Molecular Dynamics with OpenMM

1. Using the high performance computing (HPC)

MD simulations can be very computationally intensive, so it may be a good idea to make use of high-performance computing (HPC) infrastructure, if available (instructions for running locally are described in §). Here, we present a step-by-step guide for the Flemish HPC infrastructure (VSC). This infrastructure is accessible for free to all researchers and students at the UGent, request access at:

https://www.ugent.be/hpc/en/access/policy/access

1.1 Preparing the HPC environment

Installing python packages on the HPC

Unfortunately, not every package we need is installed on the HPC. Luckly, we can install many python package ourselves very easily. More information at: https://docs.vscentrum.be/software/python_package_management.html#alternatives-to-conda

Note: At the VSC website, it is mentioned to use 'export PATHONPATH=…'. This is however not useful in our case, as this replaces the default python path preventing the use of the modules. Instead, we will use the python function sys.path.append in the python script.

Here we will make use of pip.

- Log in on the HPC client site: https://login.hpc.ugent.be/pun/sys/dashboard/
- Open the shell via Clusters >_login Shell Access

- Load the appropriate Python module, i.e., the one you want the python package to be available for:

```
$ module load openMM/8.0.0-foss-2022a
```

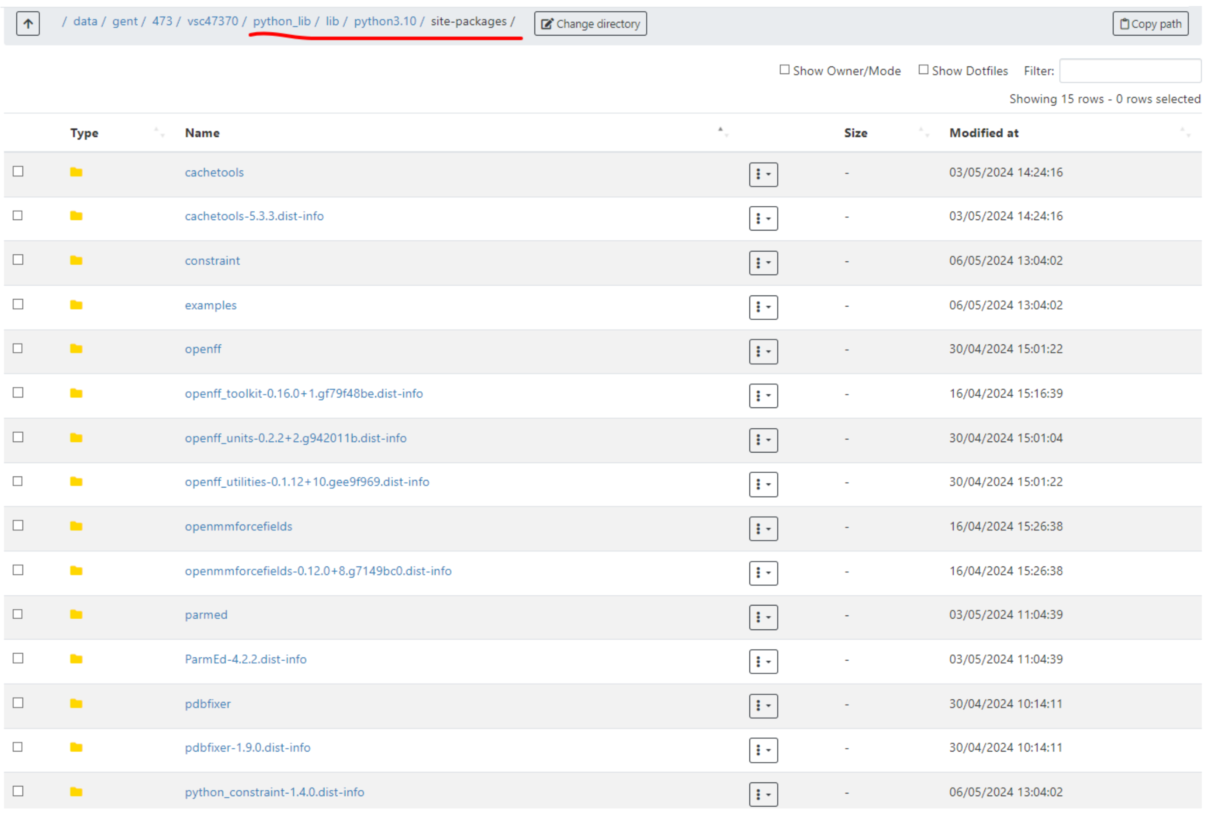
- Create a directory to hold the packages you install, the last three directory names are mandatory:

```
$ mkdir -p "${VSC_DATA}/python_lib/lib/python3.10/site-packages/"
```

