5CCP211C — Introduction to Numerical Modelling
Physics Department — King's College London

KING'S College LONDON

Kinetic Monte Carlo

Written by M. Haider and Z.J. Cheung

Group 1 Pair 2

Conducted on November 6, 2018

1 Background and Theory

The Kinetic Monte Carlo method (KMC) is a variant of the Monte Carlo (MC) method of simulation which accounts for the time evolution of processes which are involved in dynamically evolving systems from state to state. Unlike the Metropolis approach in MC, which is employed to investigate systems which have reached equilibrium, KMC can be used to study non-equilibrium dynamical systems such as those involved in chemical reactions and epitaxial growth. Using the MC approach has the familiar advantages of modelling a stochastic process that can achieve importance sampling within a high dimension, many-partical system.

This report will be focusing on the epitaxial crystal growth on a surface. Interactions within such a model are limited to very small time frames and lengths. Simulations using conventional methods such as molecular dynamics work by propagating equations of classical motion forward in time. Integration techniques used in this approach require time steps short enough (10^{-15}s) to resolve atomic vibrations, whereas modelling the evolution of crystal growth poses the challenge of covering huge length and time scales [1]. The KMC attempts to overcome the problem of time scales. This is done using the fact that the dynamics of such a system can be expressed by transitions from state to state. This results in vastly longer time scales which show the full extent of the system's growth.

In an infrequent-event system, each state corresponds to a single energy basin, and for the state to evolve it must overcome an energy barrier to get to another energy basin. For each possible transition to an adjacent basin, there is an associated rate constant k_{ij}, the probability per unit time that it escapes to that state. The transition probabilities for exiting state i have no relation to that prior to entering state i. This type of system dynamic corresponds to a Markov walk [2]. Given that the transition depends only on the transition rate constant, a simple stochastic procedure can be designed to simulate the system from state to state.

The probability that the system remains in the same state is given by,

$$p_{survival}(t) = e^{-k_{tot}t} \tag{1}$$

where k_{tot} is the sum of the rates of transitioning to another state. The probability distribution function for the time of the first transition is given by the negative time derivative of $p_{survival}$,

$$p(t) = k_{tot}e^{-k_{tot}t} \tag{2}$$

The average time τ for transition is the mean of the distribution in Equation 2,

$$\tau = \int_0^\infty tp(t)dt = \frac{1}{k_{tot}} \tag{3}$$

An exponentially distributed random number t_{draw} drawn from the distribution in Equation 2 can be achieved by manipulating a random number r between 0 and 1 ($r \neq 0$),

$$t_{draw} = -\frac{1}{k_{tot}}ln(r) \tag{4}$$

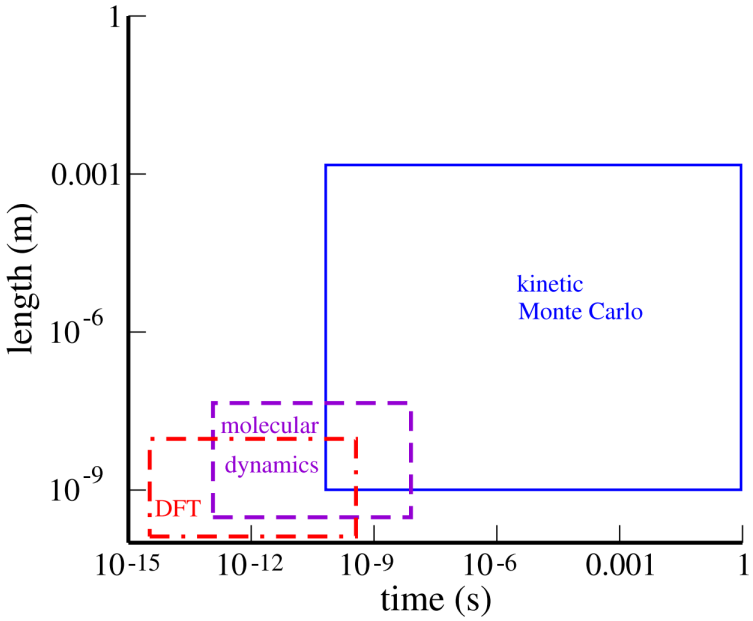


Figure 1: [3]Comparison of length and time scales between density functional theory (DFT), molecular dynamics and kinetic Monte Carlo.

