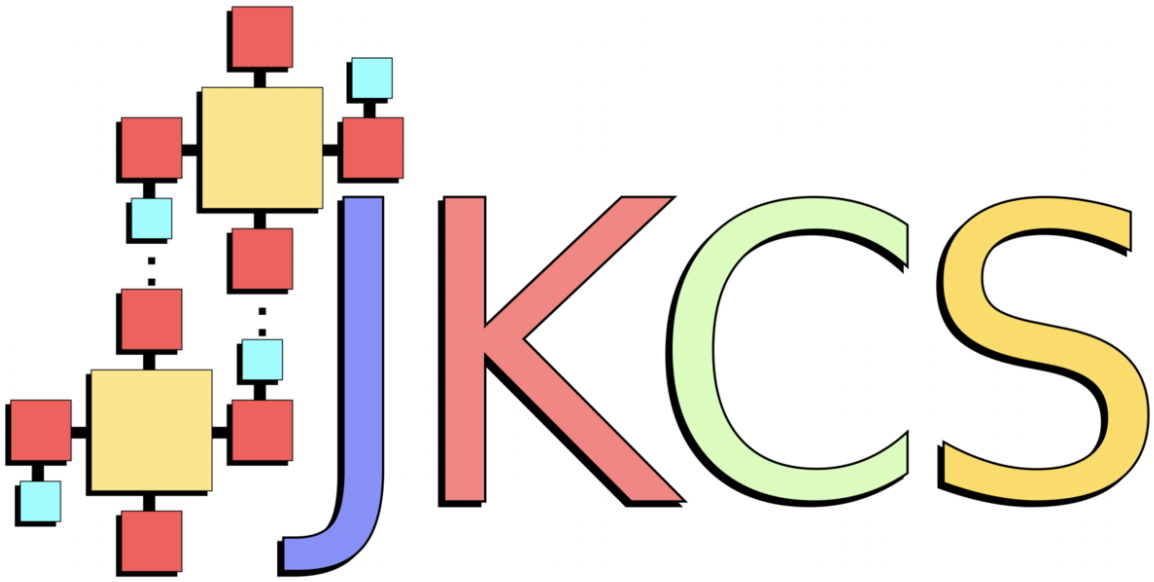
JKCS

Release 3.0

Jakub Kubečka

Apr 01, 2024

GET STARTED



This is the online documentation for JKCS = Jammy Key for Configurational Sampling.

JKCS The JKCS program can be on GitHub. Installation of JKCS enables JKQC and JKML, too.

Installation Click here to see the installation tutorial.

Citation I would appretiate whetheher you could cite our paper (see link). Nevertheless, you must cite all papers for the methods and applications you use on the way toward studying your molecular clusters. (See section Citations.)

Questions / comments If you have questions, feel free to contact me: ja-kub-ecka@chem.au.dk

Thanks I would like to thank all people who helped me to develope, improve, and debug the JKCS/JKQC/JKML program: Vitus Besel, Ivo Neefjes, Yosef Knattrup, and others

Thankx also goes to the newest contributors: Daniel Ayioubi (for JKTS), and Haide Wu (for JKout2xyz)

CHAPTER

ONE

MANUAL STRUCTURE

1.1 Setup & Installation

Note: In order to install and use JKQC and JKML you need to install JKCS!

1.1.1 Requirements

- git
- Python >3.8 but <4.0

Depending on your need other programs are required too (e.g. ABCluster, XTB, Gaussian, ORCA)

1.1.2 JKCS

Clone the JKCS from github, (modify setup [see the hint below first]), and run `setup.sh` (this might take a while):

```
git clone https://github.com/kubeckaj/JKCS2.1.git
cd JKCS2.1
sh setup.sh -help  #see the help first
sh setup.sh -r     #-r = rewrites your current ~/.JKCSusersetup.txt setup
#or: sh setup.sh -r -python python3.9 -module "module load python-data/3.9"
```

Once you modify all paths in `~/.JKCSusersetup.txt`, you can check if everything that you need is set properly by:

```
sh test.sh
```

If something fails, you must modify `~/.JKCSusersetup.txt` in order to use the JKCS features.

Hint: Users of some clusters (Puhti, Mahti, Grendel) can use predefined paths and python modules by typing:

```
sh setup.sh puhti -r   #for Puhti users
sh setup.sh grendel -r #for Grendel users
sh setup.sh mahti -r   #for Mahti users
```

Note: If you need to resetup the JKCS use *-r* or *-r2*, e.g.:

```
sh setup.sh -r grendel   #reinstall all python libs and rewrites ~/.JKCSusersetup.txt
sh setup.sh -r2 grendel  #only rewrites ~/.JKCSusersetup.txt
```

1.1.3 JKQC

All features are automatically installed with JKCS, yeay :-D

1.1.4 JKML

In order to use JKML, you must install some Python packages:

Note: *-qml* will install all libraries for quantum machine learning (QML)

-nn will install all libraries for neural network modelling (SchNetPack)

-descriptors will install dscribe library for some JKML extra features (*-sampleeach*)

1.1.5 3rd-party programs

ABCluster

ABCluster can be obtained from http://www.zhjun-sci.com/software-abcluster-download.php The online manual is available at http://www.zhjun-sci.com/abcluster/doc/ Modify the following lines in the `~/.JKCSusersetup.txt`:

```
PATH_ABC="[-ABCluster-folder-path-]"      #e.g.: "/users/kubeckaj/ABCluster-2.0-Linux/"
MODULE_ABC="module load gcc"              #e.g.: "module load gcc/8.2.0" || "module load␣
↪GCC/8.2.0-2.31.1"
```

If you want to use the ABCluster program by yourself, put the following lines to your `~/.bashrc` file:

```
export PATH=$PATH:[-ABCluster-folder-path-]
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:[-ABCluster-folder-path-]
```

then you should be able to use ABCluster, e.g.:

```
#source ~/.bashrc
module load gcc
bee
...
```

XTB

The Linux version can be obtained from https://www.chemie.uni-bonn.de/pctc/mulliken-center/software/xtb/xtb The online manual is available at https://xtb-docs.readthedocs.io/en/latest/contents.html Modify the following line in the `~/.JKCSusersetup.txt`:

```
PATH_XTB="[-XTB-folder-path-]"            #e.g.: "/user/kubeckaj/XTB6.4"
```

If you want to use the XTB program by yourself, either use the full path directory to the executables or put the following lines to your `~/.bashrc` file:

```
export PATH=[-XTB-folder-path-]/bin:$PATH #e.g.: "/user/kubeckaj/XTB6.4/bin"
export XTBHOME=[-XTB-folder-path-]        #e.g.: "/user/kubeckaj/XTB6.4"
```

then you should be able to run XTB, e.g.:

```
#source ~/.bashrc
xtb file.xyz --opt vtight
```

Gaussian

I hope that you know how to call Gaussian jobs. If not ask a blessed person around you how to do it. Usually you load Gaussian from a module, e.g.:

```
module load gaussian
```

then you can figure out where are located Gaussian executables, e.g.:

```
$USER: > which g16
/appl/soft/chem/gaussian/G16RevC.01_new/g16/g16
```

based on that modify the following lines in the `~/.JKCSusersetup.txt`:

```
PATH_G16="/appl/soft/chem/gaussian/G16RevC.01/"
MODULE_G16="module load gaussian/G16RevC.01"    #"module load Gaussian"
```

If you want to run Gaussian by yourself, use some predefined scripts (something like `subg16` etc.)

How to setup Jupyter

Activate JKCS-python environment for Jupyter

```
(.venv) $ pip install --user ipykernel
(.venv) $ python -m ipykernel install --user --name=jkcs
```

1.2 Citations

Please, if you use any code, try to check if you have cited all literature.

1.2.1 JKCS

When you use any JKCS command, please, cite the following paper that basically introduced JKCS to the world:

kubecka19

BibTeX

Kubečka, J.; Besel, V.; Kurtén, T.; Myllys, N.; Vehkamäki, H. Configurational Sampling of Noncovalent (Atmospheric) Molecular Clusters: Sulfuric Acid and Guanidine. *J. Phys. Chem. A* **2019**, *123*, 6022–6033. https://doi.org/10.1021/acs.jpca.9b03853

```
@article{kubecka19,
  author = {J. Kube{\v c}ka and V. Besel and T. Kurt{\'e}n and N. Myllys and H. Vehkam{\
↪"a}ki},
  title = {Configurational Sampling of Noncovalent (Atmospheric) Molecular Clusters:␣
↪Sulfuric Acid and Guanidine},
  journal = {J. Phys. Chem. A},
  year = {2019},
  volume = {123},
  pages = {6022--6033},
  doi = {https://doi.org/10.1021/acs.jpca.9b03853},
}
```

1.2.2 JKQC

JKQC come together with our first machine learning methods, hence please cite:

kubecka22

BibTeX

Kubečka, J.; Christiansen, A. S.; Rasmussen, F. R.; Elm, J. Quantum Machine Learning Approach for Studying Atmospheric Cluster Formation. *Environ. Sci. Technol. Lett.* **2022**, *9(3)*, 239–244.

```
@article{kubecka22,
  author = {J. Kube{\v c}ka and A. S. Christensen and F. R. Rasmussen and J. Elm},
  title = {Quantum Machine Learning Approach for Studying Atmospheric Cluster Formation},
  journal = {Environ. Sci. Technol. Lett.},
  year = {2022},
  volume = {9},
  pages = {239--244},
  number = {3},
  doi = {https://doi.org/10.1021/acs.estlett.1c00997},
}
```

1.2.3 JKML

When you use JKML, please cite:

kubecka22

BibTeX

Kubečka, J.; Christiansen, A. S.; Rasmussen, F. R.; Elm, J. Quantum Machine Learning Approach for Studying Atmospheric Cluster Formation. *Environ. Sci. Technol. Lett.* **2022**, *9(3)*, 239–244.

```
@article{kubecka22,
  author = {J. Kube{\v c}ka and A. S. Christensen and F. R. Rasmussen and J. Elm},
  title = {Quantum Machine Learning Approach for Studying Atmospheric Cluster Formation},
  journal = {Environ. Sci. Technol. Lett.},
  year = {2022},
  volume = {9},
  pages = {239--244},
  number = {3},
  doi = {https://doi.org/10.1021/acs.estlett.1c00997},
}
```

QML

If you use kernel ridge regression (KRR) with the FCHL19 molecular representation, cite also the following:

qml,christiansen20

BibTeX

Christensen, A. S.; Faber, F. A.; Huang, B.; Bratholm, L. A.; Tkatchenko, A.; Muller, K. R.; von Lilienfeld, O. A. QML: A Python Toolkit for Quantum Machine Learning. **2017**; https://github.com/qmlcode/qml (*accessed February 7, 2023*).