- Install the package, using the --prefix install option to specify the install path:

```
$ pip install --prefix="${VSC_DATA}/python_lib" -U
git+https://github.com/openforcefield/openff-toolkit.git
$ pip install --prefix="${VSC_DATA}/python_lib" -U git+
https://github.com/openmm/openmmforcefields.git
$ pip install --prefix="${VSC_DATA}/python_lib" cachetools
$ pip install --prefix="${VSC_DATA}/python_lib" python-constraint
$ pip install --prefix="${VSC_DATA}/python_lib" ParmEd
$ pip install --prefix="${VSC_DATA}/python_lib" -U
git+https://github.com/openmm/pdbfixer.git
```

Now our $VSC_DATA should contain the folders ‘python_lib>lib>python3.10>site-packages’. The last folders contains all the installed packages and should look something like this:



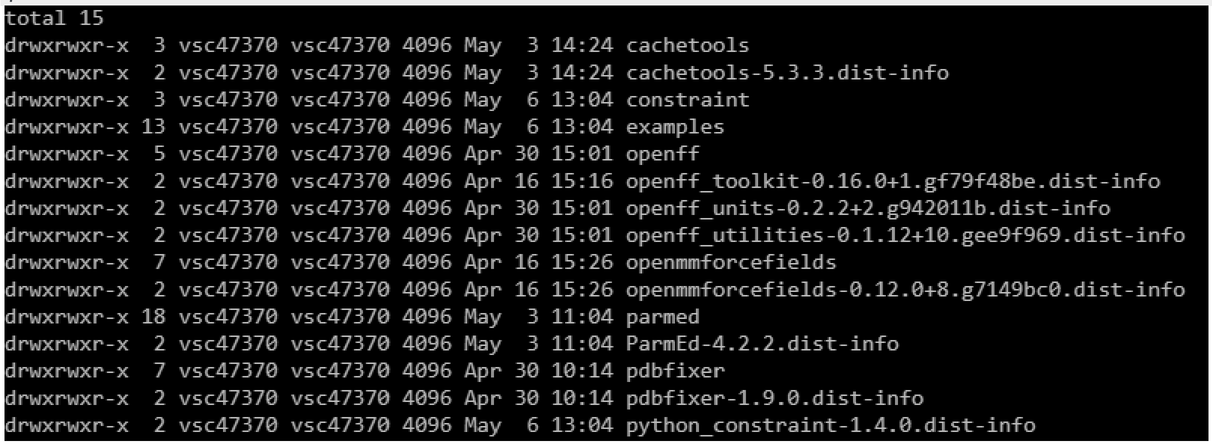
/ data / gent / 473 / vsc47370 / python_lib / lib / python3.10 / site-packages /

Showing 15 rows - 0 rows selected

Type	Name	Size	Modified at
📁	cachetools	-	03/05/2024 14:24:16
📁	cachetools-5.3.3.dist-info	-	03/05/2024 14:24:16
📁	constraint	-	06/05/2024 13:04:02
📁	examples	-	06/05/2024 13:04:02
📁	openff	-	30/04/2024 15:01:22
📁	openff_toolkit-0.16.0+1.gf79f48be.dist-info	-	16/04/2024 15:16:39
📁	openff_units-0.2.2+2.g942011b.dist-info	-	30/04/2024 15:01:04
📁	openff_utilities-0.1.12+10.gee9f969.dist-info	-	30/04/2024 15:01:22
📁	openmmforcefields	-	16/04/2024 15:26:38
📁	openmmforcefields-0.12.0+8.g7149bc0.dist-info	-	16/04/2024 15:26:38
📁	parmed	-	03/05/2024 11:04:39
📁	ParmEd-4.2.2.dist-info	-	03/05/2024 11:04:39
📁	pdbfixer	-	30/04/2024 10:14:11
📁	pdbfixer-1.9.0.dist-info	-	30/04/2024 10:14:11
📁	python_constraint-1.4.0.dist-info	-	06/05/2024 13:04:02

