

Kinetic Monte Carlo Modeling with kmcos

Mie Andersen, Juan Manuel Lorenzi, Meelod Waheed

January 27, 2022

1	Introduction	3
1.1	What is Kinetic Monte Carlo	3
1.2	Features of kmcos	3
1.3	Installing Ubuntu on VirtualBox	3
1.3.1	Installing VirtualBox	3
1.3.2	Setting up Ubuntu on VirtualBox	3
1.3.3	Downloading Ubuntu Disk Image File	3
1.3.4	Installing Ubuntu on VirtualBox	3
1.3.5	Creating a Shared Drive Between Local and Ubuntu	4
1.3.6	Bidirectional Clipboard Between Local and Ubuntu	4
1.4	Creating a Python Virtual Environment on Ubuntu	4
1.5	Installing kmcos on Ubuntu	4
1.6	Installing Viewer GUI	5
1.7	Frequently Asked Questions	5
2	Visualization & Quantitative analysis	5
2.1	Scripting, Plotting and Visualization	5
2.1.1	Elements of Python Syntax	5
2.1.2	Scientific Computing Using The NumPy Package	6
2.1.3	Plotting Using Matplotlib	7
2.1.4	Visualizing atomic structures using ASE	7
2.2	First API Steps: Running kmcos Interactively. (review**)	7
2.3	Client Scripts	9
2.3.1	Generating an Arrhenius Plot	9
2.3.2	TASK2: TOF and coverages vs p diagrams	9
2.3.3	Relaxing the system	10
2.3.4	TASK 3: The effect of the initial state	10
2.3.5	TASK 4: Random initial state from guess coverages	10
2.3.6	TASK 5: Sensitivity analysis	10
2.3.7	TASK 6: ModelRunner	11
3	Building a kmcos model	11
3.1	The elements of a kmcos project	12
3.1.1	Project	12

3.1.2	Meta	12
3.1.3	Layer	12
3.1.4	Species	12
3.1.5	Parameter	13
3.1.6	Process	13
3.2	A model step-by-step: O2 adsorption / desorption.....	13
3.3	TASK 7: The ZGB Model	13
3.4	Modeling (lattice) diffusion	13
3.4.1	A simple ion diffusion model	13
3.4.2	Preparing the system	13
3.4.3	TASK 8: Implement the boundary conditions	14
3.4.4	TASK 9: Extending and testing the lattice diffusion model	14
3.5	Lateral interactions in kmcos	14
3.6	TASK 10: Solid-on-solid crystal growth model	14
3.7	TASK 11: Diffusion in the SOS model	14
3.8	TASK 12: Lateral interactions in the diffusion model.....	14
3.9	TASK 13: Defects in the diffusion model	14
4	References	15

1 Introduction

1.1 What is Kinetic Monte Carlo

Kinetic Monte Carlo is a special case of monte carlo methods. Monte carlo methods are techniques in which random numbers and probabilities are used to choose the next steps. Simple examples include dice and card game simulations. Kinetic Monte Carlo is a special case that assigns an elapsed time between each step. Kinetic Monte Carlo simulations can be used to simulate chemical kinetics including at the atomic/molecular scale.

- Picture of species of a surface (refer to local_smart)
- Add a trajectory
- New figure (powerpoint image/flowchart)
- Sentences about what info can be extracted
- Include info on expected inputs

1.2 Features of kmcos

- Snapshots
- Acceleration of quasi-equilibrium processes
- Sensitivity analysis
- Desired timesteps
- Machine Learning identification of local configurations

1.3 Installing Ubuntu on VirtualBox

It is recommended to install kmcos on Ubuntu within a python virtual environment, and our instructions are written accordingly. If you plan to use a windows machine, it is recommended to first get VirtualBox and to make an Ubuntu Virtual Machine.

1.3.1 Installing VirtualBox

Download VirtualBox with the link provided: <https://www.virtualbox.org/wiki/Downloads>. Note that VirtualBox 6.1.30 is used and confirmed to work with kmcos.

1.3.2 Setting up Ubuntu on VirtualBox

Open VirtualBox, then click on "New" to create a new virtual machine. Name the Ubuntu machine and select "Linux" as the type and select "Ubuntu (64-bit)" as the version. You will then be prompted to allocate the amount of memory to the Ubuntu virtual machine. We recommend that you allocate at least 8gb of ram if possible. Click next and select "Create a virtual hard disk now," click create, then select "VHD (Virtual Hard Disk)." Next, we'll want to dynamically allocate storage on our physical hard disk on the Ubuntu machine. The recommended is 25GB if you do not plan on installing Ubuntu updates later. If you wish to install Ubuntu updates later, we recommend allocating more than 25GB.