Christensen, A. S.; Bratholm, L. A.; Faber, F. A.; von Lilienfeld, O. A. FCHL Revisited: Faster and More Accurate Quantum Machine Learning. *J. Chem. Phys.* **2020**, *152*, 044107.

```
@misc{qml,
  author = {A. S. Christensen and F. A. Faber and B. Huang and L. A. Bratholm and A.␣
↪Tkatchenko and K. R. Muller and O. A. von Lilienfeld},
  title = {{QML}: {A} {P}ython Toolkit for Quantum Machine Learning},
  year = {2017},
  note = {\url{https://github.com/qmlcode/qml} (accessed February 7, 2023)}
}
@article{christensen20,
  author = {A. S. Christensen and L. A. Bratholm and F. A. Faber and O. A. {von␣
↪Lilienfeld}},
  title = {{FCHL} Revisited: {F}aster and More Accurate Quantum Machine Learning},
  journal = {J. Chem. Phys.},
  year = {2020},
  volume = {152},
  pages = {044107},
  doi = {https://doi.org/10.1063/1.5126701},
}
```

NN

TBC

1.2.4 JKacdc

You should cite the Pearl code itself and also the repository of T. Olenius. These codes were modified and inspired the JKacdc code:

mcgrath12,acdc

BibTeX

McGrath, M. J.; Olenius, T.; Ortega, I. K.; Loukonen, V.; Paasonen, P.; Kurtén, T.; Kulmala, M.; Vehkamäki, H. Atmospheric Cluster Dynamics Code: a flexible method for solution of the birth-death equations. *Atmos. Chem. Phys.* **2012**, *12(5)*, 2345–2355.

Olenius T. ACDC: Atmospheric Cluster Dynamics Code. **2023**; https://github.com/tolenius/ACDC (*accessed February 7, 2023*).

```
@article{mcgrath12,
  author = {McGrath, M. J. and Olenius, T. and Ortega, I. K. and Loukonen, V. and␣
↪Paasonen, P. and Kurt{\'e}n, T. and Kulmala, M. and Vehkam{\"a}ki, H.},
  title = {Atmospheric Cluster Dynamics Code: a flexible method for solution of the␣
↪birth-death equations},
  journal = {Atmos. Chem. Phys.},
  volume = {12},
  year = {2012},
  number = {5},
  pages = {2345--2355},
  doi = {https://doi.org/10.5194/acp-12-2345-2012}
}
@misc{acdc,
  author = {T. Olenius},
  title = {ACDC: Atmospheric Cluster Dynamics Code},
  year = {2023},
  note = {\url{https://github.com/tolenius/ACDC} (accessed February 7, 2023)}
}
```

1.3 Cluster submission

When working on a (super)computer cluster, all default parameters for submitting jobs are taken from `parameters.txt` in each subfolder formed from `input.txt` by JKCS1_prepare. It is assumed that the computer system uses SBATCH to submit jobs. Hence, additional parameters for the SBATCH command could be eventually modified in the `~/.JKCSusersetup.txt` under the program_SBATCH function. When calling a JKCS script, it is also possible to overwrite the submission parameters by the arguments shown on this page.

Remember that only JKCS2_explore, JKCS3_run, JKCS4_collect, and JKML are scripts that would submit jobs to cluster as these scripts employ 3rd party programs that might demand considerable computational resources. Using the submission parameters for JKCS0_copy, JKCS1_prepare, and JKCS5_filter is not possible (they run locally).

1.3.1 Run locally

If you want to also run JKCS2-4 on your local computer and not submit any jobs, you can use:

-loc Perform all tasks on computer you are now logged in.

```
JKCS3_run -p XTB -m "--opt --gfn 1" -loc
JKCS4_collect XTB -loc
JKCS4_collect DFT_freq -loc
```

Note: It is completely fine to run some tests cluster login nodes BUT keep the tests very short (at maximum 1 or very few minutes), and you should also stay low with memory requirements.

1.3.2 Submission arguments

To submit jobs to queue the sbatch command of the following format is automatically used by JKCS (you can modify it in `~/.JKCSusersetup.txt` under the program_SBATCH function):

```
sbatch -J "Job_name" -p "Partition_name" --time "Walltime" -N 1 --mem-per-cpu "Memory"␣
↪script.sh
```

You can overwrite the default arguments by using these commands:

-cpu <integer> number of CPUs used for one calculation. If you specify only the number of CPUs, the rest of submission arguments (e.g., walltime, partition name) are still taken from `parameters.txt`

```
JKCS3_run -p G16 -rf XTB -nf DFT_opt -m "# wb97xd 6-31++g** opt" -cpu 8
```

-par, -partition <string> partition (queue) name (e.g., test, short, longrun, or hugemem). You should see all partitions by typing the command: `sinfo`

-time <time_format> requested walltime (e.g., 72:00:00, 1-00:00:00 or 10:00). If you need to submit a simple/fast test on the test partition, run:

```
JKCS2_explore -pop 2 -gen 2 -lm 2 -par test -time 10:00
JKCS4_collect ABC -par test -time 10:00
```

-mem, -memory <memory_string_format> size of memory allocated per CPU [e.g., 4000mb or 32gb]

-tpj <integer> number of calculations combined into 1 job (=tasks per job/1 submitted job). For instance, 100 Gaussian optimizations can be submitted as 20 jobs where each job will perform 5 calculations using 8 CPUs:

```
JKCS3_run -rf XTB -nf DFT_opt -p G16 -m "# wb97xd 6-31++g** opt" -tpj 5 -cpu 8
```

If you have many conformer combinations, you can reduce the configurational search for each of them and run them in series. If you have 300 combinations, you can submit only 30 jobs using (+ you can do the same with the subsequent XTB optimization):

```
JKCS2_explore -pop 50 -gen 50 -lm "6000/NoC" -tpj 10
JKCS3_run -tpj 10
```

-maxtasks <integer> max. number of tasks to be submitted (per cluster subfolder). I am worried that people sometimes do not adequately calculate how many jobs they could submit with one command. Therefore, I did restrict your submission to max 100 jobs. You can easily raise this threshold by this argument.

-N, -nodes <integer> number of nodes. It is by default 1. However, the functionality of this argument was not properly tested yet. See the greasy-multitask section on this page for more details.

Note: The order of the arguments is not important.

1.3.3 Greasy (multinodal) multitask single job

There is an option to submit only single job that contains several tasks that will run parallely. It is greasy (=dirty) way of using multinodal clusters where submission of single jobs is not allowed or maximal number of submitted jobs is limited. Since the submitted job has to wait for all tasks to be finished (also the slowest one), it leads to waste of computational resources where CPUs are not used. Hence, it is called greasy.

-greasy this will activate greasy mode

-con, -cores_on_node <integer> number of cores on node where you submit jobs. Default = 40; however, you should set the correct number of cores. Use the `sinfo -N -l` command to see how many cores are on which partition. Please, be aware that if a partition has 256 cores but 2 nodes then you should specify -con 128.

-N, -nodes <integer> number of nodes. If the number of tasks requires more nodes, you should set it. If you set more than necessary, then the maximal necessary number of nodes will be used (so feel free to e.g. set -nodes 20 if you do not want to worry about that). Remember that there is also some maximal number of nodes per cluster partition.

OK, let us see some examples. What about 64 Gaussian calculations each using 8 CPUs while submitting to the "medium" partition with 128 cores per node:

```
JKCS3_run -p G16 -rf XTB -n DFT -m "# wb97xd 6-31++g** opt" -con 128 -cpu 8 -nodes 2 -
↪greasy -par medium -time 12:00:00
```

I can also submit more jobs per each task. For instance, one greasy-worker will do 2 calculation jobs. I will run 64 calculations which will still fit to 2 nodes (2nodes * 128cores * 2jpt / 8cpu = 64jobs). I will increase the walltime though

```
JKCS3_run -p G16 -rf XTB -n DFT -m "# wb97xd 6-31++g** opt" -con 128 -cpu 8 -nodes 2 -
↪greasy -par medium -time 24:00:00 -jpt 2
```

Note: I did not test how durable is the argument -jpt. However, at least 3 jobs per task went through easily. 100 did not. Let me know if you find the limit.

Note: Yet, the greasy option works only for JKCS3_run.

1.4 About & Examples

JKCS presents a systematic approach for configurational sampling (CS) of molecular clusters. To perform CS on a chosen cluster, JKCS starts from the individual monomers that comprise the cluster. Of these monomers, all possible conformers are taken into account. This is done because the optimal structure of an individual monomer is not necessarily the optimal structure for this monomer inside the cluster. Also, relevant deprotonated and protonated conjugates of monomers are considered, as proton transfer can occur between components in the cluster. JKCS will find all possible combinations of these monomer conformers that fulfil the chosen cluster composition and total charge. Of each of these combinations, a large number of guess structures are created by adding the conformers together at random orientations. The guess structures of all possible combinations are subsequently optimized through quantum chemistry calculations at progressively higher levels of theory. After each optimization, unlikely structures are removed. The filtering out of these structures allows the next calculation at a higher level of theory to run at a reasonable computational cost. At the end of the configurational sampling, we end up with just a few appropriate structures that are optimized at a high level of theory.

It is important to note that there is no guarantee that the global minimum of the cluster is actually found through configurational sampling. It becomes more probable if we include all monomer conformers, use a large number of guess structures and optimize more structures at higher levels of theory, but absolute certainty that the global minimum is found will never be achieved.

JKCS PROVIDES:

- a collection of scripts which can be used for CS with a large set of structures and files
- communication with 3rd-party computational programs such as Gaussian, ORCA, ABCluster, and XTB (and for advanced users: CREST, IMoS)
- automated handling of jobs and submissions to (super)computer clusters
- it is coupled with JKQC and the pickled file/structures databases can be easily forwarded to JKML for machine-learning purposes

1.4.1 Basic example

Let us do a simple test on our local/login computer. This test can also help you to verify that you did setup JKCS/JKQC correctly. Go to a working directory and start your first CS test, e.g.:

```
cd <working-dir>
JKCS0_copy sa w
cat input.txt
JKCS1_prepare
cd SYS_1sa_1w
JKCS2_explore -pop 5 -gen 5 -lm 3 -loc
JKCS4_collect ABC -loc
cat resultsABC.dat
molden movieABC.xyz
JKCS3_run -p XTB -of ABC -loc
JKCS4_collect XTB -oc -loc
JKQC collectionXTB.pkl -movie
mv movie.xyz movieXTB.xyz
molden movieXTB.xyz
JKQC collectionXTB.pkl -sort el -select 1 -ePKL -el > resultsXTB_FILTERED.dat
cat resultsXTB_FILTERED.dat
JKCS3_run -rf resultsXTB_FILTERED.dat -p XTB -nf XTB_freq -m "--ohess -gfn 1" -loc
```

(continues on next page)

(continued from previous page)

```
JKCS4_collect XTB_freq -oc -loc
JKQC collectionXTB_freq.pkl -b -el -g
```

This procedure will prepare input file for CS of sulfuric acid and water dimer. Performs short (and bad) configurational space exploration using ABCluster. Collects the results and visulazize them with molden. Then performs XTB optimization, and again, collects the results and visulazize them with molden. Afterwards, only one lowest XTB energy structure is taken for vibrational frequency analysis again at XTB level. The final file basename, energy, and Gibbs free energy is printed out.

