbf{R}, z) = (\hbar k_{iz})^2/2m,$$

obtain the classical turning point for each lateral displacement $\mathbf{R}$ on the surface. This will return a set of values $z = \zeta(\mathbf{R})$ - which is the corrugation function.

- Calculate the intensity for a range of momentum transfers $\Delta\mathbf{K}$ using the Eikonal/Sudden approximations.

Try varying the coverage of the surface and observing how this affects the intensity distribution.

Task 2: Solution

The first step is to define all the scattering parameters as in previous assignments. We define a range of momentum transfers we wish to consider and use the function `beamprops_3()`, to calculate the corresponding normal components of the wavevectors (see appendix):

```
1  % clears workspace variables
2  clear;
3
4  % prevents console output for fsolve()
5  o = optimset('Display','off');
6
7  %------------------------------------------------------------------
8  % initialise scattering parameters
9  %------------------------------------------------------------------
10
11 % reduced Planck's constant units of J, kg and u
12 hbar = 2.05;
13
14 % Incident energy /meV
15 E = 35;
16
17 % Probe particle mass /amu
18 m = 4;
19
20 % Incident angle /rad
21 theta_i = 0;
22
23 %------------------------------------------------------------------
24 %------------------------------------------------------------------
25
26
27
28
29 %------------------------------------------------------------------
30 % define the momentum transfers we're interested in
31 %------------------------------------------------------------------
32
33 % number of moduli of transfers we wish to consider
```

```
nDK = 500;

% set of moduli in given range
mods_DK = linspace(-20,20,nDK);

% set of considered directions
args_DK = 0;

% final set of considered momentum transfers as vectors
DKs = [cos(args_DK); sin(args_DK)] * mods_DK;

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



%-------------------------------------------------------------------
% beam propagation for each considered momentum transfer
%-------------------------------------------------------------------

% z-components of wavevectors /A^{-1}
[k_iz,k_Gz] = beamprops_3(E,m,theta_i,DKs);

%-------------------------------------------------------------------
%-------------------------------------------------------------------
```

The next step is to set up the grid and add some adsorbates. Here we set `len=5` and add 40 random adsorbates, giving adsorbates 5 timesteps to diffuse towards equilibrium between each call to `addRandParticle()`.

```
%-------------------------------------------------------------------
% lattice simulation parameters
%-------------------------------------------------------------------

% number of unit cells per side (x and y) of the considered
    surface
len = 10;

% maximum allowed vertical height of stacked adsorbates
hgt = 1;

% substrate lattice parameter /A
a = 3.912;

% adsorbate adlayer depth /A
c = 1;

% temperature /K
T = 300;

% number of adsorbates
na = 40;

% number of timesteps between each adsorbate
nt = 5;

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


```

```
89
90
91  %-------------------------------------------------------------------
92  % run simulation
93  %-------------------------------------------------------------------
94
95  % intialise grid
96  grid = Grid(a, c, T, len, hgt);
97
98  % iteratively add adsorbates, giving time (~nt ps) for them to
        diffuse between each one
99  for ia =1:na
100
101     % add a particle to the surface @ a random location
102     grid.addRandParticle();
103
104     % allow nt timesteps for the particles to approach
            equilibrium
105     for it = 1:nt
106
107             % perform one timestep
108         grid.timestep();
109     end
110 end
111
112
113
114 % Optionally plot positions of adsorbates
115 % grid.plot();
116
117
118
119 % extract grid of surface
120
121 % grid of the orthogonal (x,y,z) positions of possible adsorbate
         sites
122 R = grid.Rxyz;
123
124 % all x-coords
125 X = R(:, 1);
126
127 % all y-coorda
128 Y = R(:, 2);
129
130 % all z-coords
131 Z = R(:, 3);
132
133 % extract index (u,v,w) positions of adsorbates and reshape (
        reshaping isn't strictly necessary due to something in
        Matlab called 'linear indexing', but is done here for
        clarity)
134 rs = reshape(grid.rs, [len^2*hgt 1]);
135
136 %-------------------------------------------------------------------
137 %-------------------------------------------------------------------
```

