Foundations of Chemical Kinetics Lecture 16: The kinetic Monte Carlo method

The multi-molecule simulation method

Marc R. Roussel

November 17, 2021

Simulating many molecules

- What we have so far is a method for simulating a single molecule.
  We can repeat single-molecule simulations many times to collect statistics.
- Often, we are less interested in detailed single-molecule trajectories than in the evolution of a probability distribution.
- KMC can handle this case, too. The basic idea is that if $w_{ij}dt$ is the probability that a single molecule executes a jump from state i to state j in time dt, then if there are N_i molecules in state i, the probability that one of them jumps from state i to state j in time dt is $N_i w_{ij} dt$ provided dt is sufficiently small that there is a negligible probability that there is more than one jump in that time.
- Instead of tracking the states of individual molecules, we will keep track of how many molecules are in each state.

System, states and propensities

- The system consists of a number of molecules distributed over the accessible molecular states.
- The state of the system is a vector of populations of molecules in each molecular state: $\mathbf{N} = (N_1, N_2, \ldots, N_s, \ldots)$.
- The propensity of a jump from molecular state i to molecular state j is $a_{ij} = N_i w_{ij}$.
  This is a probability per unit time of one molecule making a jump from molecular state i to state j.
- The KMC algorithm for many molecules is much the same, except that
  - Propensities are used to determine jump probabilities.
  - Instead of updating the state of one molecule, we update the populations of the "from" and "to" states.

The Kinetic Monte Carlo (KMC) Algorithm

Simulating a population of molecules

Initialize:

1. Store the w_{ij} values.
2. Store the initial N_i populations.
3. Set $t = 0$.

The Kinetic Monte Carlo (KMC) Algorithm

Simulating a population of molecules (continued)

At each step:

1. Calculate the propensities $a_{ij} = w_{ij} N_i$.
2. Calculate $a_{\text{tot}} = \sum_{i,j} a_{ij}$.
3. Generate two random numbers $r_{1,2} \in (0, 1)$.
4. Use r_1 to generate the time to next jump:

$$\Delta t = -\ln r_1 / a_{\text{tot}}$$

The Kinetic Monte Carlo (KMC) Algorithm

Simulating a population of molecules (continued)

5. Use r_2 to pick the destination of the jump as follows:
   1. Arrange the a_{ij} so that they are stored in a vector $\hat{\mathbf{a}}$.
      The best way to do this will depend on the structure of the problem.
   2. Find the smallest k such that

$$\sum_{\ell=1}^{k} \hat{a}_\ell > r_2 a_{\text{tot}}$$

      then work backwards to determine the (i, j) values corresponding to k.
      How we find the (i, j) values corresponding to k will depend on how we 'linearized' the a_{ij} values.

The Kinetic Monte Carlo (KMC) Algorithm

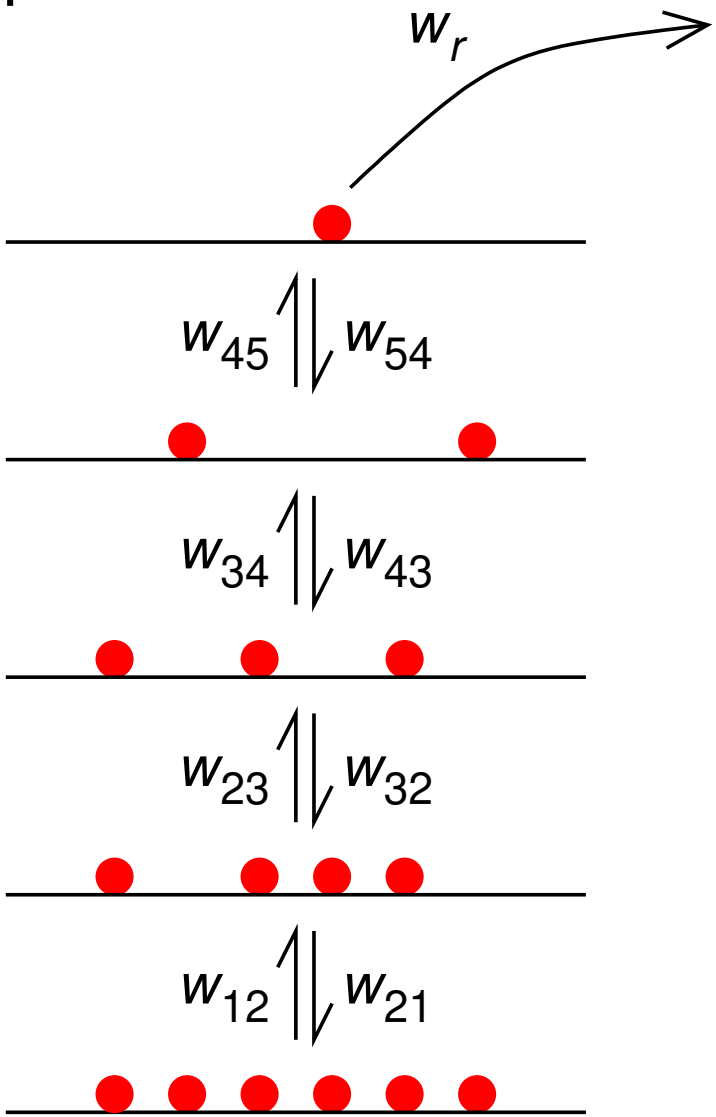
Simulating a population of molecules (continued)