Or in the console

```
$ cd $VSC_DATA/python_lib/lib/python3.10/site-packages
$ ls -l
```



```
total 15
drwxrwxr-x  3 vsc47370 vsc47370 4096 May  3 14:24 cachetools
drwxrwxr-x  2 vsc47370 vsc47370 4096 May  3 14:24 cachetools-5.3.3.dist-info
drwxrwxr-x  3 vsc47370 vsc47370 4096 May  6 13:04 constraint
drwxrwxr-x 13 vsc47370 vsc47370 4096 May  6 13:04 examples
drwxrwxr-x  5 vsc47370 vsc47370 4096 Apr 30 15:01 openff
drwxrwxr-x  2 vsc47370 vsc47370 4096 Apr 16 15:16 openff_toolkit-0.16.0+1.gf79f48be.dist-info
drwxrwxr-x  2 vsc47370 vsc47370 4096 Apr 30 15:01 openff_units-0.2.2+2.g942011b.dist-info
drwxrwxr-x  2 vsc47370 vsc47370 4096 Apr 30 15:01 openff_utilities-0.1.12+10.gee9f969.dist-info
drwxrwxr-x  7 vsc47370 vsc47370 4096 Apr 16 15:26 openmmforcefields
drwxrwxr-x  2 vsc47370 vsc47370 4096 Apr 16 15:26 openmmforcefields-0.12.0+8.g7149bc0.dist-info
drwxrwxr-x 18 vsc47370 vsc47370 4096 May  3 11:04 parmed
drwxrwxr-x  2 vsc47370 vsc47370 4096 May  3 11:04 ParmEd-4.2.2.dist-info
drwxrwxr-x  7 vsc47370 vsc47370 4096 Apr 30 10:14 pdbfixer
drwxrwxr-x  2 vsc47370 vsc47370 4096 Apr 30 10:14 pdbfixer-1.9.0.dist-info
drwxrwxr-x  2 vsc47370 vsc47370 4096 May  6 13:04 python_constraint-1.4.0.dist-info
```

Loading the required modules

Now we load all the required modules on the HPC. This is a bit fuss since we must load the dependencies of our own installed packages as well. Luckily, we only have to do this once and save the loaded modules to a collection.

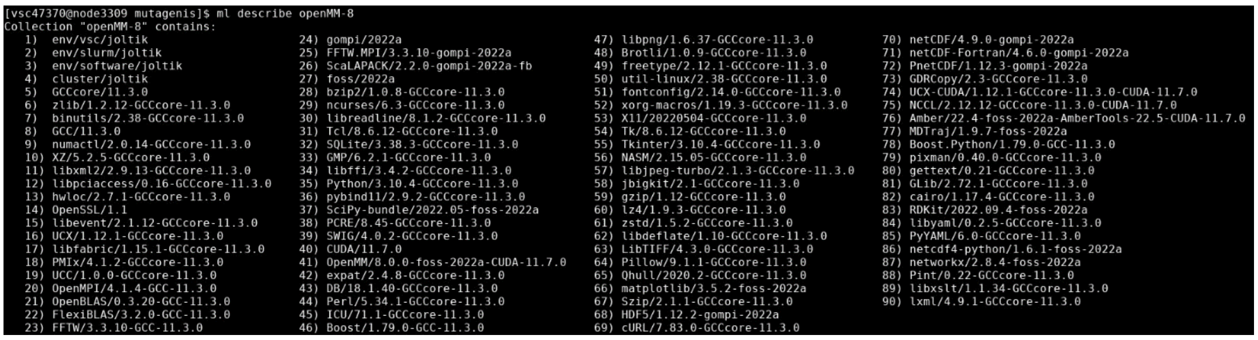
- Swap cluster to joltik and load the modules.

```
$ module swap cluster/joltik
$ module load OpenMM/8.0.0-foss-2022a-CUDA-11.7.0 Amber/22.4-foss-2022a-
AmberTools-22.5-CUDA-11.7.0
$ module load MDTraj/1.9.7-foss-2022a RDKit/2022.09.4-foss-2022a
networkx/2.8.4-foss-2022a
$ module load libyaml/0.205-GCCcore-11.3.0 PyYAML/6.0-GCCcore-11.3.0
netcdf4-python/1.6.1-foss-2022a Pint/0.22-GCCcore-11.3.0 lxml/4.9.1-
GCCcore-11.3.0 libfabric/1.15.1-GCCcore-11.3.0
```

