Hands-on exercise 2

Molecular Dynamics with ASE, ASAP and Documentation

Instructions

1.1 Preface

The purpose of these exercises are to give the project team a starting point for approaching the ASE and ASAP as libraries for handling the time-sensitive lowest-level code in a molecular dynamics program as well as get some training in documentation -- especially embedded documentation and auto-generated documents from that inline documentation. At the very end after these instructions is a section *"Milestone after the exercise"* that describes where the team as a group is meant to be after these exercises. The exercises are meant to be done in groups of 1-2 persons.

This is a "Hands-on Exercise". Feel free to explore and tinker as you go through the steps and ask the teaching assistant(s) for input. The session will likely be less meaningful if you just rush through the steps without stopping to think and explore. If you are not previously familiar with the covered topics, you should not expect to reach the end during the hands-on session itself. However, the whole team is meant to reach the required state described in the section at the end *"Milestone after the exercise"*, so you may need to return to the material at your own time.

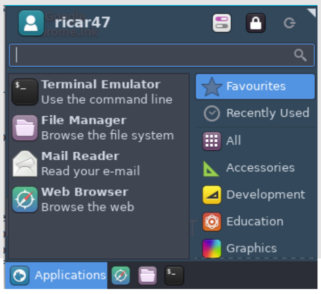
The two topics are placed in separate sections below. You are recommended to jump between the sections to get started on both topics.

1.2 Terminal windows and the command line

1. Open a new terminal/command window. If you are using a LiU Linux lab computer, run the "Terminal Emulator" program (may be an icon in the task bar, or you should be able to find it in the Applications menu.)

2. Create a working directory for the lab (if you already created it for the previous exercise you can skip the `mkdir` command):

```
mkdir project-hands-on
cd project-hands-on
```



(3) Initialize a new git repository for this hands-on-exercise

```
mkdir hands-on-3
cd hands-on-3
git init
```

(4) If you are using a LiU Linux lab computer, run the following command to initialize the environment to get access to ASE and ASAP Python modules (note that, for technical reasons, this references another course directory TFYA74).

```
    source /courses/TFYA74/software/bin/init.sh
```

Short review of Linux/Unix commands

In the terminal window you write one command at a time and then press the `enter` key to run it. Here follows some essential commands:

Two hints that can greatly speed up your use of the command line is: *(i)* the arrow up and arrow down key allows you to navigate through past commands you have issued. *(ii)* By pressing the "tab" key the terminal tries to automatically fill in the rest of a filename (and many other things) that you are about to enter. If there are several alternatives that starts the same way you can see them by pressing tab two times quickly.

Commands for files and directory navigation

pwd Each terminal window has a "current working directory", usually phrased as "the directory you are currently in." The command `pwd` shows the path to, and name of, this directory. The command is an abbreviation of "print working directory". The same information is often present as part of the command prompt you see in the terminal window.

ls Shows a list of the files and subdirectories of a directory. Just typing `ls` with no options show the files in your current working directory. The command `ls directory1` shows the files in the directory `directory1`.

cd Changes the working directory. If you write `cd directory1` you change the current working directory to be `directory1`. This directory must be a direct subdirectory to your present current working directory. You can also ascend from your current working directory by the command `cd ..`. (The directory name `..` is a general identifier for the parent of a directory.) If you just issue `cd`, your current working directory is changed to your home directory.

cp The command `cp file1 file2` copies `file1` to an identical copy `file2`. If you want to copy a whole directory with all its contents, use `cp -r directory1 directory2` (`-r` is for recursive).

mv The command `mv file1 file2` moves (i.e., renames) `file1` to `file2`. You can also move a file to another directory: `mv file1 dir1/filex`. This moves the file `file1` into the directory `dir1` and simultaneously renames it to `filex`. If you do not what to change the name of the file it is sufficient with `mv file1 directory`.

rm The command `rm file1` removes the file `file1`.

rmdir The command `rmdir dir1` removes the directory `dir1`. Note that a directory *must be empty* before it can be removed this way.

mkdir The command `mkdir dir1` creates a new directory with the name `dir1`.

man This command gives you a help text about other commands. For example, command `man mkdir` explains all the options for the command `mkdir`.

Other useful commands for this exercise

gedit A simple text editor with a graphical user interface. (Note: sometimes when gedit is started, it prints warnings in the terminal windows. These are usually irrelevant.)

nano A text editor without a graphical user interface, i.e., it allow you to edit a text file inside the terminal window.

git Commands using the command line interface for git that we will cover in this exercise are start with `git`.

A good portal for more help is available here: http://linuxcommand.org/

Molecular dynamics with ASE, ASAP

2.1 ASE: Building systems and simple visualizations

5. Start a Python interpreter that you can give direct commands to; try `ipython3` for a nicer interface if available, otherwise use `python3`.