We now define the potential energy functions, which we do here using anonymous functions. It would be more efficient to cleverly use separate functions to avoid iteration, however for clarity we choose the former method:

```
138 %-----------------------------------------------------------------
139 % helium/platinum potential
140 %-----------------------------------------------------------------
141
142 % parameters
143
144 % multiplicative factor, parameter /a.u.
145 D_e = 2.89*10^(-4);
146
147 % exponential prefactor, parameter /a.u.
148 alpha = 0.52;
149
150 % point where Vs(z_m)=0, parameter /a.u.
151 z_m = 11.46;
152
153
154 % potential for He -> Pt surface
155 Vs = @(z) D_e*(exp(-2*alpha*(z-z_m)) - exp(-alpha*(z-z_m)));
156
157 %-----------------------------------------------------------------
158 %-----------------------------------------------------------------
159
160
161
162
163 %-----------------------------------------------------------------
164 % helium/single silver adatom potential
165 %-----------------------------------------------------------------
166
167 % parameters
168
169 % multiplicative factor, parameter /a.u.
170 C_12 = 6.33 * 10^6;
171
172 % multiplicative factor, parameter /a.u.
173 C_6 = 29.0;
174
175 % potential function for He -> single Ag adsorbate
176 V_He_Ag = @(x0,y0,z0,x,y,z) C_12./((x-x0).^2 + (y-y0).^2 + (z-z0
        ).^2).^6 + C_6./((x-x0).^2 + (y-y0).^2 + (z-z0).^2).^3;
177
178 %-----------------------------------------------------------------
179 %-----------------------------------------------------------------
```

To sum up the potentials from all adsorbates, we iterate over the grid and add a new contribution to the potential for each position where an adsorbate lies. Note that we have reshaped `rs`, allowing us to iterate over a single index `uv` in a meshgrid fashion rather than over two indices (this could also be achieved using linear indexing, but is done explicitly here for clarity). Note that on line 63 we pass in as arguments the matrices `X` and `Y` so that the potential is only explicitly a function of `z`, but is in fact a matrix equation for all lattice points in the grid.

```
180 %-----------------------------------------------------------------
181 % helium/all silver adatoms potential
182 %-----------------------------------------------------------------
183
184 % Sum up all adsorbate potentials:
```

```
185
186 % initialise Va function
187 Va = @(x,y,z) 0;
188
189 % for each index position in the grid...
190 for uvw = 1:len^2*hgt
191
192     % if there's a particle at (u,v,w)
193     if rs(uvw)==1
194
195             % add a contribution to the adsorbate potential with
                    its origin at this position
196         Va = @(x,y,z) Va(x,y,z) + V_He_Ag(X(uvw),Y(uvw),Z(uvw),x
                ,y,z);
197     end
198 end
199 %-----------------------------------------------------------------
200 %-----------------------------------------------------------------
201
202
203
204
205 %-----------------------------------------------------------------
206 % calculate final potential
207 %-----------------------------------------------------------------
208
209 % Redefine meshgrid of surface to ignore the z-dimension:
210
211 % exclude z-dimension since z is now a variable
212 R = R(1:len^2,1:2);
213
214 % redefine X,Y to ignore permutations over possible z-coords
215 X = R(:,1); Y = R(:,2);
216
217 % total configuration potential for all adsorbates + surface
218 Vf = @(z) Vs(z) + Va(X,Y,z);
219
220 %-----------------------------------------------------------------
221 %-----------------------------------------------------------------
```

We then solve for the classical turning point at each lateral position (x, y) on the grid, which is done numerically using `fsolve()`.

```
222 %-----------------------------------------------------------------
223 % Solve energy-conservation for corrugation
224 %-----------------------------------------------------------------
225
226 % energy conservation equation
227 eqn = @(z) Vf(z) - (hbar * k_iz)^2/(2*m);
228
229 % initial guess (value far from adsorbates on preliminary tests)
230 z0 = 0.25*ones([len^2 1]);
231
232 % solve numerically for corrugation
233 zeta = fsolve(eqn, z0);
234
235 % grid area /A^2
236 S = sqrt(3)/2 * (len*a)^2;
237
238 %-----------------------------------------------------------------
239 %-----------------------------------------------------------------
```