For more details see the other sections of JKCS manual. For help with individual programs continue reading on this page.

1.4.2 Advanced example

Here is more advanced example:

```
JKCS2_explore -pop 3000 -gen 200 -lm 2000/NoC -cpu 1 -time 3-00:00:00 -exploded -par q64,
↪q48,q40,q28,q24,q20
JKCS3_run -of ABC -nf XTB -m "--gfn 1 --opt" -cpu 1 -time 1-00:00:00 -par q64,q48,q40,
↪q28,q24,q20
JKCS4_collect XTB -oc -time 1-00:00:00
JKQC collectionXTB.pkl -unique rg,el,dip -out filteredXTB0.pkl
JKQC filteredXTB0.pkl -reacted -sort el -select 1000 -noex -out filteredXTB.pkl -ePKL >␣
↪TO_DO_DFT.dat
JKCS3_run -p G16 -rf TO_DO_DFT.dat -nf DFT_SP -m "# wb97xd 6-31++g**" -time 2:00:00 -cpu␣
↪8 -maxtasks 10000 -arraymax 25
JKCS4_collect DFT_SP -oc -time 1-00:00:00
JKQC collectionDFT_SP.pkl -sort el -select 100 -noex -ePKL > runDFT.dat
JKCS3_run -p G16 -rf runDFT.dat -nf DFT_opt -m "# wb97xd 6-31++g** opt" -time 3-00:00:00␣
↪-cpu 8 -maxtasks 10000 -arraymax 25
JKCS4_collect DFT_opt -oc -time 1-00:00:00
JKQC collectionDFT_opt.pkl -noex -ePKL > runDFTO.dat
JKCS3_run -p G16 -rf runDFTO.dat -nf DFT_freq -m "# wb97xd 6-31++g** freq" -time 1-
↪00:00:00 -cpu 8 -maxtasks 10000 -arraymax 25
JKCS4_collect DFT_freq -time 1-00:00:00 -oc
JKQC collectionDFT_freq.pkl -pass lf 0 -sort g -fc 100 -v 0.996 -select 5 -noex -ePKL >␣
↪runDLPNO.dat
JKCS3_run -p ORCA -rf runDLPNO.dat -nf DLPNO -m "! aug-cc-pVTZ aug-cc-pVTZ/C cc-pVTZ-F12-
↪CABS DLPNO-CCSD(T) TightSCF" -time 2-00:00:00 -cpu 8 -mem 20gb
JKCS4_collect DLPNO -time 1-00:00:00 -orca -oc
```

The very same thing with comments:

```
#configurational sampling that produces over all possible
# monomer cominations only 2000 minima. The files are in
# the end pickled and remove to save memory. Structures
# with large radius are completely removed.
JKCS2_explore -pop 3000 -gen 200 -lm 2000/NoC -cpu 1 -time 3-00:00:00 -exploded -par q64,
↪q48,q40,q28,q24,q20

#Performs XTB optimization with the same folder architecture
# as in the ABC folder, i.e. one job is submitted for each
```

(continues on next page)

(continued from previous page)

```
# conformer combination and optimizing 2000/NoC structures
JKCS3_run -of ABC -nf XTB -m "--gfn 1 --opt" -cpu 1 -time 1-00:00:00 -par q64,q48,q40,
↪q28,q24,q20

#This will Only Collect the pickle file: collectionXTB.pkl
JKCS4_collect XTB -oc -time 1-00:00:00

#Here we get rid of those structrues that converged to the same
# minima, i.e. having the same configuration. For that we compare
# similarity between electronic energy, gyration radius and dipole
# at the same time. You could use e.g. -arbalign <float> instead
JKQC collectionXTB.pkl -unique rg,el,dip -out filteredXTB0.pkl

#well this one will work for you only if your JKsend is set properly.
# Nevertheless, the JKQC command will remove structures that reacted
# during XTB optimization (it does compare the non-hydrogen skelets),
# then the 1000 electronic-energy lowest structures are saved for
# next calculation. (-noex = do not print example)
JKQC filteredXTB0.pkl -reacted -sort el -select 1000 -noex -out filteredXTB.pkl -ePKL >␣
↪TO_DO_DFT.dat

#Run Gaussian SP calculation from the Result File TO_DO_DFT.dat. It
# does submit 8-cpu job for each line in that file. The jobs is always
# an array, but now only 25 calculations will run at a time. The
# maxtasks value says that we guarantee that no more than 1000 lines
# is saved in the results file (prevents submitting to many jobs)
JKCS3_run -p G16 -rf TO_DO_DFT.dat -nf DFT_SP -m "# wb97xd 6-31++g**" -time 2:00:00 -cpu␣
↪8 -maxtasks 1000 -arraymax 25

#Again we collect the data
JKCS4_collect DFT_SP -oc -time 1-00:00:00

#Now we select only 100 lowest energies
JKQC collectionDFT_SP.pkl -sort el -select 100 -noex -ePKL > runDFT.dat

#and submit them again for Gaussian calculation
JKCS3_run -p G16 -rf runDFT.dat -nf DFT_opt -m "# wb97xd 6-31++g** opt" -time 3-00:00:00␣
↪-cpu 8 -maxtasks 100 -arraymax 25

#Again we collect the data
JKCS4_collect DFT_opt -oc -time 1-00:00:00

#We just prepare next calculation
JKQC collectionDFT_opt.pkl -noex -ePKL > runDFTO.dat

#Now we calculate vibrational frequencies. Theoretically, you could
# combine it with the previous step but this is very practical as
# some of the optimization do not finish do to presence of shallow minima
JKCS3_run -p G16 -rf runDFTO.dat -nf DFT_freq -m "# wb97xd 6-31++g** freq" -time 1-
↪00:00:00 -cpu 8 -maxtasks 10000 -arraymax 25

#Again we collect the data
```

(continues on next page)

(continued from previous page)

```
JKCS4_collect DFT_freq -time 1-00:00:00 -oc

#We filter out all data for which the lowest vibrational frequency
# is less than 0 (i.e. imaginary). Then we select 5 lowest Gibbs free
# energy structure after we correct for the low vibrational frequencies
# with threshold of 100 cm-1
JKQC collectionDFT_freq.pkl -pass lf 0 -sort g -fc 100 -v 0.996 -select 5 -noex -ePKL >␣
↪runDLPNO.dat

#Now we submit ORCA calculation. We add an extra memory
JKCS3_run -p ORCA -rf runDLPNO.dat -nf DLPNO -m "! aug-cc-pVTZ aug-cc-pVTZ/C cc-pVTZ-F12-
↪CABS DLPNO-CCSD(T) TightSCF" -time 2-00:00:00 -cpu 8 -mem 20gb

#Again we collect the data
JKCS4_collect DLPNO -time 1-00:00:00 -orca -oc
```

1.4.3 Large clusters

This is what we (H. Wu and G. Hasan) typically use for large clusters (freshly-nucleated particles) where only one conformer combination (ionic) is used.

```
#Detail protocol is available here: https://doi.org/10.1021/acsomega.3c06794
#ABCluster
JKCS2_explore -pop 1280 -gen 320 -repeat 10 -sc 4 -lm 1000 -expl -cpu 8
JKCS4_collect ABC -oc

#XTB calculation
JKCS3_run -p XTB -rf collectionABC.pkl -nf XTBopt -m "--opt --gfn 1" -maxtasks 1000 -cpu␣
↪2 -tpj 10 -mf 1000 -time 10:00:00 -arraymax 400
JKCS4_collect XTBopt -oc
JKQC collectionXTBopt.pkl -uniq rg,el,dip -out collectionXTBopt_filtered.pkl #-reacted␣
↪(optional)

#DFT single point calculation
JKCS3_run -p ORCA -rf collectionXTBopt_filtered.pkl -nf B97-3Csp -m "! b97-3c TightSCF" -
↪time 1-00:00:00 -maxtasks 1000 -cpu 2 -tpj 10 -mf 1000 -mem 8GB -arraymax 400
JKCS4_collect B97-3Csp -orca -oc
JKQC collectionB97-3Csp.pkl -sort el -select 1000 -out collectionB97-3Csp_filtered.pkl

#DFT Partial Geometry Optimization
JKCS3_run -p ORCA -rf collectionB97-3Csp_filtered.pkl -nf B97-3Cpartopt -m "! b97-3c opt␣
↪TightSCF" -time 12:00:00 -maxtasks 1000 -cpu 2 -mem 8GB -arraymax 150 -add maxiter 40
JKCS4_collect B97-3Cpartopt -orca -oc
JKQC collectionB97-3Cpartopt.pkl -unique rg,el,dip -sort el -select 100 -out␣
↪collectionB97-3Cpartopt_filtered.pkl

#DFT Full Geometry Optimization and vib. frequency calculation
JKCS3_run -p ORCA -rf collectionB97-3Cpartopt_filtered.pkl -nf B97-3Coptfreq -m "! b97-
↪3c opt TightSCF Anfreq" -time 12:00:00 -maxtasks 100 -cpu 8 -mem 12GB
JKCS4_collect B97-3Coptfreq -orca -oc -loc
```

1.4.4 Tips & Tricks

GLOBAL MINIMUM

There is no guarantee that the global minimum of the cluster is actually found through configurational sampling. It becomes more probable if we include all monomer conformers, use a large number of guess structures and optimize more structures at higher levels of theory, but absolute certainty that the global minimum is found will never be achieved.

DETAIL OF OUTPUT

The amount of printed output can be adjusted by using command '-print NUM':

- -print 0 .... basically just error messages
- -print 1 .... [DEFAULT] traditional output
- -print 2 .... enlarged output, all algorithm steps are commented
- -print 3 .... very very detailed output

PREVIOUS COMMANDS

The JKCS/JKQC/JKML commands are saved into the 'output' file. You can examine them by:

```
grep COMMAND output
```

CALLING JKCS COMMANDS

Each JKCS command can be performed inside a specific subfolder 'SYS_{system}', or from your 'parent directory' where the subfolders are located. In this case the algorithm enters each directory and performs the command there.

If you wish to perform a command from your 'mother directory', but only for some specific subfolders, you can give the subfolder as an argument:

```
JKCS2_explore SYS_3SA SYS_4SA -gen 100
JKCS2_explore SYS_1SA_1-5AM -pop 2000 -gen 150
```

JKcheck

JKcheck can be used to check how many calculations have been finished.

ORDER OF ARGUMENTS

The order of arguments does not matter.