6. Build and visualize a water molecule:

```
from ase.build import molecule
from ase.visualize import view
h2o = molecule("H2O")
view(h2o)
```

This opens ASE:s somewhat primitive default visualizer. You can rotate the water molecule by holding down the right mouse button. You can zoom in and out with the mouse wheel. ASE integrates with other visualizers, but the default visualizer is built-in into ASE and thus does not require an external installation.

7. Lets build copper (Cu) in its natural fcc crystal structure:

```
from ase.build import bulk
cu_prim = bulk("Cu", "fcc", a=3.6)
view(cu_prim)
```

This shows the primitive cell representation of fcc copper. It seems the atom does not move when you rotate, only the "unit cell" dashed lines. Do you understand why? (Otherwise, discuss with the hands-on assistant.)

(8) Lets see the cubic non-primitive unit cell version of fcc Cu:

```
cu_cube = bulk("Cu", "fcc", a=3.6, cubic=True)
view(cu_cube)
```

Do you understand what you see? (Otherwise, discuss with the hands-on assistant.)

(9) Lets build a cubic supercell of Cu:

```
cu_super = cu_cube*(4,4,4)
view(cu_super)
```

(10) Lets also try a nanotube:

```
from ase.build import nanotube
cnt1 = nanotube(6, 0, length=4)
view(cnt1)
```

The ASE functions used so far simplifies building the most common systems. You can see some documentation about what is available here:

- https://wiki.fysik.dtu.dk/ase/ase/build/build.html

As you can see on that page, the "See also" box reference other modules to build somewhat more complex systems. One of those is the `ase.lattice` module, documented here:
https://wiki.fysik.dtu.dk/ase/ase/lattice.html

(11) Build a 4x4x4 supercell of Cu using the lattice module:

```
from ase.lattice.cubic import FaceCenteredCubic

atoms = FaceCenteredCubic(
          directions=[[1, 0, 0], [0, 1, 0], [0, 0, 1]],
          symbol="Cu",
          size=(4, 4, 4),
          pbc=True)
```

(12) Take a look at it using the ASE visualizer.

Now we will test reading a structure from a file. In the previous hands-on exercise we met the materialsproject.org website and downloaded large sets of crystal structures from there. This time we start with just downloading a single one.

(13) Go to the materialsproject.org website and log in (create a new account if necessary).

(14) Go to "explore materials" (the big periodic table of elements)

(15) Click a 2-3 elements on random, and you should get a table of materials appearing below.

(16) Click on the topmost material (it is the most stable one with that composition)

(17) On the right hand side there is a label "Final Structure" with a download ("cif") button. Click on that, select "Computed". You get a cif file.

(18) Move the cif file into the directory where you are doing the exercise.

(19) Load the cif file into ASE and visualize it by:

```
from ase.io import read
mpstruct = read("<filename>")
view(mpstruct)
```

(20) Experiment with building a supercell of this material and view it.

(21) Exit the `ipython3` program by writing `quit`.

2.2 ASAP: Your first MD program

ASAP molecular dynamics example programs are available in these two places:

- https://wiki.fysik.dtu.dk/asap/Examples
- https://wiki.fysik.dtu.dk/ase/tutorials/md/md.html

(22) Visit the second link and copy the **second** example on that page into a python file `md.py`.

Study the steps of the program to get a rough idea about what is going on (note that we covered a very similar example quite extensively in one of the lectures.) Make sure the `use_asap` variable is `True`.

(23) Run the program: `python3 md.py`.

Does the overall behavior of the potential, kinetic, and total energy behave as discussed in the lecture?

Do you see the potential and kinetic energy reaching equilibrium?

(24) Try to run the program once with `use_asap` set to `False` and note the difference when running. By looking at the code, note also the difference in what is being run for the two cases. Do you understand what is happening?

Make sure to change back to `use_asap = True`

(25) Play with the time step length in `dyn = VelocityVerlet( ...`, the total number of steps in `dyn.run(200)`, and/or the `interval=10` setting to more precisely simulate the early time development of the system. Can you see the process of the system reaching equilibrium better this way?

(26) Lets save the atomic positions so we can look at them.

Add the following line to the top of the file:

```
from asap3 import Trajectory
```

Add the following line to below the line where `dyn` is defined:

```
traj = Trajectory("cu.traj", "w", atoms)
dyn.attach(traj.write, interval=10)
```

27 Run the simulation again.

28 Look at the trajectory with the command line command:

```
ase gui cu.traj
```

A number of windows open, both for looking at a video of atomic movement, but also for simple plotting of key quantities as function of time step.