Using the Eikonal approximation, we then calculate the phase factor $e^{i\Delta\mathbf{k}\cdot\mathbf{r}}$ and Fourier transform it as in the Eikonal approximation tutorial:

```
240 %--------------------------------------------------------------------
241 % calculate intensity for each momentum transfer
242 %--------------------------------------------------------------------
243
244 % Eikonal + Sudden approximation formula for integrand
245 phi = reshape(exp(1i*(-zeta*(k_Gz - k_iz) + R*DKs)), [len len
        nDK]);
246
247 % Integrate for intensity and apply kinematic correction
248 I = abs(k_Gz'/k_iz) .* reshape(abs(trapz(Y(1:len), trapz(X(1:len
        ), phi, 1), 2)).^2 ./S^2, [nDK 1]);
249
250 % plot graph
251 figure();
252 plot(mods_DK,I);
253
254 %--------------------------------------------------------------------
255 %--------------------------------------------------------------------
```

This produces varying figures for different numbers of adsorbates:

The intensity distribution for the clean surface is as expected for a flat surface with weak corrugation (see the Eikonal Approximation assignment).

The addition of randomly-distributed adsorbates introduces perturbation in the regular Bragg peaks of the scatter from the surface. Note in particular the sharp drop in the intensity of the specular peak.

Full adlayers, whilst they restore the general shape of the distribution, don't fully restore all its features (as discussed in the Eikonal approximation assignment). Analysis of the entire distribution is impractical, so in practice we only consider the spectral peak, as in the next question.

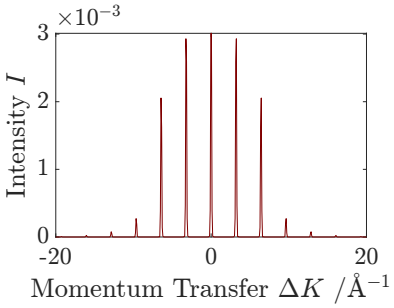

Figure 3 | Intensity of scatter from a surface of hgt=1 and len=10, with zero adsorbates.

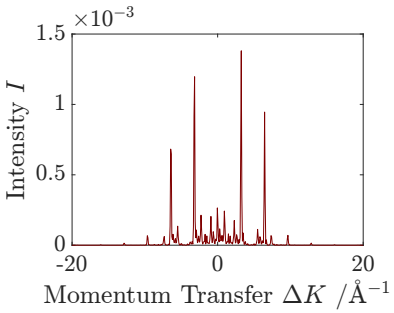

Figure 4 | Intensity of scatter from a surface of hgt=1 and len=10, with 40 adsorbates.

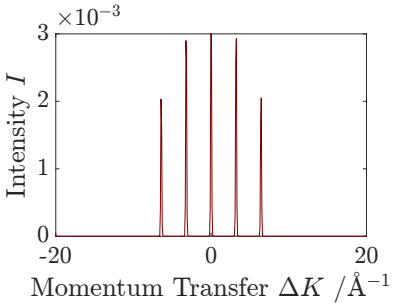

Figure 5 | Intensity of scatter from a surface of hgt=1 and len=10, with 100 adsorbates (i.e. a full adlayer).

Task 3: Question

(a) The final task in this tutorial is to plot an uptake curve of reflectivity against coverage for the surface. To achieve this, we consider only the specular beam, however we must calculate its intensity at a range of coverages (*i.e.* as we add adsorbates to the surface). Alter your code to allow for such a computation by wrapping your existing code in a loop.

(b) Considering only small coverages, plot an uptake curve and try fitting the two analytic equations for the reflectivity given in the theoretical background to your data. You will get a higher-resolution curve if the number of unit cells is larger (*i.e.* each adsorbate contributes less to the coverage) and the curve will deviate less if you repeat your calculation over multiple cycles of adsorption and take an average over the cycles (*i.e.* average over

more adsorbate configurations as in the theoretical background). For what coverage range is your fit reasonable? How might this be influenced by the assumptions in our model?