- Now we check if everything is loaded correctly.

```
$ ml
```

The output should look something like this.



```
[vsc47370@node3309 mutagenis]$ ml describe openMM-8
Collection "openMM-8" contains:
   1)  env/vsc/joltik                   24) gompi/2022a                              47) libpng/1.6.37-GCCcore-11.3.0          70) netCDF/4.9.0-gompi-2022a
   2)  env/slurm/joltik                 25) FFTW.MPI/3.3.10-gompi-2022a              48) Brotli/1.0.9-GCCcore-11.3.0           71) netCDF-Fortran/4.6.0-gompi-2022a
   3)  env/software/joltik              26) ScaLAPACK/2.2.0-gompi-2022a-fb           49) freetype/2.12.1-GCCcore-11.3.0        72) PnetCDF/1.12.3-gompi-2022a
   4)  cluster/joltik                   27) foss/2022a                               50) util-linux/2.38-GCCcore-11.3.0        73) GDRCopy/2.3-GCCcore-11.3.0
   5)  GCCcore/11.3.0                   28) bzip2/1.0.8-GCCcore-11.3.0               51) fontconfig/2.14.0-GCCcore-11.3.0      74) UCX-CUDA/1.12.1-GCCcore-11.3.0-CUDA-11.7.0
   6)  zlib/1.2.12-GCCcore-11.3.0       29) ncurses/6.3-GCCcore-11.3.0               52) xorg-macros/1.19.3-GCCcore-11.3.0     75) NCCL/2.12.12-GCCcore-11.3.0-CUDA-11.7.0
   7)  binutils/2.38-GCCcore-11.3.0     30) libreadline/8.1.2-GCCcore-11.3.0         53) X11/20220504-GCCcore-11.3.0           76) Amber/22.4-foss-2022a-AmberTools-22.5-CUDA-11.7.0
   8)  GCC/11.3.0                       31) Tcl/8.6.12-GCCcore-11.3.0                54) Tk/8.6.12-GCCcore-11.3.0              77) MDTraj/1.9.7-foss-2022a
   9)  numactl/2.0.14-GCCcore-11.3.0    32) SQLite/3.38.3-GCCcore-11.3.0             55) Tkinter/3.10.4-GCCcore-11.3.0         78) Boost.Python/1.79.0-GCC-11.3.0
  10) XZ/5.2.5-GCCcore-11.3.0           33) GMP/6.2.1-GCCcore-11.3.0                 56) NASM/2.15.05-GCCcore-11.3.0           79) pixman/0.40.0-GCCcore-11.3.0
  11) libxml2/2.9.13-GCCcore-11.3.0     34) libffi/3.4.2-GCCcore-11.3.0              57) libjpeg-turbo/2.1.3-GCCcore-11.3.0    80) gettext/0.21-GCCcore-11.3.0
  12) libpciaccess/0.16-GCCcore-11.3.0  35) Python/3.10.4-GCCcore-11.3.0             58) jbigkit/2.1-GCCcore-11.3.0            81) GLib/2.72.1-GCCcore-11.3.0
  13) hwloc/2.7.1-GCCcore-11.3.0        36) pybind11/2.9.2-GCCcore-11.3.0            59) gzip/1.12-GCCcore-11.3.0              82) cairo/1.17.4-GCCcore-11.3.0
  14) OpenSSL/1.1                       37) SciPy-bundle/2022.05-foss-2022a          60) lz4/1.9.3-GCCcore-11.3.0              83) RDKit/2022.09.4-foss-2022a
  15) libevent/2.1.12-GCCcore-11.3.0    38) PCRE/8.45-GCCcore-11.3.0                 61) zstd/1.5.2-GCCcore-11.3.0             84) libyaml/0.2.5-GCCcore-11.3.0
  16) UCX/1.12.1-GCCcore-11.3.0         39) SWIG/4.0.2-GCCcore-11.3.0                62) libdeflate/1.10-GCCcore-11.3.0        85) PyYAML/6.0-GCCcore-11.3.0
  17) libfabric/1.15.1-GCCcore-11.3.0   40) CUDA/11.7.0                              63) LibTIFF/4.3.0-GCCcore-11.3.0          86) netcdf4-python/1.6.1-foss-2022a
  18) PMIx/4.1.2-GCCcore-11.3.0         41) OpenMM/8.0.0-foss-2022a-CUDA-11.7.0      64) Pillow/9.1.1-GCCcore-11.3.0           87) networkx/2.8.4-foss-2022a
  19) UCC/1.0.0-GCCcore-11.3.0          42) expat/2.4.8-GCCcore-11.3.0               65) Qhull/2020.2-GCCcore-11.3.0           88) Pint/0.22-GCCcore-11.3.0
  20) OpenMPI/4.1.4-GCC-11.3.0          43) DB/18.1.40-GCCcore-11.3.0                66) matplotlib/3.5.2-foss-2022a           89) libxslt/1.1.34-GCCcore-11.3.0
  21) OpenBLAS/0.3.20-GCC-11.3.0        44) Perl/5.34.1-GCCcore-11.3.0               67) Szip/2.1.1-GCCcore-11.3.0             90) lxml/4.9.1-GCCcore-11.3.0
  22) FlexiBLAS/3.2.0-GCC-11.3.0        45) ICU/71.1-GCCcore-11.3.0                  68) HDF5/1.12.2-gompi-2022a
  23) FFTW/3.3.10-GCC-11.3.0            46) Boost/1.79.0-GCC-11.3.0                  69) cURL/7.83.0-GCCcore-11.3.0
```