The plot that shows up displays `i, e-E[-1]`, which means that the x-axis shows the snapshot index `i`, and the y-axis shows the difference between the total energy at time step `i` and the total energy at the final time step.

29 Use the controls in the "Film" window to animate the atomic movement.

30 Find the `Graphs` window and use the `Help` button to see what other quantities are available for plotting (note: the Help window may contain more lines than visible and, if so, you can scroll with the mouse scroll wheel.)

Plot the time evaluation of the kinetic energy, the potential energy, and the temperature. Draw the plots using the upper `Plot x, y1, y2, ...` button. Try to draw the quantities both individually, and in the same diagram.

You may also want to try to draw trajectories with the original settings (200 steps of 5 fs logged at every 10 step) and settings that better shows the process reaching equilibrium.

Think a bit over why the plots look this way. Discuss with the hands-on exercise assistant if you want.

You can see more info about options for `ase gui` here:

- https://wiki.fysik.dtu.dk/ase/ase/gui/basics.html

1. Commit the final state of your md.py result to your git repository.
2. You can now feel free to play around with different time step settings, different structures, atom types, etc. You can always revert your changes by `git checkout md.py`.
   Can you increase the temperature and set the steps so that one can visibly see atoms change places in the visualization?

Documentation

3.1 Preparation

31 Fork the documentation exercise repository on GitHub by first going to the following URL: `https://github.com/comp-phys-proj/hands-on-2-docs.git` and then clicking on the button labeled "Fork" in the upper right corner.

32 Clone your fork of the hands-on-2 repository by finding its URL under the green "Code" dropdown button and then issuing:

```
cd ~/project-hands-on
git clone [URL]
cd hands-on-2-docs
```

3.2 Embedded documentation

The hands-on-2-docs repository is setup as a source code repository of a very simple Python program using ASE to calculate energies.

33 Take a look at `src/hdmolecule.py` to see how it is organized and how it works.

Note at the top a typical example of a copyright and license header, as covered in the lectures.

Second, you will see the declaration of a Python class `HDMolecule`. Note the inline class documentation that follows. Look at the first method, the construtor `__init__`, and note the inline method documentation.

34 Enter the `src` directory with `cd src`, then start up the python interpreter (`python3`). We will now import the module and try the `analyze_exp_N2` function:

```
import hdmolecule
from hdmolecule import HDMolecule, analyze_N2, analyze_exp_N2

analyze_exp_N2()
```

35 Check out how Python can use the embedded documentation to provide help:

```
help(hdmolecule)

help(HDMolecule)

help(analyze_exp_N2)

help(analyze_N2)
```

Note that the last one is not particularly helpful, because that function lacks inline documentation.

(36) Add inline documentation to `analyze_N2` the same way as the already documented functions. Explore it in Python with help(...)

IMPORTANT: updating the environement with the changes you do to the module requires restarting the python interpreter, you cannot merely re-import the module.

(37) Stage, commit, and push this change up to your fork of the repository.

(38) To see how releases work together with publishing documentation later, we will now mark this change as a new release of this software:

```
git tag -a "v0.2" -m "Release v0.2"
git push --tags
```

Note: if you have the program `gpg` set up to do cryptographic signatures, you should replace the `-a` option (for "annotated tag", - which is how to indicate a release in Git) with `-s` (for "signed annotated tag"). The latter attaches a cryptographic signature to the release with which one can later validate the release.

When you are finished, you can check with `git log` to see that the previous commit has a `v0.1` tag, and your edit a `v0.2` tag.

3.3 Repository documentation files

(39) Add a README.md or README.rst with the apropriate content for this repository (note: there may be a bit of advantage later in this exercise to use the restructuredText format).

Some help with these formats:

- https://guides.github.com/features/mastering-markdown/
- https://docutils.sourceforge.io/docs/user/rst/quickstart.html

In particular, in rst you can add an automatic table of contents with:

```
.. contents:: Table of Contents
```

(40) Make sure the repository also covers the other files we talked about on the lecture, i.e., add a CHANGELOG, CONTRIBUTING, and - if you want - a CODE_OF_CONDUCT file with apropriate contents.

3.4 Documentation generation with Sphinx

Here we will set up a docs directory to use Sphinx to provide documentation for our project.

(41) Make sure your current working directory is the root of your local copy of the Hands-on-2-docs repository.

(42) Create a docs directory and run the `sphinx-quickstart` utility:

```
mkdir docs
cd docs
sphinx-quickstart
```

Select the default answers (by pressing enter) for the many questions except for the ones below:

- For `Project name:`, enter `MDMolecule`.
- For `Author name(s):` enter `MDMolecule developers` and/or your own name.
- For `Project release`, enter `0.2`.
- We want the following Sphinx extensions enabled: `autodoc`, `mathjax`, `viewcode`, and `githubpages`.