COLORS & SYMBOLS IN PRINTED OUTPUT

The colors and symbols in the printed output of JKCS commands can be turned off in 2 ways:

- Change Qsymbol or Qcolours in ~/.JKCSusersetup.txt to "no"
- COLORING TEXT: use -nocolors argument to have text without colors
- KISSING SYMBOL: use -nosymbol to remove the symbol in the begging of output

'M' & 'NoC' SYMBOLS

These symbols can be used with many JKCS commands to make values dependent on the 'number of Molecules' or 'Number of Combinations'. When a lot of conformers are taken into account for a certain monomer, 'NoC' can become very large. Therefore, always be mindful of the result that using 'M' and 'NoC' might have for the exploration of all studied systems. Example:

```
JKCS2_explore -pop 1000*M -gen 100 -lm 4000/NoC
```

BOSS AND MANAGER

Manager can run multiple JKCS commands and wait until these are finished. The commands to be executed are read from a txt file. Each command should be put on a separate line. Boss can handle several managers. Since these are already complicated command with self-submission, I will not provide manual for those and only if you consider yourself an experienced user, contact J. Kubečka and he will provide the manual.

1.5 Input file

Configurational sampling with JKCS starts by making `input.txt`, which can be generated by JKCS0_copy (see next chapter). This file contains the most import definitions for sampling the desired clusters in the current directory. Example:

```
#########################################################
## SUPERCOMPUTER PARAMETERS                            ##
## Number of Combinations - NoC                        ##
#########################################################
## MAXTASKS CPU NODES REQ.TIME  PARTITION MEMPERCPU ##
=========================================================
ABC   NoC     1    1   72:00:00   small      4000mb
XTB   NoC     1    1   72:00:00   small      4000mb
G16   100     8    1   72:00:00   small      4000mb
ORCA  100     8    1   72:00:00   small      4000mb
CC    100     8    1   330:00:00  longrun    4000mb
-loc  1       1    1   -          -          4000mb
=========================================================

#####################################
## SYSTEM CHARGE AND MULTIPLICITY  ##
#####################################
TotalCharge        0
TotalMultiplicity  1

#####################################################################
## COMPOSITION:                                                    ##
## e.g.: 1_1_2 1_2_1 1_3                                           ##
## e.g.: 1_3-6             = 1_3 1_4 1_5 1_6                       ##
## e.g.: (1,3,5)_1         = 1_1 3_5 5_1                           ##
## e.g.: (2,4)_1-3_1       = 2_1_1 4_1_1 2_2_1 4_2_1 2_3_1 4_3_1 ##
## e.g.: 1_1_F2            = 1_1_0-2 #protons# to fulfill charge ##
#####################################################################
Composition     1_1  2_2

#####################################
## STRUCTURES OF BUILDING MONOMERS: ##
#####################################
# name | q | path
  sa     0   /projappl/hvehkama/kubeckaj/Apps/JKCS2.1/JKCSx/../TOOLS/STRUCTURES/JACOB/sa.
↪xyz
  sa     0   /projappl/hvehkama/kubeckaj/Apps/JKCS2.1/JKCSx/../TOOLS/STRUCTURES/JACOB/
↪h2so4_cis.xyz
  sa    -1   /projappl/hvehkama/kubeckaj/Apps/JKCS2.1/JKCSx/../TOOLS/STRUCTURES/ABC/hso4.
↪xyz
```

(continues on next page)

(continued from previous page)

```
  sa     -2   /projappl/hvehkama/kubeckaj/Apps/JKCS2.1/JKCSx/../TOOLS/STRUCTURES/ABC/so4.
↪xyz
  am      0   /projappl/hvehkama/kubeckaj/Apps/JKCS2.1/JKCSx/../TOOLS/STRUCTURES/ABC/nh3.
↪xyz
  am      1   /projappl/hvehkama/kubeckaj/Apps/JKCS2.1/JKCSx/../TOOLS/STRUCTURES/ABC/nh4.
↪xyz
```

Note: When configurational sampling needs to be done on a cluster that contains monomers not included within JKCS, then `input.txt` can be created from scratch. It is of course also possible to let JKCS0_copy create an input.txt file for a random system and subsequently change the necessary parameters to fit the desired system.

Important: When creating `input.txt` file through JKCS0_copy, it is likely that some parameters, like the composition, still need to be altered.

`input.txt` is divided into four parts:

Contents

1.5.1 SUPERCOMPUTER PARAMETERS

This section defines parameters for submitting JKCS to (super)computer cluster. JKCS communicates with 3rd-party programs that are further abbreviated as: ABC = ABCluster, XTB, G16 = Gaussian16, and ORCA. For each program, the following parameters need to be specified:

MAXTASKS maximum number of tasks that can run in parallel. Beginners should not adjust this.

CPU number of CPUs.

NODES number of nodes. Beginners should leave 1.

REQ.TIME the requested/required walltime (expected length of calculations)

PARTITION the name of cluster partition (e.g. small, large, hugemem)

MEMPERCPU the amount of memory per CPU

Note: For almost all of them, you can use variables "NoC" and "M" to define the parameter as a function of "Number Of (monomer) Combinations" or "(total) number of Molecules".

Hint: When calling one of the next JKCS scripts (e.g., JKCS3_run), the submission parameters for running that script can also be specified as additional arguments to the script. We could, for instance, call

```
JKCS3_run XTB -par small -mem 8gb
```

to change the partition name and memory per CPU from what is written in the `input.txt` file (or actually in `parameters.txt` formed later for each cluster type).

These commands are further explained in the 'Cluster submission' section of this manual.

Hint: The default table for supercomputer parameters can be changed in ~/.JKCSusersetup.txt.

1.5.2 SYSTEM CHARGE AND MULTIPLICITY

This section consists of two parameters that need to be set: total charge and total multiplicity of the cluster(s).

Note: The multiplicity is equal to the number of unpaired electrons plus one (TotalMultiplicity = 2S+1).

Hint: If you want to study clusters of different charges, use different folders.

1.5.3 COMPOSITION

The composition defines the number of each monomer in the desired cluster. For each cluster, the composition is written as n[1]_n[2]_n[3]…_n[M], where n[i] is the number of monomers of type i in the desired cluster. The order in which the monomers appear in this format should be the same as the order in which the monomers are listed in the "structure of building monomers" part of `input.txt` (see below). If there are listed molecules "sa" and "am", the composition "1_2" equals to cluster (sa)1(am)2.

When configurational sampling of multiple clusters with different compositions needs to be done, each composition can be written on one line with a space between two separate compositions. For multiple clusters, some symbols can also be used to quickly define the clusters. Writing "1-3_(4,5)" would for instance be equivalent to "1_4 2_4 3_4 1_5 2_5 3_5".

1.5.4 STRUCTURE OF BUILDING MONOMERS

This section should list the name, path and charge of all available conformers and conjugate acids/bases for all the monomers in the desired cluster. All conformers and conjugate species of the same molecule should have the same name (e.g. "sa" for cis- and trans-sulfuric acid as well as for the bisulfate and sulfate ion). The order of the list should be the same as the order in which the composition was written.

EXAMPLE: Consider as an example that we would like to perform configurational sampling on a negative cluster containing two sulfuric acid molecules and one ammonia molecule. For this, we would change the total charge of the cluster to -1. The multiplicity would be left at 1. The composition would be given as "2_1". Lastly, we fill in the "structure of building monomers" part. As we have filled in "2_1" in the composition, we first list all sulfuric acid conformers and conjugate bases and then we list the ammonia structure and conjugate acid. There are two sulfuric acid conformers to take into account: cis- and trans-sulfuric acid. Ammonia has only one conformation. We take multiple conformers of monomers into account because it is not certain that the lowest energy monomer conformer is also the preferred conformation inside the cluster. Sulfuric acid has two conjugate bases: bisulfate and sulfate. Ammonia has one conjugate acid: ammonium. We consider the conjugate acids and bases because internal acid-base reactions could

occur between monomers in the cluster. The paths to all the different structures related to one monomer should be listed together.

1.6 JKCS0_copy

JKCS0_copy automatically creates `input.txt` based on the monomers given as arguments. JKCS contains optimized structures for the different conformers and conjugated acids/bases of some of the most prominent monomers in atmospheric clusters (such as sa = sulfuric acid, am = ammonia, na = nitric acid or w = water). Calling JKCS0_copy creates `input.txt` containing the paths to the optimized structures of the conformers and conjugated acids/bases of all the requested monomers in the "structure of building monomers" part of the file.

It is once again important to note that even when creating `input.txt` through JKCS0_copy, it is likely that some parameters, like the composition, still need to be altered. JKCS0_copy will write two example compositions for the chosen monomers. These of course do not necessarily have to coincide with the desired cluster composition. Therefore, always check `input.txt` before proceeding.

Hint: For help use:

```
JKCS0_copy -help
```

Here are few examples:

```
JKCS0_copy sa am       #creates input.txt with links to sulfuric acid and ammonia
JKCS0_copy msa am dma  #creates input.txt with links to methansulfonic acid, ammonia,␣
↪and dimethylamine
JKCS0_copy -all        #created input.txt with links to all available structures
```

Note: If your structure is not available in JKCS database, you have to create the ABCluster input by yourself. See ABCluster manual for more details.

```
w ............ water (H2O,OH-,H+)          cd .......... carbon dioxide (CO2)
aq ........... water (H2O)                 m ........... methane (CH4)
                                           ar .......... argon (Ar)
nta .......... nitric acid (HNO3)          ne .......... neon (Ne)
sa ........... sulfuric acid (H2SO4)       he ........... helium
msa .......... methanesulfonic acid
                                           h ............ proton (H+)
am ........... ammonia (NH3)               na ........... sodium (Na+)
gd ........... guanidine (CN3H5)           cl ........... chloride (Cl-)
dma .......... dimethylamine (C2H7N)
tma .......... trimethylamine (C3H9N)      ica ......... iodic acid (HIO3)
urea ......... urea (CH4N2O)               isa ......... iodous acid (HIO2)
buoh ......... butanol (C2H5OH)            ip .......... iodine pentoxide (I2O5)
```

EXAMPLE: Consider an example of a negatively charged cluster containing 1 ammonia and 2 sulfuric acid molecules. Type "JKCS0_copy sa am" to creat `input.txt`. `input.txt` will already contain the information of cis- and trans-sulfuric acid, bisulfate, sulfate, ammonia and ammonium in the "structure of building monomers" section. The total charge and composition will however not be the same as our desired cluster. We will thus have to change the total charge to -1 and the composition to "2_1".

If we would have called "JKCS0_copy A SA", ammonia and ammonium would first be mentioned in the "structure of building monomers" section followed by the sulfuric acid monomers and conjugated bases. In this case, we should write the composition as "1_2" to obtain our desired cluster.