- Now to avoid to have to load everything each time again, we will save them to a collection. More info about collections at: https://docs.hpc.ugent.be/Windows/running_batch_jobs/?h=#save-and-load-collections-of-modules

```
$ ml save <collection_name>
```

Before proceeding

Firstly, we must tell python to look for our manually installed packages in the $VSC_DATA storage. We will not do this using the 'export' functions in linux, rather via the sys.path in python itself.

- Open the utils.py file in the editor (:| > edit)
- At the top we will find: sys.path.append('/data/gent/473/vsc47370/python_lib/lib/python3.10/site-packages')
- Replace the **'473/vsc47370'** you own VSC user id! (can be found by

```
$ echo $VSC_DATA
```

1.2 Preparing the input files

Now our HPC environment is working, we can start preparing the necessary files our simulation run.

1.2.1 Preparing the protein

Protein structures found on the PDB seldom include H-atoms and often complete. Therefore, we need to add the H-toms and any missing terminal groups before simulating. Luckly, the

MDsim.py script can do this for us. Alternatively, we can use the openmm-setup application (https://github.com/openmm/openmm-setup).

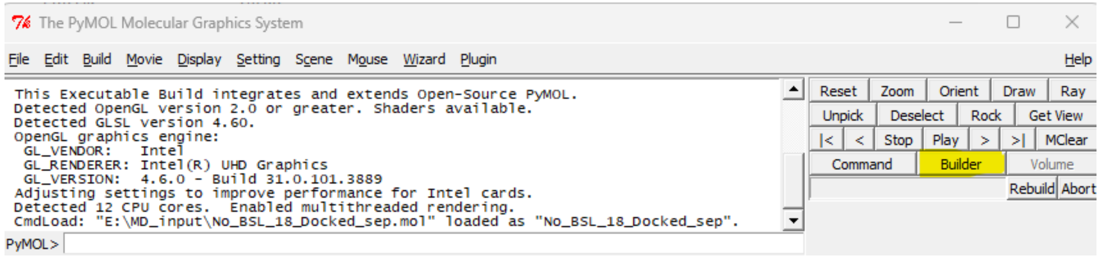
1.2.2 Preparing the ligand

The ligand requires a bit more work than the protein. The ligand must be docked in the protein binding site in advance. For detailed instructions to prepare and run a docking with AutoDock Vina, please watch the video at:

https://vina.scripps.edu/tutorial/

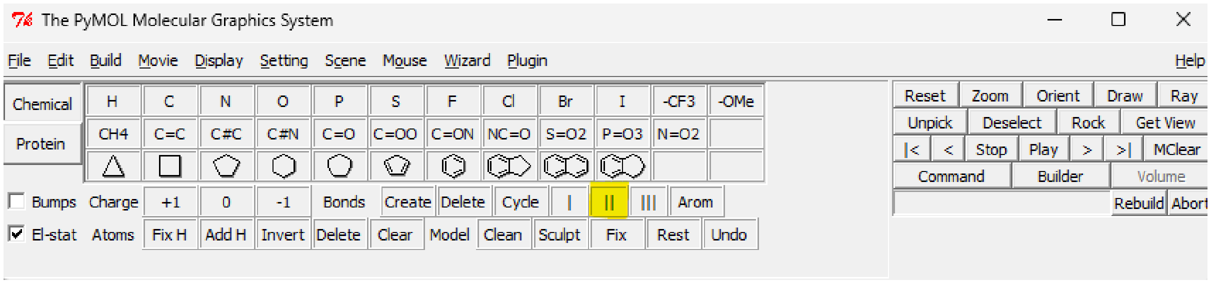
The .pdbqt file removes all double bonds, so now we first need to re-add these to the ligand.