The last one allows you to use equations marked up with `:math:` in your documentation.

If you make a misstake and want to restart, just remove the docs directory, re-create it, and re-start `sphinx-quickstart`.

(43) Check out the directory contents.

(44) Edit index.rst and add under `Welcome to MDMolecules documentation!`, under the `toctree` segment, add the same info as in your README.md (but note that the format now is restructuredText). Leave the rest of the file as it is.

(45) Build the docs:

```
sphinx-build . _build
```

(46) Open `_build/index.html` in a web browser, e.g.:

```
firefox _build/index.html
```

(47) Try to change the theme by editing `conf.py` and changing the line with `html_theme` to:

```
html_theme = "sphinx_rtd_theme"
```

(48) Regenerate with:

```
sphinx-build . _build
```

Note: you may get an error saying *"sphinx_rtd_theme is no longer a hard dependency"* in which case you need to install this theme. On the LiU lab computers you can do:

```
pip install --user sphinx_rtd_theme
```

If you are on a Linux Ubuntu distribution, you can instead do:

```
apt install python3-sphinx-rtd-theme
```

(49) Check the output again with, e.g.:

```
firefox _build/index.html
```

Note how the `Index` and `Module Index` links do not yet work. We will now look into auto-generating pages from the embedded documentation.

(50) We must first update `conf.py` so that the Python paths are set correctly to allow the source code to be scanned properly. Find the `Path setup` section at the top of the file. Uncomment the lines and change them so that they look like this:

```
import os
import sys
sys.path.insert(0, os.path.abspath('../src'))
```

(51) Now we invoke a sphinx tool to generate documentation "stubs" for all Python modules it can find under the `src` directory in our repository and place them under `docs/source`:

```
sphinx-apidoc -o source/ ../src
```

(52) Now, re-generte the documentation again:

```
sphinx-build . _build
```

Note: you will get a consistency warning about modules.rst not being "included in any toctree." You can ignore that warning.

(53) Again, open `_build/index.html` in your web browser (or reload, if you already have it open). Explore the Index, and Module Index sections of the documentation.

(54) Feel free to play around a bit with modifying the embedded documentation, and see how that is reflected in the Sphinx outout. For example: add an equation to the documentation using either of the following syntaxes:

```
:math:`\int_0^{42} f(x) dx`

.. math::

   \int_0^{42} f(x) dx
```

3.5 Hosting documentation with Read the docs (optional)

This part of the exercise explores the use of the service *Read the docs* to host repository documentation. This can be a nice way to make on-line browsable reference documentation available for both current and past releases of your program.

However, you will learn more about how to run automated repository "Actions" in a later hands-on exercise on automated testing. These "Actions" were originally targeted towards testing, but allow operations that can be used for many different things, including generation of documentation websites. Hence, the use of *Read the docs* and similar services are becoming less relevant in practice.

55. Make sure your current working directory is the root of your local copy of the Hands-on-2-docs repository.

56. For Read the Docs (and various other hosting options) we need to push the documentation to GitHub repository:

```
git add docs
git commit -m "Add sphinx documentation"
```

57. Create an account at Read the Docs: https://readthedocs.org/

58. Click 'Import a project', connect to GitHub, find and select your cloned `hands-on-2-docs` repository.

59. Do not change any of the default options.

60. Click on the "Versions" tab.

61. Click "Activate" on v0.2 and v0.1 and select "active".

62. Go back to "Overview", and at "Build a version" select "v0.2" and click "Build"

63. Wait for the green "Build completed" label to appear.

64. Check out the docs with the "View docs" button.

    Note the dropdown in the lower left corner where you can select between versions. Check out the function you added documentation to between version v0.2 and v0.1.

    IMPORTANT: It can take quite long time (10 min) between the read-the-docs build, and the sphinx web pages actually being visible at "View docs". If you see "This is an autogenerated index file. Please create an index.rst or README.rst file...", hold on for some time to see if the site updates.

Milestones after the exercise

In the next team meeting after this hands-on-exercise the team should verify that the following holds:

1. The `develop` branch of the central development repository has merged the tutorial ASE and ASAP program from this exercise (or something on similar funcitonal level).
2. All team members are able to `pull` this version to their own development machines and *execute it* -- meaning the ASE and ASAP libraries are installed and work.
3. All team members have been present at and/or reviewed the lecture on the inner workings of a molecular dynamics program so that they have an understanding of what the instructions in the tutorial example are doing.
4. All team members agree to document the code as taught in this exercise as part of their merge requests.

If any team member is struck on something in this list, they must engage the other team members to help, or ask the teachers in the course for help.