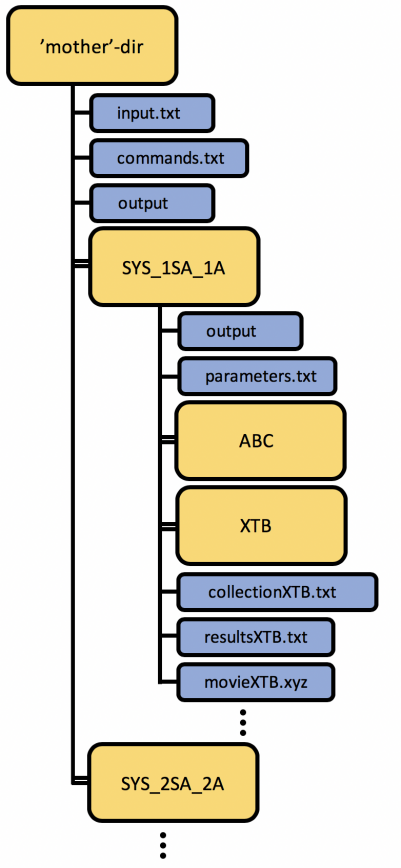
1.7 JKCS1_prepare

JKCS1_prepare uses the information from `input.txt` to create SYS_{system} subfolders in your working directory. Each SYS_{system} subfolder corresponds to a specific cluster composition. JKCS1_prepare also creates `parameters.txt` inside each of these subfolders. `parameters.txt` file is similar to `input.txt` and is used further for additional calculations in each subfolder. JKCS1_prepare has however added all possible combinations of conformers and conjugated acids/bases of monomers, which fulfil the composition and total charge as defined in `input.txt`, to the end of `parameters.txt`. User does not have to check those but these monomer combinations will be used in the configurational sampling. JKCS1_prepare does not require any arguments.

Hint: For help use:

```
JKCS1_prepare -help
```

The folder/subfolders tree should during configurational sampling look like this:



Hint: Each subsequent script (e.g. JKCS2_explore or JKCS3_run) can either be called from the working directory or from one of the SYS_{system} subfolders. When called from a subfolder, it is only applied to that specific subfolder.

When called from the working directory, the script is applied to all subfolders. However, you can also specify to which subfolder you want to apply scripts. See examples:

```
JKCS3_run -p XTB -m "--opt --gfn 1" SYS_2sa_*am
JKCS4_collect XTB SYS_2sa_1am
```

if you want to perform a specific command (or sequence of commands) in each subfolder, use:

```
JKfor head -n 3 resultsXTB.dat
JKfor "head -n 1 resultsDFT_freq.dat > ../all_results.dat"
JKfor 'head -n 50 resultsXTB.dat > resultsXTB_FILTERED.dat; JKCS3_run -p XTB -rf XTB -nf␣
↪XTB_freq -m "--hess --gfn 1"'
```

EXAMPLE: Calling JKCS1_prepare for a negatively charged cluster of 1 ammonia and 2 sulfuric acid molecules, `SYS_2am_1sa` folder would be created. This folder would contain `parameters.txt` where 5 possible cluster combinations are written:

- 0_1_0_0_2_0 (NH4+)1(HSO4-)2
- 0_1_0_1_0_1 (NH4+)1(cis-H2SO4)1(SO4-)1
- 1_0_0_1_1_0 (NH3)1(cis-H2SO4)1(HSO4)1
- 0_1_1_0_0_1 (NH4+)1(trans-H2SO4)1(SO4-)1
- 1_0_1_0_1_0 (NH3)1(trans-H2SO4)1(HSO4)1

You can easily verify that these clusters all have total charge 1 and that these are all the combinations that can be made for this specific composition and total charge.

1.7.1 Arguments

-s, -sample <integer> sample from all monomer combinations. In the case of many combinations, you can pick just random combinations (helpful when molecules with a lot of conformers are used). For instance:

```
JKCS1_prepare -s 100
JKCS1_prepare -s NoC/2
```

Note: You can use variables "NoC" and "M" to define the parameter as a function of "Number Of (monomer) Combinations" or "(total) number of Molecules".

However, I recommend lowering the simulation parameters and using all combinations. In the case of 200 conformer combinations, the following two options would lead to a similar computational cost, but the second one would lead to a more thorough configurational search:

```
# REDUCING CONFORMER COMBINATIONS:
JKCS1_prepare -s 100
JKCS2_explore -pop 100 -gen 200 -lm 500
JKCS3_run

# REDUCING SIMULATION LENGTH (200 NoC)
# -jpt 2 = each submitted job performs 2 simulations/calculations
JKCS1_prepare
```

(continues on next page)

(continued from previous page)

```
JKCS2_explore -pop 100/2 -gen 200/2 -lm 500/2 -jpt 2
JKCS3_run -jpt 2
```

-o, -overwrite overwrites `parameters.txt` in already existing subfolders. Not important for beginners.

1.8 JKCS2_explore

JKCS2_explore employs ABCluster (http://www.zhjun-sci.com/software-abcluster-download.php) to search for the global and local minima of clusters. ABCluster implements the Artificial Bee Colony (ABC) algorithm which was first proposed by Karaboga in 2008. For every cluster combination, the ABCluster program creates a specified number of guess structures (pop = population) by adding the monomers of that cluster combination together at random orientations with respect to each other. The structures are optimized and evaluated using predefined energy potentials. Further, each new structure is formed by using weighted information of several structures combined (+ it is again optimized). Therefore, the population evolves as the new structures replace the old ones. This evolution run for a specified number of steps (gen = generations). If the potential energy of a specific guess structure does not change for more than a specified number of steps (sc = scout bee), the structure is replaced by a new random guess structure. This ensures that the algorithm does not get stuck in any local minimum for too long. At the end of all the optimization/generation steps, you can decide how many lowest minima are saved printed out (lm = local minima).

Hint: For help use:

```
JKCS2_explore -help
```

You can run a test of a sort simulation (gen = 5) with a very small population (pop = 5) on local (or login) computer. Save a few lowest minima (lm 3) and examine the subfolders. Example code:

```
cd SYS_1sa_1am
JKCS2_explore -gen 5 -pop 5 -lm 3 -loc
cd ABC
ls
cd ABC_1_0_0_0_1_0
ls
vim calc.out
cd calc-LM
ls
molden 1sa1am-100010_1_1.xyz
```

You should not run simulations shorter than 100 generations for proper configurational sampling. Remember that configurational spaces increase with cluster size, and thus you should use greater parameters for larger clusters. Use also adequate size of population and save enough of local minima.

```
JKCS2_explore -gen 200 -pop 1000 -lm 3000
JKCS2_explore -gen 100 -pop 300*M -lm 3000/NoC
```

Note: You can use variables “NoC” and “M” to define the parameter as a function of “Number Of (monomer) Combinations” or “(total) number of Molecules”.

Hint: Some variables are by default defined as functions of the ‘number of Molecules’ (M) and the ‘Number of

Combinations' (NoC). This is particularly useful when working with several clusters. Smaller cluster do not need such extensive exploration as larger clusters. M and NoC allow for the scaling of the exploration according to the cluster size and complexity. You can use these symbols as well to define arguments.

It is important to note that when a lot of conformers are taken into account for a certain monomer, NoC can become very large. As a result, M/NoC becomes very small. Therefore, always be mindful of the result that using M and NoC might have for the exploration of all your studied systems. When dealing with large NoC, we believe that better performance of configurational sampling is reached when doing a small exploration on all combinations, rather than an exhaustive exploration of only a few selected combinations.

1.8.1 Arguments

-JKQC after the run, the structures are pickled into a single file (`database.pkl`, see JKQC for more details). This saves memory and significantly reduces number of files. The subsequent commands should then use the -JKQC command as well. Example:

```
JKCS2_explore -gen 200 -pop 1000 -lm 3000 -JKQC
JKCS3_run -JKQC
JKCS4_collect XTB
```

-pop, -i, -init <integer> population size (or also number of initial guesses). The size of the colony that evolves over generations. [default = 300*M]

-g, -gen <integer> number of (ABC) generations (loops). [default = 100]

-l, -lm <integer> (maximal) number of the lowest local minima to be saved. [default = 300*M/NoC]

-s, -sc <integer> lifetime = maximum generations, i.e. number of loops before replacing unchanged structure. The best is to keep this parameter as 2-5. [default = 4]

-box <float|integer> simulation box size. When you use small or large (compared to sulfuric acid) molecules, you should modify the box size otherwise the resultant clusters could contain evaporated molecules or the configuration exploration would not be thorough enough. [default = 7+M]

-repeat <integer> each simulation is repeated (in parallel) X-times

```
JKCS2_explore -gen 200 -pop 300 -lm 1000 -sc 3 -box 2+M
```

1.8.2 Coupling between ABC and XTB

This method is useful for running flexible molecules. Note that in *input.txt* you should point to pure xyz (not the abc-xyz). You can create example input file also with JKCS0_copy

```
JKCS0_copy -helpxyz
JKCS0_copy XYZna XYZbuoh
<modify input.txt>
JKCS1_prepare
```

Ok, time to run JKCS2_explore with the ABC_XTB coupling:

```
JKCS2_explore -helpxtb
JKCS2_explore -abcxtb -gen 2000 -repeat 2 -cpu 8
```

1.8.3 ABC_XTB arguments

-abcxtb switch to the ABC_XTB version

-g, -gen <integer> number of (ABC) generations (loops). [default = 100] (i.e. number XTB runs)

-box <float|integer> simulation box size from -X to X. When you use small or large (compared to sulfuric acid) molecules, you should modify the box size otherwise the resultant clusters could contain evaporated molecules or the configuration exploration would not be thorough enough. [default = 7+M]

-repeat <integer> each simulation is repeated (in parallel) X-times [def=1]

-gfn <integer> defines GFN1-xTB or GFN2-xTB [def=1]

1.9 JKCS3_run

JKCS3_run utilizes 3rd-party programs to optimize clusters and evaluate their properties (e.g., electronic energy or free energies). The script either uses the output structures in the folder from previous step (e.g., `XTB`, `ABC`, `DFT_opt`) or those listed in a file (e.g., `resultsXTB.dat`).