The transition rate constants are found by applying the harmonic approximation to transitioning state theory (TST). Where the ensemble average transition rate is given by the following formula:

$$k^{TST} = n_p \nu e^{-\frac{E_a}{k_B T}} \tag{5}$$

where n_p is the number of possible jump directions, ν is the harmonic frequency, E_a is the energy difference between the basins, k_B is Bolztmann's constant and T is the temperature. The term $n_p\nu$ is called the pre-factor and is dependent on the difference in normal mode frequencies between the minimum and saddle points (stationary point that is not an extermum [2]). The energy difference E_a is also the difference between the saddle point and the minimum. The different moves of deposition, diffusion and assembly are associated with different pre-factors and energy differences. These have been provided for calculation. Once all rates associated with the different moves have been found, the KMC can be initiated.

1.1 KMC Structure

This investigation of epitaxial growth uses a simple square lattice to model an interacting surface where deposition, diffusion and assembly (DDA model) may take place. The type of move that occurs at each step in the simulation is decided by using KMC.

To commence, the input rates and all parameters are initialised by setting them to zero. The rates corresponding to all possible moves in the model are calculated and stored. A random site is chosen to carry out a move. This is done by generating a random set of coordinates within the lattice. A move is chosen based on the analysis of the site and KMC method on the rates for the eligible candidates. The KMC method for choosing which move to perform is found by the following:

The eligible candidates of moves are first found. When initiating the simulation there are no particles within the lattice, so first a particle is spawned (deposition) at a random coordinate. The particle may now diffuse (by one step) in any direction or a new particle may be deposited onto the lattice. When faced with more than one move, the KMC method is utilised. The sum of all eligible

rates are first found (k_{tot}). This corresponds to the addition of diffusion (k_{dif}) and deposition rates (k_{dep}) (at this stage). A uniform random number (r) is generated from $0 \rightarrow 1$ and is multiplied by the total rate. To decide which event is to be carried out the following comparison is made:

$$k_{dep} < rk_{tot} \leq (k_{dep} + k_{dif}) \tag{6}$$

If the quantity rk_{tot} falls within that of k_{dep} than deposition is chosen. If it is above this value and falls within k_{dif} than diffusion is chosen.

At a later simulation time there may also be an option to aggregate with another existing point(s) on the lattice. When given this option, the rate corresponding to aggregation is added to any other rates corresponding to other available moves and the same procedure is carried out. The propagation of time is found by using Equation 4 along each move and is added to the total time.

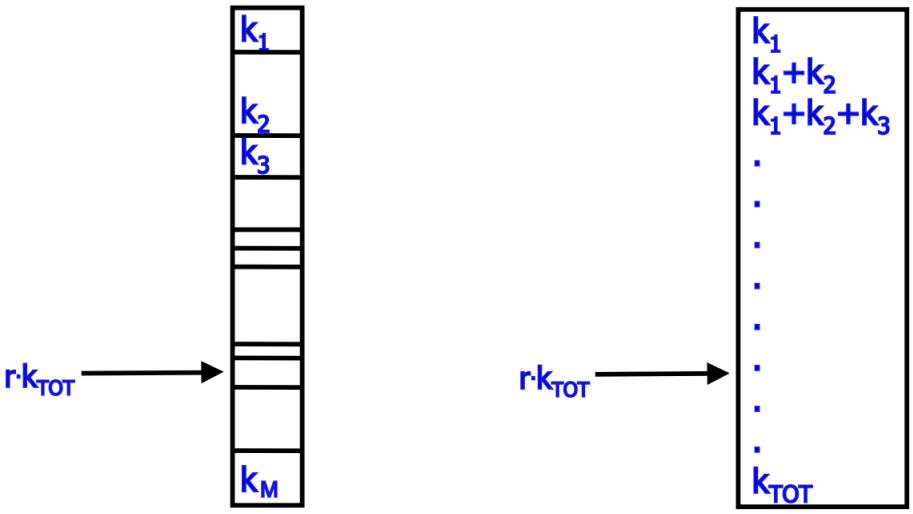


Figure 2: [1] Processes are chosen with a probability proportional to their rates. On the left can be seen rates visualised in term of length - and on the right the approach used to achieve this in the code.

This type of process has no discarded attempts (rejection-free), in contrast to Metropolis Monte Carlo. In general, the KMC should result in an accurate model if the transitions have a Poisson distribution (probability density $\propto Re^{-Rt}$), the processes themselves are independent and the time increments and rates are calculated correctly.

1.2 DDA Model

To find out which moves to conduct after each Monte Carlo step, it is necessary to perform an analysis of the positioning of each point in relation to the others. In total there are 12 positions which need to be identified for any given point in the lattice. There are four positions corresponding to the immediate north, east, south and west neighbours. There are four more for the diagonal neighbours in the four different directions. Lastly there are four more positions that correspond to neighbours whom may be separated by a section - in all four directions.