- Open PyMOL and load the ligand.pdbqt file (not all versions PyMOL support .pdbqt files, so may need to convert it to a .pdb file first.).
- Select the 'Builder' tab in the toolbar, and make sure the line representation is displayed.


The PyMOL Molecular Graphics System
File Edit Build Movie Display Setting Scene Mouse Wizard Plugin Help
Reset Zoom Orient Draw Ray
Unpick Deselect Rock Get View
|< < Stop Play > >| MClear
Command Builder Volume
Rebuild Abort
PyMOL>

```
This Executable Build integrates and extends Open-Source PyMOL.
Detected OpenGL version 2.0 or greater. Shaders available.
Detected GLSL version 4.60.
OpenGL graphics engine:
 GL_VENDOR:   Intel
 GL_RENDERER: Intel(R) UHD Graphics
 GL_VERSION:  4.6.0 - Build 31.0.101.3889
Adjusting settings to improve performance for Intel cards.
Detected 12 CPU cores.  Enabled multithreaded rendering.
CmdLoad: "E:\MD_input\No_BSL_18_Docked_sep.mol" loaded as "No_BSL_18_Docked_sep".
```

- Now you may notice that the double bounds are gone. We will re-add these manually. Select the double bound tool and click on the bonds you want to double


The PyMOL Molecular Graphics System
File Edit Build Movie Display Setting Scene Mouse Wizard Plugin Help
Chemical
Protein
H C N O P S F Cl Br I -CF3 -OMe
CH4 C=C C#N C#N C=O C=OO C=ON NC=O S=O2 P=O3 N=O2
Bumps Charge +1 0 -1 Bonds Create Delete Cycle | || ||| Arom
El-stat Atoms Fix H Add H Invert Delete Clear Model Clean Sculpt Fix Rest Undo
Reset Zoom Orient Draw Ray
Unpick Deselect Rock Get View
|< < Stop Play > >| MClear
Command Builder Volume
Rebuild Abort

- Save the molecule as **.mol** file.

1.2.3 Preparing the configuration file

With the protein, and optional ligand file, ready, we can start setting up the rest of the simulation parameters. The easiest way to do this is by making a configuration file using MDsim-setup. This can be run as .exe on Windows devices or in an interactive session on the HPC.

- Add the MDsim-setup.py file to the $VSC_SCRATCH

- Open an interactive session on the HPC. Any cluster should work, but I recommend using joltik. (https://docs.hpc.ugent.be/Windows/running_interactive_jobs/)

- navigate to $VSC_SCRATCH, load the wxPython module and launch

```
$ cd $VSC_SCRATCH
$ module load wxPython/4.2.0-foss-2021b
$ python MDsim-setup
```

A window should now appear where you can input the needed parameters. When everything is filled in correct, click OK and name you file. This name will be used for both the configuration file and the output generated by MDsim.

1.2.4 Preparing the job script

The job script is needed to run our simulation on the HPC. More info about the job script can be found here: https://docs.hpc.ugent.be/Windows/jobscript_examples/

Here we'll use the job_script.sh example as template.

The first lines contain the basic setting for our job. Because we'll make use of the GPU instead of CPU, increasing the ppn will have little effect on the run, rather increase the number of GPU (max 4/node).

Next, we switch to the joltik cluster (GPU processing), load all the needed modules and optionally load the plotly module if we want to analyse the output right away.

```
$ module swap cluster/joltik
$ module restore <collection_name>
($ module load plotly.py/5.12.0-GCCcore-11.3.0)
```

Finally, we add our scripts. The '@' is used to indicate a configuration file rather than an argument. Additional arguments can be added using in the standard fashion (--<parameter>). Added parameters already present in the configuration file will be overwritten.

```
$ python MDsim.py '@name_config.txt'
$ python analyse -p output_protein.pdb -t output_traj.dcd -o output_2 -r -
n 4
```

1.3 Submitting the job to HPC

The only left is to submit our job and wait for it to finish. When you job is finished you will receive an email.

```
$ qsub job_script.sh
$ exit
```

Note: the $VSC_SCRATCH storage is not really suited for long term storage of large files, so it is better to move the output files to the $VSC_DATA, once the download is finished.