Once you create the Virtual Hard Disk, you will be able to see the Ubuntu machine on your VirtualBox dashboard.

1.3.3 Downloading Ubuntu Disk Image File

Download the Ubuntu LTS Desktop file with this link: <https://releases.ubuntu.com/18.04/>. Note that Ubuntu 18.04.6 LTS is used and confirmed to work for kmcos. Later versions of Ubuntu are not confirmed to work. Now go back to VirtualBox, click "Settings," "Storage," "Empty." In attributes, click the blue disk icon and add the Ubuntu disk file that was downloaded.

1.3.4 Installing Ubuntu on VirtualBox

Start the Ubuntu machine on VirtualBox and select the iso file you selected previously in settings. Once Ubuntu boots up, select "Install Ubuntu" when prompted on the screen. Select your keyboard layout, then check "Normal Installation" and uncheck "Download updates while installing Ubuntu" when prompted on

the next screen. It's important you uncheck "Download updates while installing Ubuntu" as we cannot guarantee the program will work with the option checked. On the next screen, check "Erase disk and install Ubuntu" and continue. Next, select your current location and set up your name and password on the next screen. After the installation process is completed, restart the Virtual Machine. Log in and you should see the Ubuntu desktop.

As a note: if you want to enter full-screen mode with the virtual machine, type right CTRL + F.

1.3.5 Creating a Shared Drive Between Local and Ubuntu

On the top left corner of the VirtualBox window, click on "Devices," then "Insert Guest Additions CD Image." We now want to add a user account to the group vboxsf. Go to the Ubuntu Terminal, and input these commands:

```
sudo usermod -a -G vboxsf "$USER"
```

Note that "\$USER" is the name of your user account profile in Ubuntu. Reboot the Ubuntu virtual machine completely.

Click on "Devices," then "Shared Folder." Select the folder you want to share between the Ubuntu and Windows PC on the Windows File Manager by clicking on "Other" on the "Folder Path" dropdown. Check off "Share Folder Options," and right-click on "Machine Folders." Check off "Auto-mount" and "Make Permanent."

1.3.6 Bidirectional Clipboard Between Local and Ubuntu

Click on "Devices" on the top right, then "Shared Clipboard" and click on "Bidirectional. You can also allow "Shared Drag and Drop" from the same menu as well. You should now have the ability to access the shared folder and bidirectional clipboard between your local and Ubuntu machines.

If the Ubuntu machine provides an error on the Guest Additions or the shared folder feature does not work, input this command to manually install Guest Additions:

```
sudo apt-get install virtualbox-guest-dkms
```

Then reboot the Ubuntu machine again.

If you run into an issue where the Ubuntu machine does not load after resetting, complete multiple full shutdowns and start the Ubuntu machine on VirtualBox.

1.4 Creating a Python Virtual Environment on Ubuntu

For all installations, we highly recommend installing kmcOS on Ubuntu within a python virtual environment, and our instructions are written accordingly. Enter these commands in the Ubuntu terminal to create a virtual environment in order:

```
cd ~
sudo apt-get update
sudo apt-get install python3
sudo apt-get install virtualenv
virtualenv -p /usr/bin/python3 ~/VENV/kmcOS
source ~/VENV/kmcOS/bin/activate
```

For all installations, be sure to be within the virtual environment created with the command: source ~/VENV/kmcOS/bin/activate. If you need to exit the virtual environment, enter 'deactivate' in the Ubuntu terminal. It is important that you always remain in the virtual environment when installing the following dependencies and further files in this documentation.

1.5 Installing kmcOS on Ubuntu

To fetch the latest version of kmcOS using git, enter these commands in the Ubuntu terminal:

```
sudo apt install git
git clone https://github.com/kmcOS/kmcOS-installers
cd kmcOS-installers
bash install-kmcOS-linux-venv.bash
```

You can also directly download the repository from <https://www.github.com/kmcos/kmcos> and then bash install kmcos from the terminal as well.

1.6 Installing Viewer GUI

To start up the viewer GUI, we need to download some dependencies from kmcos-installers:

```
cd kmcos-installers
bash install-kmcos-view-linux-venv.bash
```

Now we need to cd into ...kmcos/examples/MyFirstSnapshot_local_smart and run:

```
ipython
```

Run the following commands in ipython:

```
from kmcos.run import KMC_Model
model = KMC_Model()
model.view()
```

To control the zoom of the atoms, you can add an optional argument like this: `model.view(scaleA = 6)`.