When a particle performs aggregation with one of its near neighbours, it forms an island and is no longer available to diffuse. Conversely, if a particle is identified to have no near neighbours, it is unable to aggregate and hence has to diffuse. To combat these limiting conditions, it was ensured in

the code that when choosing the random site to preform KMC that these would be limited to free particles - those that had not aggregated.

Periodic boundary conditions (PBC's) are a set of system boundary properties which are used to approximate large systems by modelling a limited section called a unit cell. This model will use a simple 2D PBC to simulate how particles move and interact at the boundaries of the lattice. The geometrical representation of such a model can be compared with the mapping of a 2D surface onto the surface of torus. This means that when the particle moves across a boundary, it reappears on the exact opposite side. This also implies that two or more particles will be able to aggregate across lattice boundaries and form islands.

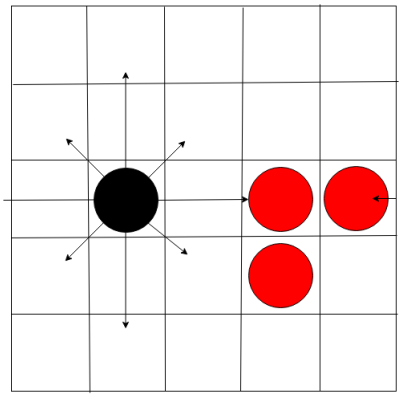


Figure 3: Nearest neighbours of given particle in the lattice. Notice the application of PBC across the boundary.

2 Overview of Algorithms

The basic structure of the code consists of a single Python script. The 2D lattice is generated as a class object, with its state saved in the form of an array consisting of elements of 0 and 1 to depict empty and occupied states, respectively. Each element can be referenced via two values that access a sub-list within a list.

The rates are of a constant value and are calculated by inputting the pre-factor and energy difference for each move into Equation 5. The total rate combinations (when given a choice between moves) are also of a constant value and are found by summing the appropriate rates. The value of the Boltzmann constant is set to 1 since the negative exponential which normally results is so large that the rate is found to be 0. Since this is applied to all rates, a comparison can still be made, albeit the rate values are not accurate.

A random site within the lattice is chosen to conduct the KMC method. This is restricted to particles which are free to diffuse or aggregate. In order to identify these points, a function under the KMC class is called to determine whether all adjacent points are vacant. The function takes in a single argument, a coordinate, and returns a list containing coordinates of the 4 immediate neighbours, i.e. if the length of the list is 0, there are no immediate neighbours and the particle is free to move. Boundary conditions are added by utilising the negative indexing allowed in Python.

Lists of coordinates of empty sites (*returnempty*), occupied sites (*returnoccupied*) and the nearest neighbours regardless of being occupied (*surroundingcoords*) are generated using functions and called when required. A list of all points which can be moved are generated by calling a function *canMove* on the list of all occupied sites. This function calls another function, *checkNeighbours*, which utilizes

len() to check whether the length of the list of coordinates of immediate neighbours is equal to 0. When this condition is satisfied, the point corresponding to this coordinate is appended to a list *canMove*. Next a random coordinate is selected from this list by using the *random.choice* function (within the *choosePoint* function).

The *adsorb* function takes the list of all empty coordinates in the lattice (*returnEmpty*) and randomly selects a coordinate and changing its state to occupied (using *choosePoint* function). The *diffuse* function first provides a list of adjacent coordinates that a point can move to such that it does not aggregate. A random coordinate is chosen from the list by once again recalling the *choosePoint*. The *agglomerate* function works in a similar way. A list of coordinates which includes all coordinates that a point can aggregate to is specified. This is done in a similar fashion to $checkNeighbours$, where the 8 positions (non-immediate neighbours) surrounding the chosen coordinate are investigated. This is done by running a neighbour check function on the point's immediate neighbouring coordinates. These are also applied with boundary conditions, again using the negative indexing available through python. The output results in the generation of a list of coordinates which a point can move to for aggregation.

The next part of the code is the most important. This is where the KMC method is applied and the moves described above are performed. The conditions for satisfying all combinations of moves must be specified. These correspond to; 1) adsorption + aggregation, 2) adsorption + diffusion, 3) adsorption + diffusion + aggregation. This is stored under as a function *actionChoice* under the *kmc* class. Throughout the simulation, this is the main function that is called for each MC step. A random occupied point that can move is selected and checked for whether it can diffuse, aggregate or both. This corresponds to the conditions 1), 2), 3) stated above. The calculation of rates is then carried out and a selection is made, calling the respective function (from the choice of D,D,A). In the case of adsorption, the random occupied point is not needed. While doing so, the time is updated.