Hint: For help use:

```
JKCS3_run -help
```

For instance, if you want to run XTB calculations on structures from `ABC` folder, you run (of = old folder):

```
JKCS3_run -of ABC
JKCS3_run          #this would do the same
```

If you did collect (and filter) structures into a file (e.g., `resultsXTB.dat`), to use those run (rf = results file):

```
#JKCS4_collect XTB
JKCS3_run -p G16 -m "# wb97xd 6-31++g** opt=verytight" -nf DFT_opt -rf XTB
JKCS3_run -p G16 -m "# wb97xd 6-31++g** opt=verytight" -nf DFT_opt -rf resultsXTB.dat
↪#this would do the same
```

If you did extra-filter the data and `resultsXXX_FILTERED.dat` exists, it will be used instead of `resultsXXX.dat`:

```
#JKCS4_collect XTB
#JKCS5_filter -d 10  #selects only the 10 lowest kcal/mol
JKCS3_run -p G16 -m "# wb97xd 6-31++g** opt=verytight" -nf DFT_opt -rf XTB
JKCS3_run -p G16 -m "# wb97xd 6-31++g** opt=verytight" -nf DFT_opt -rf resultsXTB_
↪FILTERED.dat #this would do the same
```

Hint: You can check if your calculations are still running by using:

```
JKcheck
JKcheck XTB
```

Note: Charges and multiplicities are handled by JKCS based on `parameters.txt`

1.9.1 XTB examples

After ABC exploration, you can directly run XTB by:

```
JKCS3_run
JKCS3_run       -p XTB        -of ABC        -nf XTB      -m "-opt vtight" #this would␣
↪do the same
JKCS3_run -program XTB -oldfolder ABC -newfolder XTB -method "-opt vtight" #this would␣
↪do the same
```

Some other options:

```
JKCS3_run -p XTB -of ABC -nf XTB      -m "-opt vtight --gfn 1"
JKCS3_run -p XTB -rf XTB -nf XTB_freq -m "-ohess -temp 298.15"
```

Note: If you did pickle structures after ABC exploration, you should use -JKQC command also now:

```
#JKCS2_explore -gen 200 -pop 1000 -lm 3000 -JKQC
JKCS3_run -JKQC
```

1.9.2 Gaussian examples

JKCS is well working with Gaussian16 = G16. Although, it should work with other versions as well. The commands are similar to those in XTB. Examples:

```
JKCS3_run -p G16 -rf XTB     -nf DFT_sp   -m "# HF 6-31+g*"
JKCS3_run -p G16 -rf XTB     -nf DFT_opt  -m "# wb97xd 6-31++g** opt=verytight"
JKCS3_run -p G16 -rf DFT_opt -nf DFT_freq -m "# wb97xd 6-31++g** freq"
JKCS3_run -p G16 -rf XTB     -nf DFT      -m "# wb97xd GEN Pseudo=Read Opt Int=UltraFine␣
↪Freq MaxDisk=32GB" -bc I -mem 12GB -cpu 16
```

1.9.3 ORCA examples

This is not that well tested as people usually use their own scripts for ORCA, however, you should be able to run it as well. Examples:

```
JKCS3_run -p ORCA -rf XTB      -nf OPT   -m "! PBE0 def2-TZVP TIGHTSCF Opt D3BJ"
JKCS3_run -p ORCA -rf DFT_freq -nf DLPNO -m "! DLPNO-CCSD(T) aug-cc-pvtz aug-cc-pvtz/C␣
↪GRID4 nofinalgrid TightPNO TightSCF NOPOP NOPRINTMOS"
```

1.9.4 Arguments

-JKQC after the run, the structures are pickled into a single file (`database.pkl`, see JKQC for more details). This saves memory and significantly reduces number of files.

-nf, -newfolder <string> name of the new (calculation) folder. [default = "XTB"]

-of, -oldfolder <string> name of the old (calculation) folder. Do not combine with -rf. [default = "ABC"]

-rf, -resultsfile <string> name of the results file (e.g., XTB, resultsXTB.dat, resultsXTB_FILTERED.dat) containing list of structures for further calculation. When -rf "NAME" is used and `resultsNAME_FILTERED.dat` is available, it is used instead of `resultsNAME.dat`. Do not combine with -of.

-m, -method <string> method used by 3rd-party program [default for XTB = "-opt vtight"]

-bs, -add, -addbase <atom> insert basis set for heavy atoms to the end of file (only for Gaussian) and yet only for atoms like I or Br

1.10 JKCS4_collect

After JKCS2_explore or JKCS3_run submitted jobs are finished, JKCS4_collect script collects all data within the calculation folder.

JKCS4_collect utilises JKQC to go through all the output files within specified calculation folder.

```
JKCS4_collect XTB
JKCS4_collect DFT_freq
```

The main outcome of the JKCS4_collect is, e.g., `collectionXTB.pkl` file. However, some other readable files are produced, too: `collectionXTB.txt` (list of files, gyration radii *Rg*, and energies *E*), `resultsXTB.dat` (the same file but after filtering based on *[Rg,E]* with thresholds of *[0.01,0.001]*, and structures are sorted with respect to energy), and `movieXTB.xyz` (list of xyz concatenated within one file). For instance, the `collectionXTB.txt` could contain:

```
/path/SYS_2sa_2am/XTB_2_2_0_0_0_0_2_1_0/2sa2am-220000_1_0.xyz          2.675910073831769  ␣
↪-56.7011189
/path/SYS_2sa_2am/XTB_2_2_0_0_0_0_2_1_0/2sa2am-220000_1_100.xyz        2.4146549048885455 ␣
↪-56.6972184
/path/SYS_2sa_2am/XTB_2_2_0_0_0_0_2_1_0/2sa2am-220000_1_101.xyz        2.643341483522292  ␣
↪-56.6990367
...
```

Note: When collecting data with vibrational frequency calculations, the structures in, e.g., `resultsDFT_freq.dat` will be then sorted with respect to the 4th column = Gibbs free energy.

Hint: The `movieXTB.xyz` can be easily visualized with Molden:

```
molden movieXTB.xyz
```

If you find the `collectionXTB.txt`, `resultsXTB.dat`, and `movieXTB.xyz` files unnecessary, you can only collect (*-oc*) the pickle:

```
JKCS4_collect XTB -oc
```

Hint: I strongly recommend checking JKQC manual for structure post-processing and manipulation of, e.g., `collectionXTB.pkl` file.

The JKCS4_collect is a submission script. If you used JKCS2_explore or JKCS3_run with the *-of* argument, then X+1 jobs will be submitted, i.e. one for each subfolder and one for collecting all pickles into one with post-processing (unless the *-oc* argument (only collect) is used). In other case, 1+1 jobs are submitted. You can use all submission arguments to control the submission (see the submission section), e.g.:

```
JKCS4_collect XTB -oc -tpj 2 -cleanfull
JKCS4_collect DFT_freq -loc
JKCS4_collect ABC -par q64 -cpu 1 -time 1-00:00:00 -arraymax 2
JKCS4_collect DLPNO -oc -par qtest -time 10:00
```

1.10.1 API

<string> name of the folder which should be analysed (e.g. XTB)

-clean after collecting, removes all unnecessary files (.xyz|.log|.out|.com|.cmd). This saves memory and significantly reduces number of files.

-cleanfull after collecting, completely removes the analysed folder. This saves memory and significantly reduces number of files.

-JKQCrecollect if it happens that something fails, or you did collect folder when the jobs were still running, then the pickle files in subfolders are already created and you must force to recollect the QC outputs by this argument. (Equivalent to *rm XTB/*/*/*.pkl XTB/*/*.pkl* on init.)

1.11 JKCS5_filter

Note: Please, be aware, that JKCS5_filter script is available only due to historical development of JKCS. Currently, JKCS4_collect is coupled with JKQC and all filtering and structure processing is done via JKQC. Nevertheless, I will still keep the old JKCS5_filter script available.

JKCS5_filter provides different methods to filter the large number of structures in the configurational sampling. As input, JKCS5_filter takes `collectionXXX.txt` or `resultsXXX.dat` (unless otherwise specified) created by JKCS4_collect. After the filtering has been applied, the remaining structures are written into `resultsXXX_FILTERED.dat`. We distinguish different filtering operations:

- Uniqueness filtering (produces only unique structures)
- Filtering of failures (removing for instance reacted or exploded structures)
- Threshold filtering (removes structures which for instance has values greater then specified value)
- Sampling/selection (even when many structures remains, only several are uniformly selected so that they maximally represent the remaining set)

Hint: For help use:

```
JKCS5_filter -help
```

1.11.1 Uniqueness

Uniqueness filtering removes structures that are too similar to other structures. When calling JKCS4_collect, a uniqueness filter is automatically applied on `collectionXXX.txt` and the results are stored in `resultsXXX.dat`. This filter assumes two structures with energy difference less than 0.001 hartree and gyration radius (Rg) difference less than 0.01 Å to be the same. Users can run the JKCS5_filter uniqueness filter to fine-tune these uniqueness thresholds. Otherwise, users do not have to bother with uniqueness.

1.11.2 Outliers

Outlier filtering removes structures that have properties too far away from that of the current lowest energy structure. This operation takes as an argument the threshold for specific properties above which a structure is considered an outlier. This filter uses both relative and absolute thresholds for the el. energy. We could ask for a rel. el. energy threshold (e.g., '-d 3.5'). Any structure that has an el. energy more than 3.5 kcal/mol higher than the lowest energy structure will be removed. Or we could define an abs. el. energy threshold '-en' (e.g., '-en 100'), and structures with energy higher than -100 Hartree will be removed. For the gyration radius, only absolute thresholds can be applied (e.g., '-rg 6') removes all structures with gyration radius higher than 6 Å.

1.11.3 Selection

Sampling (or selection) uniformly (covering PES uniformly) picks a specified number of structures from the `resultsXXX.dat` file after applying outlier filtering. This is useful when there are too many structures left for calculations on a higher level of theory. Hence, to reach a reasonable computational cost, we take only very different conformers. As sampling omits some minima, it might result in losing the global minimum structure, but due to the uniform sampling, the global minimum should not be far from one of the other selected minima.

These three filter operations can also be combined. You can thus call JKCS5_filter to apply an uniqueness and outlier filter on a resultsXXX.dat file and then uniformly pick a sample of the remaining structures.

1.12 About & Installation

1.12.1 What is JKQC?

Jammy Key for Quantum Chemistry (JKQC) databases of molecular clusters makes manipulation with quantum chemistry (QC) output files and data (stored in pandas dataframe) available for easy post-processing (e.g. filtering, averaging).

The QC output files (`.log`, `.out`, `.xyz`) into a database (`.pkl`), which takes significantly less memory and accelerates the data post-processing (e.g., printing energies, including various corrections, and calculation formation properties for ACDC). Depending on the files sizes/cluster sizes/types, the pikling takes approximately from 15 seconds/1000 structures to 3 minutes/1000 structures. Nevertheless, the post-processing usually takes from -immediately- to a few seconds. (Sorry, JKQC is not parallelized but I am working on it.)

Hint: See JKQC help to get idea what everything it can do:

```
JKQC -help
```

1.12.2 Installation

Although there are several ways how to utilize only JKQC, the most easiest is anyway installing JKCS. JKQC comes with it for free ;-). See https://jkcs.readthedocs.io/en/latest/JKCSSetupAndInstallation.html

1.13 Manipulation

Hint: Use JKQC help to get idea what everything it can do:

```
JKQC -help
```

Generally, use JKQC in one of the following formats:

```
JKQC <File(s)> <Database(s)> <Options/Parameters>
```

However, I would first recommend to keep the file names in the below suggested format (automatically done within JKCS).

1.13.1 File names

To get full functionality of JKQC, keep the file names in the following format:

```
>> ls
1sa.log 1am.log 2am3sa-1conf.log 2am3sa-2conf.log 2sa-12_23.log
1sa.xyz 1am.xyz 2am3sa-1conf.xyz 2am3sa-2conf.xyz 2sa-12_23.xyz
```

This means that name has of formation of enumerated monomers composition and any additional comments are separated by a hyphen.