1.7 Frequently Asked Questions

How to Turn off Automatic Updates on Ubuntu

Go to Software & Updates application in ‘Show Applications.’ Click on ‘Updates’ and select these options:

- For other packages, subscribe to: Security updates only
- Automatically check for updates: Never
- When there are security updates: Display immediately
- When there are other updates: Display every two weeks
- Notify me of a new Ubuntu version: Never

How to Create Snapshots on Ubuntu

Open the VirtualBox window and select ‘Machine -> Tools -> Snapshots.’ At the snapshots section, you can click ‘Take’ to save the current state of Ubuntu as a snapshot. From here, you have the option to restore your current Ubuntu machine to the snapshot or clone the snapshot to a new Ubuntu machine.

How to Close Windows That Will Not Close, Such as the View Window

Use this command on a new terminal, then click on the window:

```
xkill
```

2 Visualization & Quantitative analysis

2.1 Scripting, Plotting and Visualization

The kmcos application programming interface (API) uses the Python programming language. Documentation can be found at: <https://docs.python.org/3/>. Some basic usage and examples are provided below.

2.1.1 Elements of Python Syntax

1. Printing ‘Hello World’ on the terminal:

```
print('Hello World')
```

2. Importing modules:

```
import ... as ...  
from ... import ...
```

3. (Some of) python variable types:

Basic Types:

```
some_text = 'This' # strings  
some_integer = 134 # integers  
some_float = 0.8 # floating point numbers  
some_boolean = True # boolean
```

Lists:

```
L = ['a', 3, 4.7]  
L[0] # 'a'  
L[1] # 3
```

Dictionaries:

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict ['Age'] # 7
```

4. If statements:

```
if (condition):  
...  
elif (condition):  
...  
else:  
...
```

5. Flow control with for loops:

```
for i in range (3):  
    print(i)
```

Output:

0

1

2

```
for i, j in enumerate(['hat', 'cat', 'bat']):  
    print(i + ' ' + j)
```

Output:

0 hat

1 cat

2 bat

2.1.2 Scientific Computing Using The NumPy Package

The NumPy package contains many useful things such as N-dimensional array objects, linear algebra, and random number capabilities. Documentation can be found at: <https://www.numpy.org/>. The package is imported and used as follows:

```
import numpy as np
A = np.array([0, 1])
```

2.1.3 Plotting Using Matplotlib

The Python plotting library matplotlib can be used to create publication quality figures in a variety of formats. Documentation can be found at: <https://matplotlib.org/>. Any other plotting software can of course also be used, provided the kmcos output data are stored in a convenient format. The advantage of using matplotlib is the easy integration into Python client scripts for running kmcos.

2.1.4 Visualizing atomic structures using ASE

The Atomic Simulation Environment (ASE) is a set of tools and Python modules developed for atomistic simulations. It is used in kmcos to visualize the current state of a model. Documentation can be found at: <https://wiki.fysik.dtu.dk/ase/>.

2.2 First API Steps: Running kmcos Interactively.

While we can use the GUI viewer to get a general idea of the behavior of the model, quantitative analysis requires the use of the API. Be sure that you are inside the kmcos virtual environment before proceeding. You can do so with this command in the terminal: `source ~/ENV/kmcos/bin/activate`. To launch the API, we need to launch a terminal, go to the folder containing the compiled kMC model (COoxRuO2_local_smart) and run the command:

```
ipython
```

This will initialize `ipython`, an interactive Python interface. Inside this interface, we can load our model by writing

```
from kmcos.run import KMC_Model
model = KMC_Model()
```

These commands will give some text output (including the model's name and author), and create the model object, which will use to run our simulations.

A useful shortcut is to type:

```
kmcos run
```

in the folder containing the compiled kMC model, which directly opens the interactive shell and creates the model object.

The most elementary thing we can do with our model is to directly run it:

```
NSTEPS = 1e6
model.do_steps(NSTEPS)
```

which will run NSTEPS kMC steps (in this case 1 million). This will take a while, and then we will recover the (ipython) command prompt.

Below are listed some other useful commands:

1. Change the value of a parameter (e.g. temperature or pressure):

```
model.parameters.T = 550
model.parameters.p_COgas = 0.5
```

2. To print a list of the current values of all parameters and rate constants in the model, simply type:

```
model
```

3. Print current occupations (as averaged coverages):

```
model.print_coverages()
```

4. Print a list of the number of times each process has been executed since the beginning of the

simulation:

```
model.print_procstat()
```

5. Analyze the current state of the model:

```
atoms = model.get_atoms()
```

This command returns an ASE atoms object, which can be visualized by typing:

```
model.show()
```

The atoms object also contains some additional data:

```
atoms.occupation  
atoms.tof_data  
atoms.kmc_time  
atoms.kmc_step
```

The occupations and turn-over-frequencies (TOFs) come in the form of NumPy arrays. In order to get the headers for these arrays, type:

```
model.get_occupation_header()  
model.get_tof_header()
```

The TOFs that are affiliated with the atoms object are TOFs averaged over the simulated time since the last `model.get_atoms()` call. In contrast, `atoms.occupation` is the current occupation (identical to what is printed using the `model.print_coverages()` command).

6. Sample an average model and return TOFs and coverages in a standardized format:

```
model.get_std_sampled_data(samples, sample_size, tof_method='integ',  
output='str')
```

The parameter `samples` is the number of batches to average over. The number of kmc steps in each batch or sample is given by the `sample_size` parameter. In each case check carefully that the desired observable is sampled good enough! The parameter `tof_method` allows switching between two different methods for evaluating turn-over-frequencies. The default method 'procstat' evaluates the procstat counter, i.e. simply the number of executed events in the simulated time interval. 'integ' will evaluate the number of times the reaction could be evaluated in the simulated time interval based on the local configurations and the rate constant.

The output of `model.get_std_sampled_data()` is determined by the parameter `output`, which can be set to 'str' or 'dict'. The default 'str' returns a text string containing first the values of all adjustable parameters, then the TOF(s) and coverages, and finally the total sampled kmc time, the total simulated kmc time (including also the time simulated before the `model.get_std_sampled_data()` call), and the number of sampled kmc steps, all separated by spaces. This text string can be converted to a Python list (L) using the Python `split()` command:

```
data = model.get_std_sampled_data(samples=10, sample_size=1e6)  
L = data.split(' ')  
print(L)
```

Alternatively, `output='dict'` returns the above information in the form of a Python dictionary. For example, the TOF for CO oxidation can be retrieved using the command:

```
output_dict = model.get_std_sampled_data(samples=10, sample_size=1e6,  
output = 'dict')  
print(output_dict)  
TOF_CO = output_dict['CO_oxidation']  
print(TOF_CO)
```

7. Access to the Fortran modules:

The above commands are often sufficient when running and simulating a kmcos model, but in certain

cases direct access to the Fortran data structures and methods is desirable. The Fortran modules base, lattice, and proclust are attributes of the model instance kmc_model.so. This model instance can be explored using ipython and <TAB>:

```
model.base.<TAB>
model.lattice.<TAB>
model.proclust.<TAB>
```

8. Deallocating memory:

```
model.deallocate()
```

This command is important to call once a simulation has finished, if you want to run a new simulation within the same script, as it frees the memory allocated by the Fortran modules.

Reset model:

```
model.reset()
```

This command is called after `model.deallocate()` in order to reset the model to its initial state.

2.3 Client Scripts

We have already seen that the kmcos GUI is useful to quickly investigate model behavior, and that interactive use of the API allows for finer control and more precise quantitative evaluation. In everyday use however, the most useful way of using kmcos is through client scripts. With client scripts we can automatize the task we performed using the interactive interface.

2.3.1 Generating an Arrhenius Plot

As a first example of a client script, we will see how to build an Arrhenius plot (i.e. $\log(\text{TOF})$ vs. $1/T$) for the RuO₂ CO oxidation model. Be sure that you are in the kmcos virtual environment by entering this command in the terminal: `source ~/ENV/kmcos/bin/activate`. The example script `plot_arrhenius.py` can be found in the `intro2kmcos` repository, under the `solutions` and `COoxRuO2_local_smart` directories. To clone this repository, follow the command below:

```
git clone https://github.com/jmlorenzi/intro2kmos.git
```

Now we need to make sure we have the proper text files from the JANAF Thermochemistry tables. To do so, we need to create a directory called “janaf_data” anywhere on your python path. To see the directories on our python path. To do so, run this command on the terminal:

```
python -c"import sys; print(sys.path)"
```

From here, go into the site-packages sub directory in your python virtual environment directory to create the “janaf_data” directory. Inside the janaf_data directory, create a file named `__init__.py` so that python recognizes it as a module. Use this command to create a `__init__.py` in the janaf_data directory:

```
touch __init__.py
```

Then copy all the needed data files from the NIST website in the tab-delimited text format to the janaf_data directory. To start, a compiled list of JANAF tables can be viewed in `kmcos/tests/janaf_data`.

Now that we have the necessary files, go to the `task_material` directory in `intro2kmcos` and run the `COoxRuO2__build.py` file:

```
python3 COoxRuO2__build.py
```

This creates `COoxRuO2.xml` file that contains the data for generating the Arrhenius plot. Enter the `COoxRuO2_local_smart` directory in the terminal and run the following command:

```
python3 plot_arrhenius.py
```

2.3.2 TASK2: TOF and coverages vs p diagrams

The objective of this task is to write a client script similar to the one from the previous section. In this case, you need to plot both turnover frequency (TOF) and coverages (discriminated by site type and species) as a function of CO partial pressure, for constant T and oxygen partial pressure. The plot should cover values within $10^{-1} \text{ bar} < p_{\text{CO}} < 102 \text{ bar}$ (log-scale should be used).

You can use the previous example as a basis, and either directly use matplotlib or another plotting tool you know (if available) by writing to a file.

The strings that label the output important for this task are 'CO_oxidation', 'CO_rou2_bridge', 'CO_rou2_cus', 'O_rou2_bridge' and 'O_rou2_cus'. If writing output to a file, these values will end up in columns 3, 4, 5, 6 and 7, respectively, while 'p_COgas' is in column 1. All columns are counted starting from 0, according to python standards.

2.3.3 Relaxing the system

We either begin the kmc simulation with a clean surface or prepare the system in some user-specified initial state (see Sec. 2.3.5). In any case, this initial system state might be very different from the steady-state system state. It is therefore necessary to run a number of kmc steps to relax the system before any meaningful information about steady-state TOFs and coverages can be obtained. It should thus always be checked that the system has reached steady-state before calling `model.get_std_sampled_data()`! An example showing how this can be done is provided in the script `relaxation.py`.

Preparing the initial state of the system

Since it can be difficult to estimate if the system has reached steady state or not, it can be useful to test different initial states of the system to check that the same steady-state solution is always found independently of the initial state. The state of the system can be modified at any time during the simulation, but most often one would want to prepare a given initial state before beginning the simulation. The occupation of a site <site> can be changed to the species <species> using the command:

```
model.put(site=[x,y,z,model.lattice.<site>], model.proclist.<species>)
```

where x,y,z are the coordinates of the site. The above command can be quite inefficient if changing many sites at once, since each `put()` call adjusts the book-keeping database. To circumvent this you can use the `_put()` method instead:

```
model._put(...)
model._put(...)
...
model._adjust_database()
```

Uncomment lines 16-19 in the script `relaxation.py` for an example of how to use this.

2.3.4 TASK 3: The effect of the initial state

Try relaxing the model from different initial states, e.g. clean, CO@br, O@cus, etc., by doing the corresponding modifications to the script `relaxation.py`. Also try to vary the number of kMC steps taken in each sample (`sample_size`) and the number of samples (`Nsamples`).

2.3.5 TASK 4: Random initial state from guess coverages

In some cases, one might have a good guess of the steady-state system state in terms of (averaged) coverages. This could for example be obtained by solving the corresponding model in the mean-field approximation (MFA) using rate equations. Using a good guess for the final coverages can substantially speed up the time required to relax the system. Since the MFA does not consider lattice inhomogeneity, the best way to convert the MF coverages to a kMC lattice occupation is to assume a random occupation of the lattice sites. In order to do this, the guess coverages must be converted to the number of sites occupied by each species. For each species, the corresponding number of sites is then chosen randomly among the total number of available sites in the system using the Python `random.sample()` method. Take a look at the script `relaxation_random_initialization.py` and try relaxing the model from different random initial states.

2.3.6 TASK 5: Sensitivity analysis

In a catalytic system, the macroscopic TOF is often controlled by only one or a few microscopic rate constants. This can be quantified using the so-called Degree of Rate Control (DRC) [5]. Several definitions of this concept exist, but here we will be concerned with quantifying how much the change of the rate constant k of a single microscopic process i (+ for forward process, - for reverse process) affects the TOF

according to the formula:

$$x_i^+ = \frac{k_i^+}{\text{TOF}} \frac{\partial \text{TOF}}{\partial k_i^+} \bigg|_{k_{j \neq i}^+, K_{j \neq i}, k_i^-} \quad (1)$$

Take a look at the script `DRC.py`, which calculates the DRC for CO adsorption on the *cus* site. The derivative is calculated using finite differences, changing the rate constant by plus and minus 2% (the delta value). Try to play around with the delta value and the number of sample steps used to get an averaged TOF. Hint: finite differences are extremely sensitive to numerical noise and therefore require very long sampling times to sufficiently converge. Does your result match the result for the appropriate process and CO pressure in the lower panel of Fig. 4 in Ref. [6]?

2.3.7 TASK 6: ModelRunner

For some kMC applications you simply require a large number of data points across a set of external parameters (phase diagrams, microkinetic models). For this case there is a convenient class in the file `run_ModelRunner.py` to work with. For example:

```
from kmcos.run import ModelRunner, PressureParameter,
TemperatureParameter

class ScanKinetics(ModelRunner):
    P_O2gas= PressureParameter(1)
    T = TemperatureParameter(600)
    p_COgas = PressureParameter(min=1e-1, max=1e-2, steps = 20)

if __name__ == "__main__"
    ScanKinetics().run(init_steps=1e7, sample_steps=1e7, cores=4)
```

This script generates data points over the specified range. Using the `PressureParameter` or `TemperatureParameter` assures that the corresponding parameters will be sampled in a log- or reciprocal-scale, respectively. The script above runs several kmcos jobs synchronously (as many as indicated with the `cores` argument) and generates an output file with results (in this case called `ScanKinetics.dat`).

TASK: Try to reproduce the Arrhenius plot and the plots from Task 2 using the `ModelRunner` class. Compare the time spent by the original scripts with this one. Use as many cores as your workstation has.

Hint: `ModelRunner` uses a file to keep track of which calculations took place. If you need to restart the calculations from scratch, you have to remove the file ending in `.lock`.

Note that this `ModelRunner` may take 5 minutes to run.

3 Building a kmcos model

Now that we know how to run an existing kmcos model, we will learn how to generate a new model. While doing this we will learn about the way models are abstractly represented. We will also discuss how the models are saved and how the source code necessary for running the simulations is generated and compiled.

3.1 The elements of a kmcos project

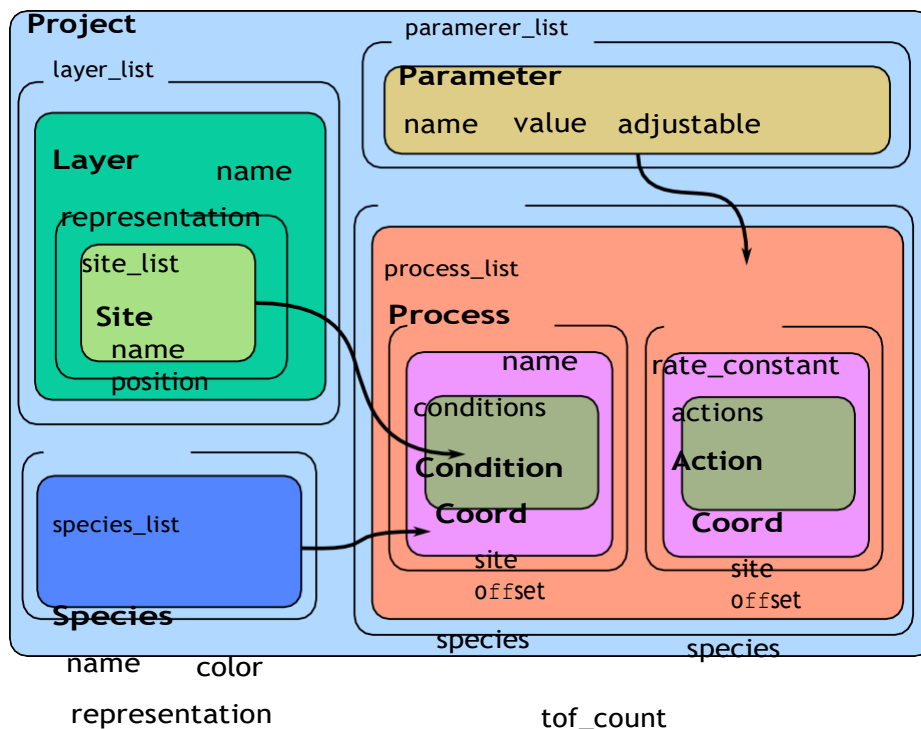


Figure 2: Structure of a kmcos model.

A kmcos model is built by putting together several building blocks. Those building blocks are found within the kmcos.types.

3.1.1 Project

The Project is the structure that contains all other elements.

3.1.2 Meta

Meta contains the meta-data of the project. This includes the model's name, the author's name and email and, importantly, the dimensionality (1D, 2D or 3D) of the system.

3.1.3 Layer

A kmcos Project contains a list of Layer objects. In this tutorial we will only consider models with a single layer, but more could be used. Each Layer has an unique name, an optional representation and a list of sites.

Site Each site has an unique name and a position in the unit cell.

3.1.4 Species

A Project also contains a list of Species. Each species needs to have an unique name. A color and a representation can additionally be added; these are useful while using the Editor and the GUI Viewer, respectively.

3.1.5 Parameter

A list of parameters is included in a Project. Each parameter needs a unique name and a default value. Parameters are used in the definition of the rate constants, and their values can be modified at run-time using the API or (if defined as adjustable) the Viewer GUI.

3.1.6 Process

The processes are perhaps the most complex structure in a kmcos model, and they are built using the elements previously discussed. Apart from a unique name, processes are composed of a list of conditions, a list of actions and a rate constant expression. Optionally, a process can also contain a `tof_count` attribute if it contributes to some of the turnover frequencies.

Condition A condition is defined by selecting a single species and a single coordinate. A coordinate is a site, but containing additionally information about its offset, i.e. the relative position of this coordinate with respect to a reference central coordinate. The list of conditions of a process encodes the necessary (local) occupation pattern necessary for the process to occur.

Action Like conditions, actions are also composed of a species and a coordinate. The list of actions of a process determines what is changed in the system state if the process is actually executed.

Rate constant Each process needs to be assigned a rate constant expression. These are given as strings that can contain common mathematical expression, any of the user-defined parameters as well some other predefined parameters and constants. All rate constants are evaluated when the model is first loaded and whenever the value of a parameter is changed.