(c) Estimate the Ag cross-section from both of these models. Compare this value with the geometric cross-section.
As a crude approximation, the geometric cross-section can be calculated from the equation for the area of a circle, using the classical turning point in the Lennard-Jones He→Ag potential as the effective radius of the Ag atoms, since it can be assumed that the He is pointlike in comparison the the Ag atoms (due to their large difference in atomic mass).

(d) Repeat the above but allow for the formation of multiple layers. In particular, try plotting uptake curves for with coverages ranging from 0 to integers larger than 1. Is the resulting curve as you would expect?

Task 3: Solution

In this question, we refer to line numbers in the previous question for brevity.

In the initialisation stage, we only consider the momentum transfer for specular scattering ($\Delta K = 0$). We also initialise empty vectors for the values of intensity and the number of adsorbates at each iteration (note that `nas(i)` will not necessarily equal `ia-1` at every iteration, since at higher coverages adsorbate placement may fail).

```
% clears workspace variables
clear;

% prevents console output for fsolve()
o = optimset('Display','off');

%-----------------------------------------------------------------
% initialise scattering parameters
%-----------------------------------------------------------------

% reduced Planck's constant units of J, kg and u
hbar = 2.05;

% Incident energy /meV
E = 35;

% Probe particle mass /amu
m = 4;

% Incident angle /rad
theta_i = 0;

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


```

```
28
29  %-----------------------------------------------------------------
30  % specular beam => only consider DK=0 (here we allow for study
        of other beams)
31  %-----------------------------------------------------------------
32
33  % number of momentum transfers of interest => only one here b/c
        we only consider specular scatter
34  nDK = 1;
35
36  % specular scatter has |DK|=0
37  mods_DK = 0;
38
39  % argument is undefined for specular scatter
40  args_DK = 0;
41
42  % only one vector => DKs = [0; 0], but useful to keep the option
         to generalise
43  DKs = [cos(args_DK); sin(args_DK)] * mods_DK;
44
45  %-----------------------------------------------------------------
46  %-----------------------------------------------------------------
47
48
49
50
51  %-----------------------------------------------------------------
52  % beam propagation for specular beam only
53  %-----------------------------------------------------------------
54
55  % Incident wavevector, z-component /A^{-1}
56  [k_iz,k_Gz] = beamprops_3(E,m,theta_i,DKs);
57
58  %-----------------------------------------------------------------
59  %-----------------------------------------------------------------
60
61
62
63
64  %-----------------------------------------------------------------
65  % lattice simulation parameters
66  %-----------------------------------------------------------------
67
68  % number of unit cells per side (x and y) of the considered
        surface
69  len = 5;
70
71  % maximum allowed vertical height of stacked adsorbates
72  hgt = 7;
73
74  % lattice parameter /A
75  a = 3.912;
76
77  % adsorbate adlayer depth /A
78  c = 1;
79
80  % temperature /K
81  T = 300;
82
83  % number of adsorbates
84  na = 11;
```

```
85
86  % number of timesteps between each adsorbate
87  nt = 5;
88
89  %-----------------------------------------------------------------
90  %-----------------------------------------------------------------
```

We then wish to iterate over a number of configurations, which can be implemented by looping over the entire execution `nbeta` times to produce a matrix of intensities and adsorbate numbers which can be averaged across:

```
91  %-----------------------------------------------------------------
92  % define parameters for looping over configurations
93  %-----------------------------------------------------------------
94
95  % number of confiugurations
96  nbeta = 10;
97
98  % matrix of 'intensity' columns for each configuration
99  I_beta = zeros([na+1 nbeta]);
100
101 % matrix of 'no. adsorbate' columns for each configuration
102 nas_beta = zeros([na+1 nbeta]);
103
104 %-----------------------------------------------------------------
105 %-----------------------------------------------------------------
106
107
108
109
110 %-----------------------------------------------------------------
111 % loop over nbeta configurations
112 %-----------------------------------------------------------------
113
114 % iterate over nbeta configurations of adding na adparticles
        randomly to the surface
115 for ibeta = 1:nbeta
116
117     % initialise clean grid
118     grid = Grid(a, c, T, len, hgt);
119
120     % specular intensities
121     I = zeros([na+1 1]);
122
123     % corresponding number of adsorbates
124     nas = zeros([na+1 1]);
```