6. Add Δt to the simulation time.
7. Adjust populations according to the jump:

$$N_i = N_i - 1$$
$$N_j = N_j + 1$$

8. Recalculate (at least) a_{ik} and a_{jk} $\forall k$.

A simple model of a unimolecular reaction



- Assume equally spaced levels

A simple model of a unimolecular reaction (continued)

- Strong collision assumption implies
  $w_{\text{down}} \equiv w_{21} = w_{32} = w_{43} = w_{54} = Z_A = \sigma \bar{u}_{\text{ref}} L[\text{B}]$ (B = bath gas)
- Assume $w_{\text{down}} = 10^9\,\text{s}^{-1}$.
- From the equilibrium conditions,
  $w_{\text{up}} \equiv w_{12} = w_{23} = w_{34} = w_{45} = Z_A e^{-\Delta\epsilon/k_B T}$
- Assume $w_{\text{up}} = 0.8 w_{\text{down}}$.
- w_r is related to the rate of IVR and crossing the transition-state dividing surface.
- Assume $w_r = 10^{13}\,\text{s}^{-1}$.
- rate constant $= \dfrac{\text{rate of reaction}}{\text{number of unreacted molecules}} = -\dfrac{1}{N}\dfrac{dN}{dt} = -\dfrac{d\ln N}{dt}$
  where $N = \sum N_i$.

A simple model of a unimolecular reaction (continued)

- Initial condition: $N_1 = N$, $N_2 = N_3 = N_4 = N_5 = 0$
- Organization of the $\hat{\mathbf{a}}$ vector:
  $[a_{12}\ a_{23}\ a_{34}\ a_{45}\ a_{21}\ a_{32}\ a_{43}\ a_{54}\ a_r]$

Some useful Matlab knowledge

- If you run a program repeatedly, there is a chance that data stored in previous runs will interfere with your current calculation. The `clear` command clears all variables.
- We can operate on parts of a vector by referring to ranges of indices, e.g. if `a` is the propensity vector of length 9, `N` is the population vector of length 5, and `wup` is a scalar,

  ```
  a(1:4) = wup*N(1:4);
  ```

  will calculate the propensities of the upward jumps and store them in the first 4 elements of the propensity vector.
- Because vector operations are very efficient in Matlab, it may be advantageous to do a few vector operations rather than write something more complicated to try to save arithmetic operations.
- The keyword `end` can be used as an index into a vector to denote the last element of the vector.

Some useful Matlab knowledge

Conditional execution

- The ability to only run code if certain conditions are met is central to a lot of programming.
  `while` loops provide a form of conditional execution.
- `if...elseif...else` structures are often useful.
  General syntax:

```
if condition
        code to execute
elseif another condition
        alternative code
else
        Default code
end
```

Some useful Matlab knowledge

Conditional execution (continued)

- `elseif` is optional, but you can have many `elseif` statements if needed.
- `else` is optional.
  If present, it is executed if none of the previous conditions are true.
- The conditions are tested in order. Once a true condition is found, the code in that block is executed, and then the program skips to `end` of the `if` control structure.
  No more than one statement in an `if` control structure will ever be executed.

Two powerful Matlab functions

`cumsum()` calculates cumulative sums of a vector.
Example: `cumsum(a)` returns the vector $(a_1, a_1 + a_2, \ldots, \sum_\ell a_\ell)$.
Note that a_{tot} is just the last element returned by `cumsum()`.

`find()` has many useful options for locating quantities matching a particular condition.
The KMC jump destination selection rule, finding the smallest k for which

$$\sum_{\ell=1}^{k} \hat{a}_\ell > r_2 a_{\text{tot}}$$

can be coded as follows in Matlab:

```
a_sums = cumsum(a);
atot = cumsum(end);
k = find(a_sums > r2*atot,1,'first');
```

Fitting and plotting in Matlab

`coeffs = polyfit(x,y,degree)` fits a polynomial of the selected degree to (x, y) data.
`coeffs` is a vector of coefficients stored in descending order of exponent.

`hold` tells Matlab to keep adding to a plot rather than erasing and starting over.
`hold off` tells Matlab that the plot is complete.