In the case where diffusion is no longer possible, the model automatically selects aggregation. When both diffusion and aggregation are not possible, the model will default to adsorption until each site is filled with a point, at which the simulation stops.

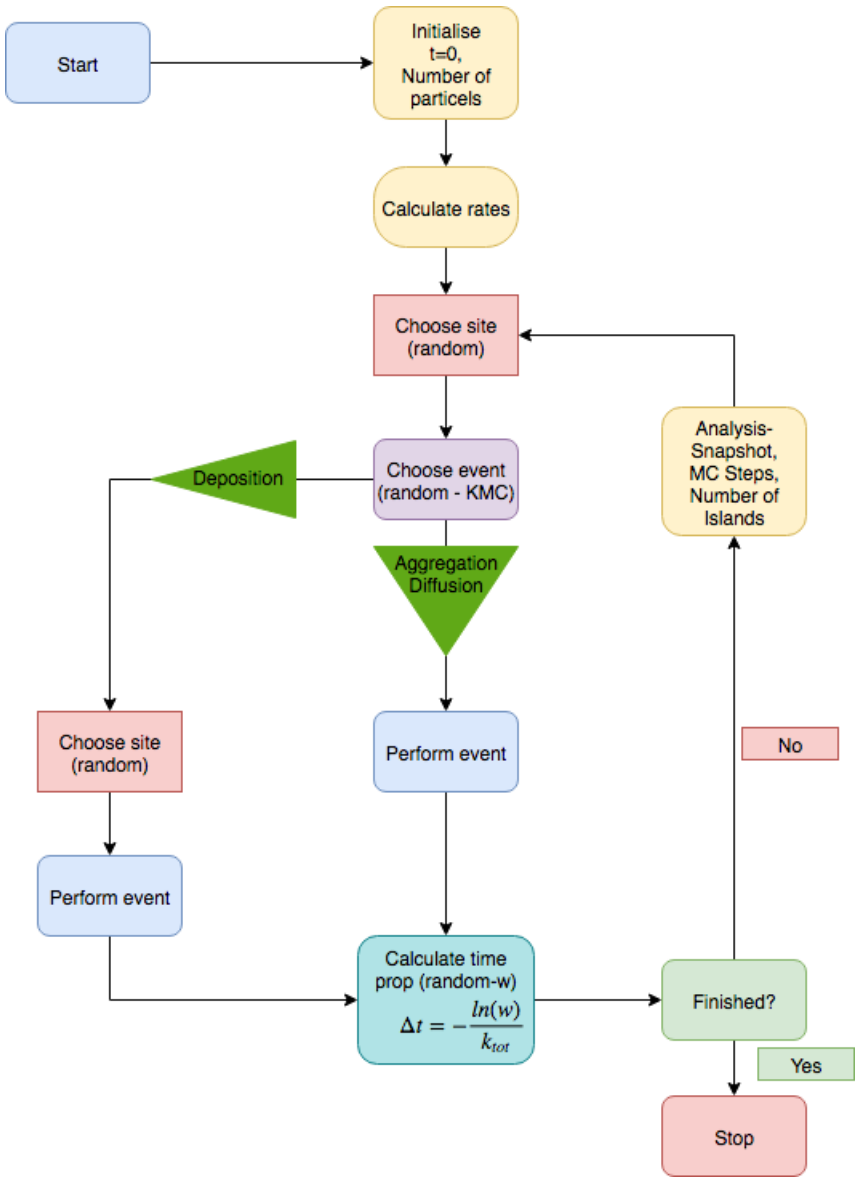


Figure 4: General structure of KMC model used in this investigation.

3 Results

The images generated by the code are given below, with the axes label as indices, useful for checking whether the code is executing properly.

The number of empty coordinates, number of points with no neighbours and number of points with neighbours are plotted against the nth MC step.