Having initialised our variables, we iteratively add one adsorbate and evaluate the specular intensity for each new adsorbate added from 0 to `na` adsorbates. The loop runs from `1` to `na+1` since we also wish to find the intensity with zero adsorbates, however since `ia` is used as an index in several matrices it cannot be 0.

```
126
127     %-----------------------------------------------------------
128     % slowly add adsorbates to the surface, allow them to
            approach equilibrium with a few timesteps, then
            calculate spectal intensity
129     %-----------------------------------------------------------
```

```
130  % iteratively add adsorbates, giving time (~nt ps) for them to
         diffuse between each one. For each new adsorbate added,
         recalculate the spectral intensity
131      for ia =1:na+1
132
133              % extract grid of surface
134
135          % grid of the orthogonal (x,y,z) positions of possible
                 adsorbate sites
136          R = grid.Rxyz;
137
138          % all x-coords
139          X = R(:, 1);
140
141          % all y-coorda
142          Y = R(:, 2);
143
144          % all z-coords
145          Z = R(:, 3);
146
147              % extract index (u,v,w) positions of adsorbates and
                     reshape (reshaping isn't strictly necessary due
                     to something in Matlab called 'linear indexing',
                      but is done here for clarity)
148          rs = reshape(grid.rs, [len^2*hgt 1]);
149
150              % calculate the number of adsorbates at this
                     iteration (not necessarily equal to ia since
                     adsorbate placement may fail)
151          nas(ia) = sum(rs);
152
153          %-----------------------------------------------------
154          % helium/platinum potential
155          %-----------------------------------------------------
156
157          % multiplicative factor, parameter /a.u.
158          D_e = 2.89*10^(-4);
159
160          % exponential prefactor, parameter /a.u.
161          alpha = 0.52;
162
163          % point where Vs(z_m)=0, parameter /a.u.
164          z_m = 11.46;
165
166
167          % potential for He -> Pt surface
168          Vs = @(z) D_e*(exp(-2*alpha*(z-z_m)) - exp(-alpha*(z-z_m
                 )));
169
170          %-----------------------------------------------------
171          % helium/single silver adatom potential
172          %-----------------------------------------------------
173
174          % multiplicative factor, parameter /a.u.
175          C_12 = 6.33 * 10^6;
176
177              % multiplicative factor, parameter /a.u.
178          C_6 = 29.0;
179
180
181          % potential function for He -> single Ag adsorbate
```

```
182         V_He_Ag = @(x0,y0,z0,x,y,z) C_12./((x-x0).^2 + (y-y0).^2
                + (z-z0).^2).^6 + C_6./((x-x0).^2 + (y-y0).^2 + (z-
                z0).^2).^3;
183
184         %-------------------------------------------------------
185         % helium/all silver adatoms potential
186         %-------------------------------------------------------
187
188         % Sum up all adsorbate potentials:
189
190         % initialise Va function
191         Va = @(x,y,z) 0;
192
193         % for each index position in the grid...
194         for uvw = 1:len^2*hgt
195
196                     % if there's a particle at (u,v,w)
197             if rs(uvw)>0
198
199                         % add a contribution to the adsorbate
                                potential with its origin at this
                                position
200                 Va = @(x,y,z) Va(x,y,z) + V_He_Ag(X(uvw),Y(uvw),
                        Z(uvw),x,y,z);
201             end
202         end
203
204         %-------------------------------------------------------
205         % calculate final potential
206         %-------------------------------------------------------
207
208             % Redefine meshgrid of surface to ignore the z-
                    dimension:
209
210         % exclude z-dimension since z is now a variable
211         R = R(1:len^2,1:2);
212
213             % redefine X,Y to ignore permutations over possible
                    z-coords
214         X = R(:,1); Y = R(:,2);
215
216         % total configuration potential for all adsorbates +
                surface
217         Vf = @(z) Vs(z) + Va(X,Y,z);
218
219         %-------------------------------------------------------
220         % Solve energy-conservation for corrugation
221         %-------------------------------------------------------
222
223         % energy conservation equation
224         eqn = @(z) Vf(z) - (hbar * k_iz)^2/(2*m);
225
226         % initial estimate of turning point (value with no
                adsorbates in preliminary tests)
227         z0 = 0.25*ones([len^2 1]);
228
229         % solve numerically for corrugation
230         zeta = fsolve(eqn, z0);
231
232         % grid area /A^2
233         S = sqrt(3)/2 * (len*a)^2;
```

```
234
235         %-----------------------------------------------------
236         % calculate intensity using Eikonal+Sudden approximation
237         %-----------------------------------------------------
238
239         % eikonal approximation for integrand
240         phi = reshape(exp(1i*(-zeta*(k_Gz - k_iz) + R*DKs)), [
                len len nDK]);
241
242         % integrate to get intensity (kinematic factor = 1 for
                specular beam)
243         I(ia) = reshape(abs(trapz(Y(1:len), trapz(X(1:len), phi,
                 1), 2)).^2 ./S^2, [nDK 1]);
244
245             % add a particle to the surface @ a random location
246         grid.addRandParticle();
247
248             % allow nt timesteps for the particles to approach
                    equilibrium
249         for it = 1:nt
250
251                 % perform one timestep
252             grid.timestep();
253
254         end
255     end
256
257     % append list of intensities at each iteration from this
            configuration to a matrix (I_beta)
258     I_beta(:,ibeta) = I;
259
260     % append list of adsorbate numbers from this configuration
            to a matrix (nas_beta)
261     nas_beta(:,ibeta) = nas;
262 end
263
264 %-------------------------------------------------------------
265 %-------------------------------------------------------------
```