3.2 A model step-by-step: O₂ adsorption / desorption

We will learn about the components of a kmcos models through a simple example: A model for oxygen adsorption and desorption into a fcc(100) (square) lattice.

3.3 TASK 7: The ZGB Model

The ZGB model [7] is a classic example of a kinetic Monte Carlo model of CO oxidation. It includes dissociative oxygen adsorption, molecular CO adsorption and a CO-O reaction to form gaseous CO₂. The only parameter in the model is the fraction of CO in the gas mixture y_{CO} . The CO adsorption rate per unit cell on empty sites is equal to y_{CO} . The Oxygen adsorption rate per unit cell is $(1 - y_{\text{CO}})$. Desorption of the reactants (CO and O₂) is neglected. To avoid deadlocks,

i.e. states in which no process can happen, desorption process with very small rate (i.e. 10^{-10}s^{-1}) should be included. CO₂ formation has “infinite” rate constant in the model, which can be modeled by using a very large value (i.e. 10^{10}s^{-1}).

Hint: Beware of double counting when implementing the oxygen adsorption process.

3.4 Modeling (lattice) diffusion

3.4.1 A simple ion diffusion model

Another type of system that can be modeled with kmcos is that of a set of particles diffusing on a lattice. Here we will consider a simple example of particles diffusing on a 2D square lattice. The example render script `render_LGD.py` can be found in the `task_material` folder.

A similar simple model in a realistic context (Li diffusion in graphene) can be found in [8].

3.4.2 Preparing the system

If used as given, trying to run the model will not work. This is because at start-up the system will be completely empty, a state in which no process can occur. To solve this problem there are two possibilities, depending on our objectives:

- One option is to prepare the system so that it contains a certain number of ions at the start, as was done in section 2.3.4. This is useful if one ones to study the relaxation of the system from the selected configuration, or to study equilibrium properties.
- Alternative, it is possible to impose special boundary conditions to the system. In this way it is also possible to study steady-state behavior under non-equilibrium conditions.

In this course, we will focus only on the second case.

3.4.3 TASK 8: Implement the boundary conditions

The objective of this task is to set up the boundary conditions for the diffusion model. This can be achieved by including a pair of auxiliary species ('source' and 'drain') and the corresponding entry and exit processes (as shown in the slides). Additionally, the `kmc_settings.py` file needs to be modified in two ways:

- First, the default system size should be changed from the default (20 × 20) to something that is longer in the direction of prominent diffusion, e.g. (50 × 20)
- Second, the `setup_model` function should be modified to place the sources and drains in the correct positions.

If you manage to set this up correctly, the model can be visualized using `model.view()` in ipython. To control the zoom of the atoms, you can add an optional argument like this: `model.view(scaleA = 6)`. Be sure that you are in the `kmc` virtual environment by entering this line on the terminal: `source ~/VENV/kmc/bin/activate`.

3.4.4 TASK 9: Extending and testing the lattice diffusion model

Extend the diffusion model to include the effects of an external electric field. For this, introduce a new parameter that measures the strength of the field, and modify the diffusion processes accordingly. Once this is set in place, test how the effects of field strength and particle concentration in the current (details in the slides).

3.5 Lateral interactions in kmc

Up to now we have only considered models with a relatively small number of processes. In some situations, however, it is possible that the rate constants depend not only on the class of process being executed, but also on the local environment around the adsorbates. For example, the particles in our diffusion problem could interact repulsively, changing the rates of diffusion. In these cases we say that there are lateral interactions in the system.