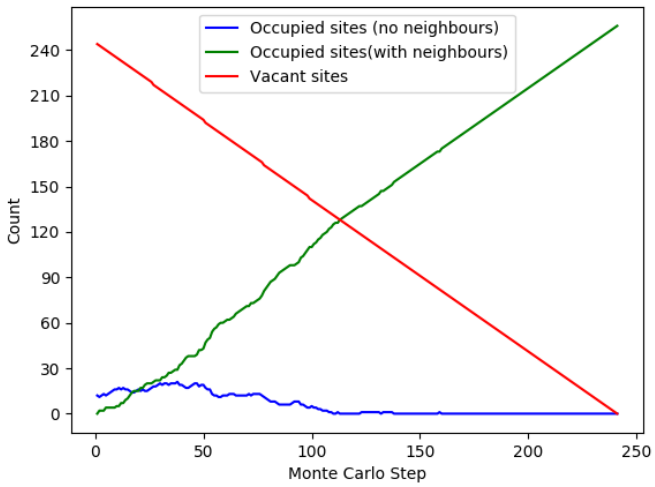
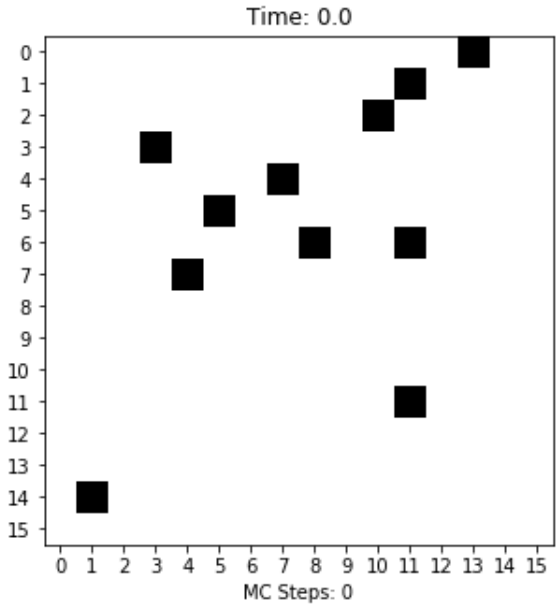
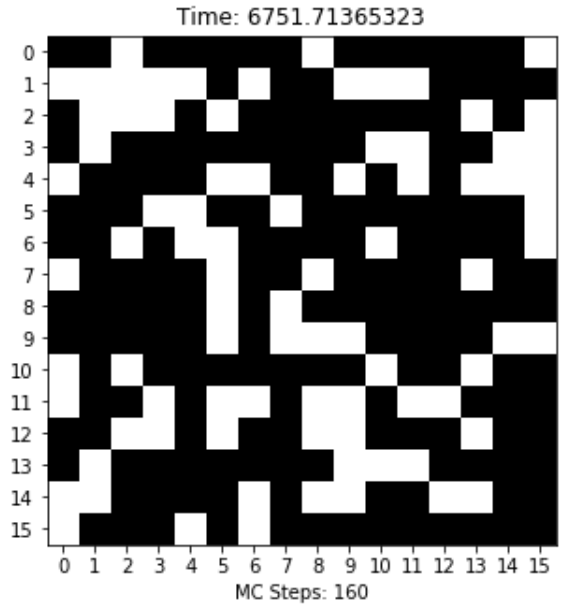


Figure 5: Plot of number of empty coordinates, occupied points with no neighbours and occupied points with neighbours against MC step

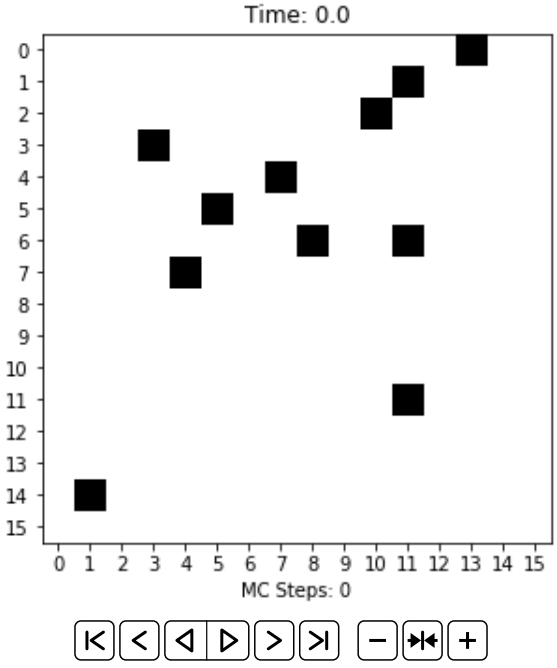
As expected, the number of vacant sites decreases, whereas the number of occupied sites with neighbours increases until all sites are occupied or until the simulation ends. The number of occupied sites with no neighbours is equivalent to the number of points that are free to move. This value oscillates during early MC steps then reduces to 0.



0 KMC steps



160 KMC steps



Evolution from 0 to 244 Monte Carlo steps - open in Adobe reader to view .gif

Figure 6: A

References

[1] *Introduction to the Kinetic Monte Carlo Method*, Aurther F. Voter.

[2] *Mathworld - Wolfram*, available at `http://mathworld.wolfram.com/`.

[3] *Monte Carlo and kinetic Monte Carlo methods a tutorial*, Peter Kratzer.