These are strong recommendations for your file names:

Table 1: Nomenclature for molecules

neutral	positive	negative
1sa = sulfuric acid		1b = bisulphate
1msa = methanesulfonic acid		1mb = methanebisulphate
1nta = nitric acid		1nt = nitrate
1am = ammonia	1am1p = ammonium	
1ma = methylamine	1ma1p = methylammonium	
1dma = dimethylamine	1dma1p = dimethylammonium	
1tma = trimethylamine	1tma1p = trimethylammonium	
1eda = ethylendiamine	1eda1p = ethylendiammonium	
1gd = guanidine	1gd1p = guanidium	
1w = water	1w1p = hydronium	1oh = hydroxide

Hint: If you work with molecules and want to calculate, e.g., atomization energies, use the following naming:

```
>> ls
1H.log 1C.log 3C8H.log 3C8H-stretched.log
1H.xyz 1C.xyz 3C8H.xyz 3C8H-stretched.xyz
```

Do not forget to use correct spin multiplicities, e.g.: H=2, C,O,S=3, N=4.

Note: By default, it is assumed that structures are saved in `*.xyz` file, the Gaussian/XTB output is saved in `*.log` files, and ORCA outputs in `*.out` files. You can however modify that by, e.g.:

```
JKQC -collect log -orcaext log -out collectedorca.pkl
```

1.13.2 Database manipulation

Input

<Files>

input files can be any .log, .out, and .xyz files. However, if you specify, e.g., `*.log`, it will collect information from all files with the same name.

database.pkl loads `database.pkl` file. You can load several databases into one

Table 2: Input data

Specified	Description
NOTHING	takes in all .log files working in folder
FILES	takes in all specified .log, .out, .xyz files
DATABASES	takes in all specified (-in) .pkl databases
COMBINED	FILES and DATABASES combined

–folder <PATH> collects data from a given folder

–collect <string> collects data for

Output

–out database.pkl output `database.pkl` pickled file

Table 3: Output database

Specified	Description
NOTHING	in classified conditions: mydatabase.out
DATABASE	saves all input data into -out specified .pkl database

You can print various properties (see the section below), e.g.:

```
JKQC *.log -b -el    #[basename] [electronic_energy]
```

You can print various other files:

-xyz creates xyz files for all pickled files

-movie concatenate all xyz into `movie.xyz`

-imos_xlsx Excel sheet input for IMoS

1.13.3 Printing properties

See JKQC help `JKQC -help` for all various properties. For instance, you can print (name and) electronic energy from files/database:

```
JKQC *.log -b -el        #[basename] [electronic_energy]
JKQC database.pkl -b -el #significantly faster
```

1.13.4 Processing

You can extract (name and) electronic energy for a specific cluster(s):

```
JKQC in.pkl -extract 1sa2w -b -el
JKQC in.pkl -extract 3sa,1sa0-10w -b -el
```

You sort your data with respect to el = electronic_energy/g = gibbs_free_energy

```
JKQC in.pkl -sort el -b -el
JKQC in.pkl -sort g -out out.pkl
```

Certainly utilize some filtering techniques (see JKQC help for greater detail):

- Uniqueness: *-uniq rg,el* or *-arbalign 0.38* (CITE ArbAlign)
- Low/High cutoff: *-pass lf 0* (removes structures with negative/imaginary frequencies), *-cut rg 10* (select structures with *Rg* less than 10 Angstrom), *-cutr el 10* (selects only 10 lowest kcal/mol structures)
- Reacted: *-reacted* (compares all conformers and tries to remove some reacted/exploded structures)

1.13.5 Post-processing

This an example how to print binding free energies in kcal/mol while taking only the global free energy minimum

```
JKQC clusters.pkl monomers.pkl -ct -g -glob -formation -unit -noex
```

and now with using treatment for low vibrational frequencies and anharmonicity correction (CITE Grimme):

```
JKQC clusters.pkl monomers.pkl -ct -g -glob -fc 100 -v 0.996 -formation -unit -noex
```

and now, assuming that the `*.log` files (Gaussian) were accompanied with `*.out` (ORCA) single-point corrections:

```
JKQC clusters.pkl monomers.pkl -ct -gout -globout -fc 100 -v 0.996 -formation -unit -noex
```

and now, at different temperature:

```
JKQC clusters.pkl monomers.pkl -ct -gout -globout -fc 100 -v 0.996 -formation -unit -
↪noex -temp 270
```

1.14 How to QML

JKML by default uses QML (https://www.qmlcode.org/index.html) for kernel ridge regression (KRR) applied on molecular representation (FCHL) to predict molecular system properties.

Note: When setting up JKCS, you should use *-qml* in order to install QML, e.g.:

```
sh setup.sh -r grendel -qml
```

I show examples for electronic energy but other molecular properties can be modelled, too (use *-column* argument). Training directly on electronic energy of a cluster

$$E^{DFT}_{cluster}$$

is alright only if you train/test on single-cluster conformers (i.e. no large energy difference between modelled molecules are present).

Otherwise, it is better to model binding energies of the studied clusters

$$\Delta E^{DFT} = E^{DFT}_{cluster} - \sum E^{DFT}_{monomer},$$

or eventually atomization energies in the case of molecules.

Quite many studies showed that one can perform a cheaper but faster QC calculation (e.g., at XTB level) and then only model the small differences between high level and low theory (both calculated for the same structure geometries):

$$\Delta E^{DFT} = E^{DFT}_{cluster} - \sum E^{DFT}_{monomer}$$

$$\Delta E^{XTB} = E^{XTB}_{cluster} - \sum E^{XTB}_{monomer}$$

$$\Delta\Delta E^{DFT|XTB} = \Delta E^{DFT} - \Delta E^{XTB}$$

The delta-ML does perform significantly better and low level theory calculations are usually cheap to perform.

1.14.1 Preparing files

The structures should be provided in a pickled .pkl files (see JKQC manual). The file naming must have specific format (you can try, i.e. with caution, to use JKname to rename your files). See examples:

```
$USER: ls
1sa.log 1w.log 1sa1w.log 1sa1w-1.log 1sa1w-2_32.log
1sa.xyz 1w.xyz 1sa1w.xyz 1sa1w-1.xyz 1sa1w-2_32.xyz
....
$USER: JKQC -collect log -out full_DFT.pkl
```

Note: It is assumed that XTB and G16 outputs have extension *.log*. Generally, ORCA output is assumed to have *.out* extension that you can combine it as SP calculation with G16. However, for JKML, please change the extension to *.log* and use *-orcaext log* for JKQC, e.g.:

```
JKQC *.log -orcaext log -out full_DFT.pkl
```

Hint: It is strongly recommended to separate monomers into a single file. Additionally, JKQC can be used to prepare other pickled files, e.g.:

```
JKQC full_DFT.pkl -extract 1sa,1w    -out monomers_DFT.pkl
JKQC full_DFT.pkl -extract 0-3sa0-5w -out train_DFT.pkl
JKQC full_DFT.pkl -extract 4sa0-5w   -out test_DFT.pkl
```

1.14.2 Machine Learning

JKML is used similarly as other JKCS commands

```
JKML -help
JKML -loc -train train_DFT.pkl -test test_DFT.pkl -monomers monomers_DFT.pkl
JKML -par test -time 10:00 -mem 5GB -cpu 2 -train train_DFT.pkl -test test_DFT.pkl -
↪monomers monomers_DFT.pkl
JKML -loc -print 2 -train train_DFT.pkl train_XTB.pkl -test test_DFT.pkl test_XTB.pkl -
↪monomers monomers_DFT.pkl monomers_XTB.pkl
```

Hint: See `Cluster submission` (under JKCS) to understand how to tune cluster parameters.

Whether you plan to use direct-learning or delta-learning is defined by number of files you provide, e.g.:

```
JKML -train train_DFT.pkl -test test_DFT.pkl -monomers M_DFT.pkl
```

or

```
JKML -train train_DFT.pkl train_XTB.pkl -test test_DFT.pkl test_XTB.pkl -monomers M_DFT.
↪pkl M_XTB.pkl
```

Note: You can first train and then test separately, e.g.:

```
JKML -train train_DFT.pkl -monomers monomers_DFT.pkl -loc
JKML -trained model.pkl -test test_DFT.pkl -monomers monomers_DFT.pkl -loc
```

The results can be found in output or in `predicted_QML.pkl`, e.g.:

```
JKQC predicted_QML.pkl -b -el
```

Note: Do not forget to optimize hyperparameters if you use different systems.

1.15 How to SchNetPack

JKML uses the SchNetPack (https://schnetpack.readthedocs.io) for training the neural network potential.

Note: When setting up JKCS, you should use *-nn* in order to install SchNetPack, e.g.:

```
sh setup.sh -r grendel -nn
```

Please, see the QML section or the JKQC section in order to understand how to prepare the input pickle files.

1.15.1 NN parameters

Let us first list all possible parameters required for training/testing/prediction with NN:

-help, -help_nn, -help_adv see help text as it is so far the most decriptive when it comes to the arguments

-nn, -painn to switch for PaiNN architecture of the NN (incompatible with *-qml*). Other architectures (e.g. *-schnet* and *-so3net*) also are available.

-train <HIGH.pkl> [<LOW.pkl>] the training database. In the case you want to use delta learning, use also a low-level-of-theroy database which corresponds to the high level of theory when it comes to file naming. Often you perform training (using GPU) and then you run new jobs where you use the trained model, which can be called as *-trained <model.pkl>*.

-test <HIGH.pkl> [<LOW.pkl>] the test database. In the case you use delta learning, provide also a low-level-of-theroy database which corresponds to the high level of theory when it comes to file naming.

-monomers <HIGH.pkl> [<LOW.pkl>] the database with one representative monomer for each specie (this could be also atoms if you want to use atomization energies; however, the file naming must be modified accordingly). In the case you use delta learning, provide also a low-level-of-theroy database which corresponds to the high level of theory when it comes to file naming.

-eval,-opt,-md <STR.pkl/xyz> [<LOW.pkl>] the database of structures for which you want to predict the properties, which you want to optimize, or for which you want to run MD (when *-spkmd* is used, provide xyz file). In the case you use delta learning, provide also a low-level-of-theroy database which corresponds to the high level of theory when it comes to file naming. See *-help_adv* for additional argument for tuning the optimization/MD.

Several arguments related to training:

-epochs <int> Number of iterations/loops/epoch you want to use for the training.

-batch_size,-bs <int> Number of structures within one batch (subpackage) used for training at the time. In each epoch, NN loops over all batches. Use preferably number which are power of 2.

See other parameters, such as *-nn_train* (overall portion used for training), *-nn_ESpatience* (early stop in case no improvement happens for several epochs), etc. in the *-help_adv*

Some arguments related to the NN architecture:

-nn_ab <int> Size of the feature vector for each atom type. Training of very large feature vectors might take more time.