The standard way to treat this in `kmc` is to explicitly incorporate all different processes arising from the interactions. In order to do this, we can use the `itertools` python module, to programmatically explore possible local states. An example of this is presented in the slides.

3.6 TASK 10: Solid-on-solid crystal growth model

Another problem that can be studied with kinetic Monte Carlo is that of crystal growth. We will consider a very simple model of crystal growth: the Solid-on-Solid (SOS) model. The model we will consider only includes adsorption and desorption processes, neglecting diffusion. The adsorption rate is constant, but the desorption rate is affected by both the system temperature and by lateral interactions (details in the slides).

Interesting examples of use of kinetic Monte Carlo in studies of growth processes can be found in [9, 10].

TASK: Implement the SOS model in `kmc`. Simulate crystal growth for two different temperatures (350 K and 450 K) and observe the resulting structures in both cases.

3.7 TASK 11: Diffusion in the SOS model

Implement diffusion processes in the SOS growth model. Compare growth patterns in the model with and without diffusion. Consider also a model without desorption (only adsorption and diffusion).

3.8 TASK 12: Lateral interactions in the diffusion model

Recalculate the plots obtained in TASK 9 for the diffusion model that includes lateral interaction. Do this for different values of the lateral interaction strength.

3.9 TASK 13: Defects in the diffusion model

Extend the 2D diffusion model by adding a 'defect' species, that blocks diffusion. Generate an initialization script that prepares the system with defects randomly located. Test the effect of the presence of defects in the current. Do this for different defect concentrations.

4 References

- [1] Arthur F. Voter. Introduction to the kinetic monte carlo method. In Kurt E. Sickafus, Eugene A. Kotomin, and Blas P. Uberuaga, editors, *Radiation Effects in Solids*, pages 1–23. Springer Netherlands, Dordrecht, 2007.
- [2] Karsten Reuter. First-Principles Kinetic Monte Carlo Simulations for Heterogeneous Catalysis: Concepts, Status, and Frontiers. In Olaf Deutschmann, editor, *Modeling Heterogeneous Catalytic Reactions: From the Molecular Process to the Technical System*, pages 71–112. Wiley-VCH, Weinheim, 2011.
- [3] Michail Stamatakis. Kinetic modelling of heterogeneous catalytic systems. *Journal of Physics: Condensed Matter*, 27(1):013001, 2015.
- [4] Max J. Hoffmann, Sebastian Matera, and Karsten Reuter. kmcos: A lattice kinetic Monte Carlo framework. *Comput. Phys. Commun.*, 185(7):2138–2150, July 2014.
- [5] Carsten Stegelmann, Anders Andreasen, and Charles T. Campbell. Degree of rate control: How much the energies of intermediates and transition states control rates. *Journal of the American Chemical Society*, 131(23):8077–8082, 2009.
- [6] Hakim Meskine, Sebastian Matera, Matthias Scheffler, Karsten Reuter, and Horia Metiu. Examination of the concept of degree of rate control by first-principles kinetic monte carlo simulations. *Surface Science*, 603(10–12):1724 – 1730, 2009. Special Issue of Surface Science dedicated to Prof. Dr. Dr. h.c. mult. Gerhard Ertl, Nobel-Laureate in Chemistry 2007.

- [7] Robert M. Ziff, Erdagon Gulari, and Yoav Barshad. Kinetic phase transitions in an irreversible surface-reaction model. *Physical Review Letters*, 56(24):2553, 1986.
 - [8] Werner Lehnert, Wolfgang Schmickler, and Ajit Bannerjee. The diffusion of lithium through graphite: a Monte Carlo simulation based on electronic structure calculations. *Chem. Phys.*, 163(3):331–337, July 1992.
 - [9] Peter Kratzer, Evgeni Penev, and Matthias Scheffler. First-principles studies of kinetics in epitaxial growth of III–V semiconductors. *Applied Physics A*, 75(1):79–88, 2002.
- P. Kratzer and M. Scheffler. Reaction-Limited Island Nucleation in Molecular Beam Epitaxy of Compound Semiconductors. *Physical Review Letters*, 88(3), January 2002.
- [10] Pavel Šmilauer, Mark R. Wilby, and Dimitri D. Vvedensky. Reentrant layer-by-layer growth: A numerical study. *Phys. Rev. B*, 47(7):4119–4122, February 1993.