We then convert `nas` to coverage and calculate the specular reflectivity. We choose to perform the average across the rows of the matrices. Technically this isn't correct since there is a possibility that a given call to `addRandParticle()` may fail and result in a duplicate element, thereby causing misalignment in the rows of the matrices. However, since our implementation minimises the amount of failed insertions by performing multiple attempts, a simple average over rows is reasonable as each row will still correspond to a set of *similar* coverages if not a single *same* coverage.

```
266 %-------------------------------------------------------------
267 % normalisation and plotting
268 %-------------------------------------------------------------
269
270 % average nas_beta and I_beta over all configurations <what do
        these represent?>
271 nas_avg = mean(nas_beta,2);
272 I_avg = mean(I_beta,2);
273
274 % convert nas to theta by dividing by the number of unit cells
```

```
    considered
theta = nas_avg./len^2;

% convert intensity to reflectivity by dividing by intensity on
    clean surface
Ref = I_avg/I_avg(1);

% plot graph
figure();
plot(theta, Ref);

%--------------------------------------------------------------------
%--------------------------------------------------------------------
```

We attempt to fit the curve of reflectivity against coverage to both of the analytical models defined in the theoretical background as follows:

```
%--------------------------------------------------------------------
% fitting analytical models
%--------------------------------------------------------------------

hold on;

% low-coverage limit

% fitting length
fitlen = floor(length(theta)/2);

% define equation of fit
f = fittype('exp(-a*x)');

% approximate parameter a in fit
fit_f = fit(theta, Ref, f);

% plot fitted equation for low-coverage
plot(fit_f);



% lattice gas model

% define equation of fit
g = fittype('(1-x)^a');

% approximate parameter a in fit
fit_g = fit(theta, Ref, g);

% plot fitted equation for lattice gas model
plot(fit_g);

% label axes
xlabel('Coverage $\Theta$');
ylabel('Specular Reflectivity $I/I_0$');


hold off;

%--------------------------------------------------------------------
%--------------------------------------------------------------------
```

This yields the following curves:

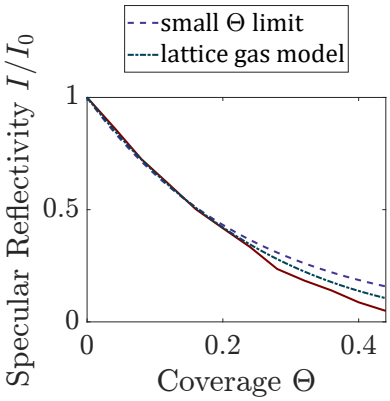

Figure 6 | Uptake curve for a simulated $5 \times 5 \times 7$ grid, averaged over 10 configurations.