-nn_int <int> Depth of the NN, i.e. how many interaction layers are used.

-nn_rb <int> Radial basis size, i.e. how many point from the atom to the cutoff distance are taken.

-nn_cutoff, -cutoff <float> The interaction distance.

Note: See the Cluster submission section for understanding how to submit the jobs. Eventually, you can modify *program_SBATCH* function in the *~/.JKCSusersetup.txt*.

1.15.2 Examples

Let us take some DFT data, shuffle them, and split them into two halves:

```
JKQC DFT.pkl -shuffle -out DFT_shuffled.pkl
JKQC DFT_shuffled.pkl -split 2
mv DFT_shuffled_v1.pkl DFTtrain.pkl
mv DFT_shuffled_v2.pkl DFTtest.pkl
JKQC DFT.pkl -sort el -select 1 -extract 1sa,1w,1am -out DFTmonomers.pkl
```

Direct training of electronic energies (dicouraged), where 4 CPU (hence, the number of workers, *-nw*, is also 4) and 1 GPU is used on the *qgpu* partition. Let us do also short training just for a test:

```
JKML -nn -train DFTtrain.pkl -par qgpu -cpu 4 -nw 4 -epochs 10
```

The result *model.pkl* can be then used for testing:

```
JKML -nn -trained model.pkl -test DFTtest.pkl -par q64 -cpu 1
tail -f output
```

The result will be most likely terrible. Hence let us train on el. binding energies instead (the monomers must be obsly calculated at the same method). I will now combine it with the test too.

```
JKML -nn -train DFTtrain.pkl -monomers DFTmonomers.pkl -test DFTtest.pkl -par qgpu -cpu␣
↪4 -nw 4 -epochs 10
```

This should give better results but still most likely not acceptable. So just increase the epochs, training set size, or generaly tune the hyperparameters of the NN architecture above (we have code for it but yet it is not automated).

In the *QML* section, we showed the theory behind delta-learning. We can do the same with NN:

```
JKML -nn -train HIGHtrain.pkl LOWtrain.pkl -monomers HIGHmonomers.pkl LOWmonomers.pkl -
↪par qgpu -cpu 4 -nw 4
JKML -nn -trained model.pkl -test HIGHtest.pkl LOWtest.pkl -monomers HIGHmonomers.pkl␣
↪LOWmonomers.pkl -par q64 -cpu 1
JKML -nn -trained model.pkl -eval str.pkl LOWstr.pkl -monomers HIGHmonomers.pkl␣
↪LOWmonomers.pkl -par q64 -cpu 1
```

Last line corresponds to prediction of binding energies of new structures.

Hint: You can train any other property by using *-column <str> <str>*:

```
JKQC DFT.pkl -info
JKML -nn -train DFT.pkl -column "log" "zero_point_energy" -par qgpu -cpu 4 -nw 4
```

Other properties can be added to the pickle file by using *-add <column> <file>* from a file that contains two columns: file basename and the property of interest:

```
JKQC DFT.pkl -add mobility mobility_file.txt -out DFT_mob.pkl
JKML -nn -train DFT.pkl -eval str.pkl -column "extra" "mobility" -par qgpu -cpu 4 -nw 4
```

1.15.3 FORCES

If we collected the forces as well (*JKQC -folder DFT -collect log -forces -out DFTwithforces.pkl*), we can the very same as above and forces will be automatically included in the training. The cost function is by default defined as 1:9 energy and force mean absolute errors (MAEs). You can turnoff trainig of the forces by using *-noforces*. The output will then contain MAE of both energies and forces too.

Note: The cutoff value can be specified for both training and OPT/MD. I have no clue what happens if you specify different value for both of them.

The structure (TODO: so far applicable only for one structure) can be optimized as:

```
JKML -nn -trained model.pkl -opt str.pkl -par q64 -cpu 1
```

Note: See other parameters of the optimizer or for MD by using *-help_adv*.

You can run also MD as:

```
JKML -nn -trained model.pkl -md str.pkl -par q64 -cpu 1 -md_timestep 0.2 -md_steps 1000 -
↪md_temperature 300
```

This will initialize velocities with respect to Maxwell Boltzmann distribution and then run MD through ASE with the Langevin thermostat. Well this part is not much tuneable yet as we sticked to the spkmd script within SchNetPack. This one you can initialize as:

```
JKML -spkmd -trained model.pkl -md 3sa3w.xyz -par q64 -cpu 1 -md_timestep 0.2 -md_steps␣
↪1000 -md_temperature 300 -langevin
```

The results can be afterwards easily visulised as:

```
JKpython
python analyse.py
```

Other predefined options can be found in the *-help_adv* or see https://schnetpack.readthedocs.io/en/latest/userguide/md.html for more details on the spkmd commands. You add several of these command by using *-spkmd_extra* <*string*>.

1.16 About & Usage

Table of Contents

1.16.1 Introduction

JKTS is a tool designed to facilitate a streamlined and automatic process of generating and monitoring files integral for transition state search in atmospheric chemistry, through the use of quantum chemical programs ORCA and Gaussian16. It implements the multiconformer transition state theory (MC-TST) to achieve realistic understanding and prediction of the kinetics in atmospheric chemical reactions between organic species and oxidants in the atmosphere. Currently, JKTS supports the reaction of hydrogen abstraction by OH radical and OH radical addition to carbon-carbon double bonds. Furthermore, tunneling effects are accounted for using the Eckart tunneling correction factor, by fitting a unsymmetrical Eckart potential to the reactant complex, transition state, and product complex energies. In the case of reactions involving ligjt atoms, such as hydrogen, the tunneling correction has shown to be crucial.

When setting up JKTS, you should use *-TS* in order to install, e.g.:

```
sh setup.sh -r grendel -TS
```

1.16.2 Workflow Overview

The JKTS tool processes transition state molecules and reactants/products using distinct workflows:

Transition State Molecules Workflow

1. Conformer sampling with CREST.
2. Constrained optimization of the conformers.
3. Transition state optimization.
4. Final energy refinement with DLPNO-CCSD(T) calculations.

Reactants and Products Workflow

1. Conformer sampling with CREST.
2. Geometry optimization of the conformers.
3. Final energy refinement with DLPNO-CCSD(T) calculations.

Note: Insert a workflow diagram here.

1.16.3 Using JKTS

JKTS can be configured with various options for different computational scenarios. As an example, to calculate the reaction dynamics of hydrogen abstraction from methane with the OH radical, use the following command:

```
JKTS CH4.xyz -OH
```

This command will create three directories: **products**, **reactants**, and **CH4_H1**.

- The **products** directory includes the H2O molecule and configurations of products from hydrogen abstraction. For methane, there is only one product type since all hydrogens are equivalent.
- The **CH4_H1** directory is for the case of single hydrogen abstraction from methane. THe JKTS program tries to treat chemically equivalent hydrogens. A rather simple but efficient approach is used by storing the indexes of carbons with three (or more) hydrogens bonded to it during the generation of the initial guess for the TS state. These are assumed to correspond to the molecules methyl groups and that these hydrogens are chemically equivalent. This assumption is justified based on the initial use of the CREST program for the conformer sampling and therefore the subsequent sampling of the methyl groups.
- The **reactants** directory contains the methane molecule and the OH radical, following the workflow for reactants.

Adjusting Monitoring Time and Restarting Jobs

For tasks that are not computationally intensive, such as the transition state search for methane, the monitoring duration can be tailored with the `-time` argument. To set the monitoring time to five hours, input the following:

```
JKTS CH4.xyz -OH -time 5:00:00
```

If the monitoring were to end prematurely, for instance during constrained geometry optimization, the calculations are able to be restarted with the command:

```
JKTS *.log
```

The wildcard symbol (*) matches all *.log* files in the directory. Logs from completed tasks are moved to **CH4_H1/log_files**, so running the command within **CH4_H1** targets only the *.log* files of incomplete tasks. JKTS will assess each *.log* file to determine the last completed step and will resume the workflow accordingly.

If we simply wanted to perform only constrained optimization without moving on to the transition state search, the *-auto false* option is available:

```
JKTS *.log -auto false
```

Advanced Usage

To run JKTS with specific settings, like a custom level of theory:

```
JKTS yourfile.xyz -OH --low_level "B97-3c" --high_level "B3LYP 6-31++g(d,p)"
```

Keep in mind the natural limitation of ORCA and Gaussian16 in relation to the basis set and method. Although ORCA supports most of the well-known methods and basis sets…

Monitoring of log files

JKTS monitors the log file with certain intervals to avoid overwhelming communication between computers. By default the program allows this communication a *100* times with a certain time interval between each check determined by `interval`. By default the time between checks is calculated based on the size of the input molecule. However, the maximum number of attempts to check the log files and the interval between them can be modified as:

```
JKTS yourfile.xyz -OH -interval 500 -attempts 200 -initial_delay 2000
```

Resulting in an initial delay of 2000 seconds before the log files are checked with 500 seconds interval between each check and this check is performed up to 200 times.

1.16.4 Command Line Arguments

JKTS accepts various arguments to control its behavior:

Input Commands	Description
-h, --help	Print help page
-auto	Enable automated processing of predefined workflow. See Workflow for more. [def = True]
-OH	Perform H abstraction with OH radical
-CC	Perform addition to C=C bonds
-OH_CC	Perform OH addition to C=C bonds
-G16	Gaussian16 is used for QC calculations (default)
-ORCA	ORCA is used for QC calculations
-constrain	Constrain is integrated into relevant input file [def = True]
-reactants	Prepare folder for reactants [def = True]
-products	Prepare folder for products [def = True]
-NEB	Prepare input file for Nudged Elastic Band [def = False]
-k	Calculate Multiconformer Transition State rate constant def = [True]
--high_level	Specify the high level of theory for QC method TS optimization [def = wB97X-D aug-cc-pVTZ]
--low_level	Specify the low level of theory for preoptimization [def = wB97X-D 6-31+G(d,p)]
-cpu	Number of CPUs [def = 4]
-mem	Amount of memory allocated for job [def = 8000mb]
-par	Partition to use [def = qany]
-time	Specify how long time the manager monitors [def = 144 Hours]
-interval	Set time interval between checks of log files [def = based on molecule size]
-initial_delay	Set an initial delay before checking log files [def = based on molecule size]
-attempts	Set how many times a log files should be checked [def = 100]
-max_conformers	Set max number of conformers from CREST [def = 50]
-freq_cutoff	Set cutoff for TS imaginary frequency to [int] cm^-1 [def = -200]
-reaction_angle	Set the angle of the active site of transition state to [int] degrees [def = 175.0]
-ewin	Set energy threshold to [int] kcal/mol for CREST conformer sampling [def = 8]
-info	Print information of molecules in log files or .pkl file
-XQC, -YQC, -QC	(G16 only) Use specified SCF algortihm instead of Direct Inversion of Iterative Space (DIIS)