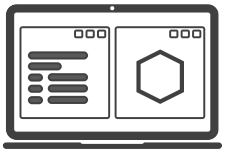

Introduction to Computational Chemistry
Handout Part 2

Eno Paenurk & Patrick Finkelstein & Felix Pultar

May 3, 2023

1 The `conda` Package Manager

`conda` is a package manager for software that is not installed on your cluster. A selection of useful `conda` commands:

Command	Description
`conda list`	Prints packages installed in current environment
`conda env list`	Prints available environments
`conda activate icc-2023`	Activates the environment "icc-2023"
`conda deactivate`	Deactivates current environment and goes to "base". If "base" is already active, deactivates `conda`
`conda info`	Prints some information on where `conda` and its environments are installed
`conda create -n dummy`	Creates the new environment "dummy" (not required for installation on Euler. Use only for installations on Grace or your local machine.)

If you don't want `conda` be activate every time you log in, run:

```
conda config --set auto_activate_base false
```

You will have to manually activate `conda` later using the full path.

2 Python Programming

`Python` is a general purpose programming language that is used to describe your reactions programmatically to `autodE`. Scripts are executed line by line from top to bottom. Here is a short explanation of the most important constructions used in the provided `Python` script.

Command	Description
`from icctools import librxn`	Use code provided by the `librxn` module of `icctools`
`rxn_smiles = "CCl.[F-]>>CF.[Cl-]"`	Assigns the variable `rxn_smiles` a string value
`rxn_temperature = 298.15`	Assigns the variable `rxn_temperature` a numerical value
`if condition:`	Runs following indented lines if condition(s) are met
`print(values, separated, by, comma)`	Prints the value(s) in the parentheses
`rxn.calculate_reaction_profile(...)`	Calculates reaction profile of variable `rxn` (parameters omitted here for brevity).
`librxn.print_results(rxn)`	Prints the results of the calculation performed

3 Setting Up Calculations with `autodE` and `icctools`

1. Activate the `conda` environment and load `ORCA` (+ dependencies)
2. Come up with one or more mechanistic hypotheses and identify each elementary reaction, i.e., each arrow push
3. Create a folder for your overall reaction, e.g. "my-aldol-reaction"
4. Create a subfolder for every elementary step, e.g. "nucleophilic attack" or "collapse-tet-intermediate"
5. Copy the `sample-autode.py` file into each subfolder using the command line (command `cp`) or FileZilla / MobaXterm
6. Rename the files, e.g., `nucleophilic-attack.py`
7. Provide values for `rxn_name`, `rxn_smiles`, `rxn_solvent` (see table in exercises), and `rxn_temperature` (Kelvin) in the in the `reaction-name.py` file. `SMILES` strings can be generated using ChemDraw.
8. Provide values for the computational resources via the variables `n_cores` and `memory_per_core`, and choose the computational method.
9. Submit the calculation using the `subade.sh` script.

```
subade.sh reaction-name.py
```

Here is an overview over the different variables you can set during calculations with `icctools`:

Variable	Description
`rxn_name`	Name of the reaction, shoudl match the `Python` filenname
`rxn_smiles`	`SMILES` representation of the reaction
`rxn_solvent`	Reaction solvent as string, see exercise
`rxn_temperature`	Reaction temperature in Kelvin
`n_cores`	Number of processing cores
`memory_per_core`	Number of memory per core in MB
`method`	Computational method (see below)

Note: If your calculation times out, it is easiest to use (a) longer times and/or (b) more cores.

Here is a list of computational methods that you can use. Stay tuned until later in the course to find out more about what these acronyms mean:

Python Code	Description
`librxn.Method.XTB`	semiempirical (very fast)
`librxn.Method.BP86`	cheap DFT (reasonably fast)
`librxn.Method.B3LYP`	DFT of choice for organic chemists (slow)
`librxn.Method.PBE0`	DFT of choice for inorganic chemists (slow)

Once your calculation has terminated successfully, there should be a `.pdf` in each subfolder with the plot of the reaction profile. Consult the overview table in the exercises for a list of files generated by `autodE`.

4 Possible Problems with `autodE`

If the calculation failed, it is worthwhile to go through `autode.log` and other `.out` and `.err` files. Common reasons for failure are:

1. Unbalanced reaction schemes
2. Modules and `conda` environment not loaded
3. Insufficient resources or time out
4. Syntax error in Shell or `Python` script, e.g. missing quotation marks around `SMILES` strings
5. Problem copying `SMILES` strings: try copying each molecule individually and separate them using the dot (.) and >>. Make sure the ChemDraw molecule is what you want (expand labels, check for automatically added hydrogens, ...)
6. Inadequate QM method for the problem (see following weeks)
7. Compounds with "challenging" molecules (e.g., some small rings, some metal complexes, radicals, ...)

Especially the last point can be frustrating and difficult to predict beforehand.

5 Useful links

- `https://conda.io/en/latest/miniconda.html`
  `Miniconda` downloads page
- `https://docs.python.org/3/tutorial/`
  `Python` tutorial for those who want to know more
- `https://duartegroup.github.io/autodE/index.html`
  `autodE` documentation
- `https://onlinelibrary.wiley.com/doi/epdf/10.1002/anie.202011941`
  `autodE` paper including supporting information with examples galore
- A good comprehensive overview of calculations of reaction mechanisms:
  `https://pubs.acs.org/doi/10.1021/acs.organomet.8b00456`