There is a lot to discuss in this simple plot. As a rough approximation, the models are valid up to $\Theta_{\text{crit}} \approx 0.25$. This is a reasonable limit since in the literature significant deviations can occur well below this coverage.

Additionally, for sub-critical coverages the lattice gas and small Θ models coincide and are both effective. At super-critical coverages, the lattice gas model provides a better fit. Note that the reflectivity shows a significant negative deviation from both models at higher coverages. This implies that the overlapping regions of adsorbate cross-sections are larger than expected, which is indicative of attractive the adsorbate-adsorbate interactions. This result is reasonable since the model given in the theoretical background was designed to simulate attractive nearest-neighbour interactions.

Using the same fit as above, the small Θ fit gives an exponential prefactor of:

$$n_s \Sigma \approx 4.195 \pm 0.223,$$

whereas the lattice gas fit gives a power of:

$$n_s \Sigma \approx 3.877 \pm 0.103,$$

both with 95% accuracy bounds. Since there is one lattice point per hexagonal surface unit cell for Pt, we have that:

$$n_s = \frac{1}{a^2\sqrt{3}/2} \text{Å}^{-2}.$$

Dividing through by n_s gives us estimates for the adsorbate cross-sections as follows:

$$\Sigma_{\text{i}} = (55.60 \pm 2.96)\,\text{Å}^2,$$
$$\Sigma_{\text{ii}} = (51.38 \pm 1.37)\,\text{Å}^2.$$

Thus there is reasonable agreement between the two fits, with an average value of $\Sigma \approx 53.5\,\text{Å}^2$.

We may also derive the geometric cross-section of Ag atoms from our potential. The classical turning point for the He→Ag potential is approximately at the location where the potential becomes positive (assuming a steep gradient) since the incident energy of helium atoms is much larger than the well-depth. Thus, the classical turning point is given by (see [1]):

$$r_{\text{tp}} \approx (C_{12}/C_6)^{1/6} = 4.1\,\text{Å},$$

giving a value for the *geometric* cross-section of:

$$\Sigma \approx \pi(r_{\text{tp}})^2 = 52.8\,\text{Å}^2.$$

This value is very similar to the value calculated from the uptake curve. This is because the long-range attractive Van der Waals potential provides the dominant contribution to the deflection of incident

He atoms in both cases. See the theory handbook for more discussion of the correspondence between scatter from free and adsorbed atoms.

To consider larger coverages, we run the simulation above with `len=5`, `hgt=7`, `na=100`, `nt=5`, `nbeta=10` (for a shorter runtime we suggest `na=75` and `nbeta=5`). These parameters produce the following figure:

which fits remarkably well with the qualitative relationship described in the theory handbook. However, the reflectivity here varies far more strongly than the uptake curves given in the theory handbook. This is likely due to the fact that our model encourages the formation of perfectly ordered layers with no lattice mismatch with the (flat) surface below (since adsorption only occurs on top-sites).

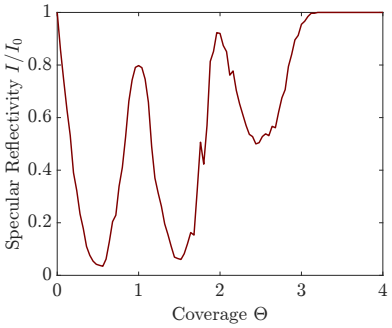

Figure 7 | Uptake curve for a simulated $5 \times 5 \times 7$ grid over a greater range of coverages, averaged over 10 configurations.