1.4 Analysis job on the HPC

When running a large amount of simulation, it can be bothersome to run the analysis script for each one separately. Therefore, we can add these to our job script and run everything together.

- Frist, add the analyse.py file to the $VSC_SCRATCH
- Open the job script and add

  ```
  module load plotly.py/5.12.0-GCCcore-11.3.0
  ```
  AFTER the restore collection
- Now add the analysis parameters (see2.5) to script
- Upon completion a html-file will be stored in the output directory

2. Using a desktop

2.1 Setting up a local environment.

The openMM is written in the python language, we thus need a python interpreter to run our program. Anaconda is the most used for scientific purposes, with both a full and minimal version available. Either of these work for presented script. For more information visit:

https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html

Additionally, this will allow us to access the conda-forge repository, containing a lot of free open-source packages.

2.1.1 Installation of OpenMM

First, we need to install openMM & openmmforcefields & openff toolkit

IMPORTANT! The openmmforcefields and openff toolkit depend on the Ambertools package, this only works op MAC-IOS and UNIX operating systems and **NOT ON WINDOWS**. The latest versions of WIN10 and all versions of WIN11 can run UNIX alongside window via WSL. Installation of WSL is explained here:

https://learn.microsoft.com/en-us/windows/wsl/install

Now we have suitable environment, we can install the required packages:

```
$ conda install -c conda-forge openmm
$ conda install -c conda-forge openmmforcefields
$ conda install -c conda-forge openff-toolkit
```

Note: it is possible you'll need to add the conda-force channel to conda. This will tell conda to also look at this repository when installing packages. This can easily be done by running the following line of code:

```
$ conda env config --append channels conda-forge
```

- Test the installation of openMM and check the OpenCl and CUDA requirements for GPU accelerated processing.

```
$ python -m openmm.testInstallation
```

2.1.2 GPU acceleration

Devices equipped with a NVIDEA GPU, can make use of CUDA to run simulations on the GPU rather than the CPU. This is generally considered to a bit faster, since it allows parallel processing. Luckily, installation is very easy. More information can be found at:

https://docs.nvidia.com/cuda/wsl-user-guide/index.html

- Check the model of you NVIDEA GPU

- Download the latest drivers (https://www.nvidia.com/Download/index.aspx)

- That's it!

NOTE Since we do not intend to develop any new CUDA applications, there is no need to install the CUDA Toolkit!

2.2 Preparing the input files

Now our local environment is working, we can start preparing the necessary files for our simulation run.

2.2.1 Preparing the protein

Protein structures found on the PDB seldom include H-atoms and often complete. Therefore, we need to add the H-toms and any missing terminal groups before simulating. Luckly, the MDsim.py script can do this for us. Alternatively, we can use the openmm-setup application (https://github.com/openmm/openmm-setup).

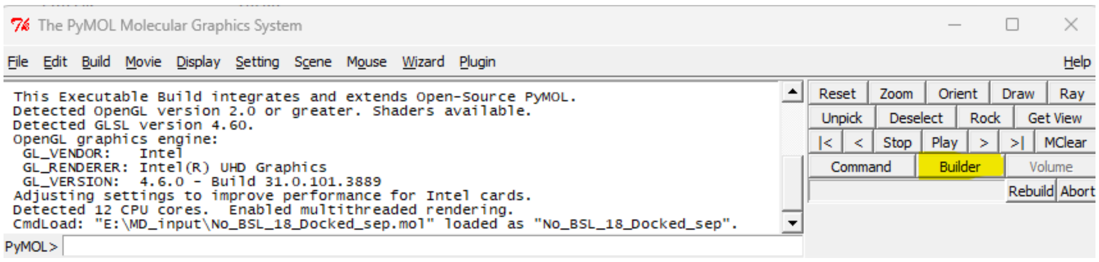
2.2.2 Preparing the ligand

The ligand requires a bit more work than the protein. The ligand must be docked in the protein binding site in advance. For detailed instructions to prepare and run a docking with AutoDock Vina, please watch the video at:

https://vina.scripps.edu/tutorial/

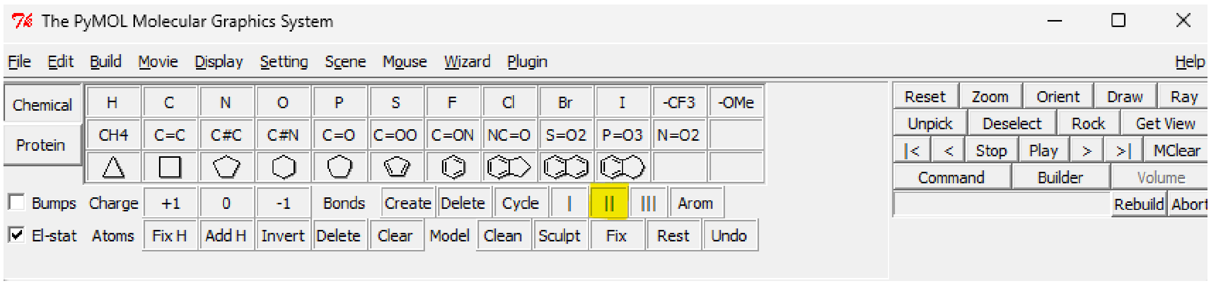
The .pdbqt file removes all double bonds, so now we first need to re-add these to the ligand.

- Open PyMOL and load the ligand.pdbqt file (not all versions PyMOL support .pdbqt files, so may need to convert it to a .pdb file first.).
- Select the ‘Builder’ tab in the toolbar, and make sure the line representation is displayed.


The PyMOL Molecular Graphics System
File Edit Build Movie Display Setting Scene Mouse Wizard Plugin Help
Reset Zoom Orient Draw Ray
Unpick Deselect Rock Get View
|< < Stop Play > >| MClear
Command Builder Volume
Rebuild Abort

```
This Executable Build integrates and extends Open-Source PyMOL.
Detected OpenGL version 2.0 or greater. Shaders available.
Detected GLSL version 4.60.
OpenGL graphics engine:
 GL_VENDOR:   Intel
 GL_RENDERER: Intel(R) UHD Graphics
 GL_VERSION:  4.6.0 - Build 31.0.101.3889
Adjusting settings to improve performance for Intel cards.
Detected 12 CPU cores.  Enabled multithreaded rendering.
CmdLoad: "E:\MD_input\No_BSL_18_Docked_sep.mol" loaded as "No_BSL_18_Docked_sep".
PyMOL>
```

- Now you may notice that the double bounds are gone. We will re-add these manually. Select the double bound tool and click on the bonds you want to double


The PyMOL Molecular Graphics System
File Edit Build Movie Display Setting Scene Mouse Wizard Plugin Help
Chemical
Protein
H C N O P S F Cl Br I -CF3 -OMe
CH4 C=C C#C C#N C=O C=OO C=ON NC=O S=O2 P=O3 N=O2
Bumps Charge +1 0 -1 Bonds Create Delete Cycle | || ||| Arom
El-stat Atoms Fix H Add H Invert Delete Clear Model Clean Sculpt Fix Rest Undo
Reset Zoom Orient Draw Ray
Unpick Deselect Rock Get View
|< < Stop Play > >| MClear
Command Builder Volume
Rebuild Abort

- Save the molecule as **.mol** file.

2.2.3 Preparing the configuration file

With the protein, and optional ligand file, ready, we can start setting up the rest of the simulation parameters. The easiest way to do this is by making a configuration file using MDsim-setup. This can be run as .exe on Windows devices or as script in a linux enviroment.

- Add the MDsim-setup.py file to the working director

- install the wxPython package and launch the MDsim-setup

```
$ conda install wxpython
$ python MDsim-setup
```

A window should now appear where you can input the needed parameters. When everything is filled in correct, click OK and name you file. This name will be used for both the configuration file and the output generated by MDsim.

Finally, we run our script. The ‘@’ is used to indicate a configuration file rather than an argument. Additional arguments can be added using in the standard fashion (--<parameter>). Added parameters already present in the configuration file will be overwritten.

```
$ python MDsim.py ‘@name_config.txt’
$ python MDsim.py ‘@name_config.txt’ -o output_2
```

3. Selecting parameters

The parameters needed will depend on the kind of simulation you want run. I highly recommend using the MDsim-setup, this provides an easy and clear way to select the needed parameters. The section below serves as global overview.

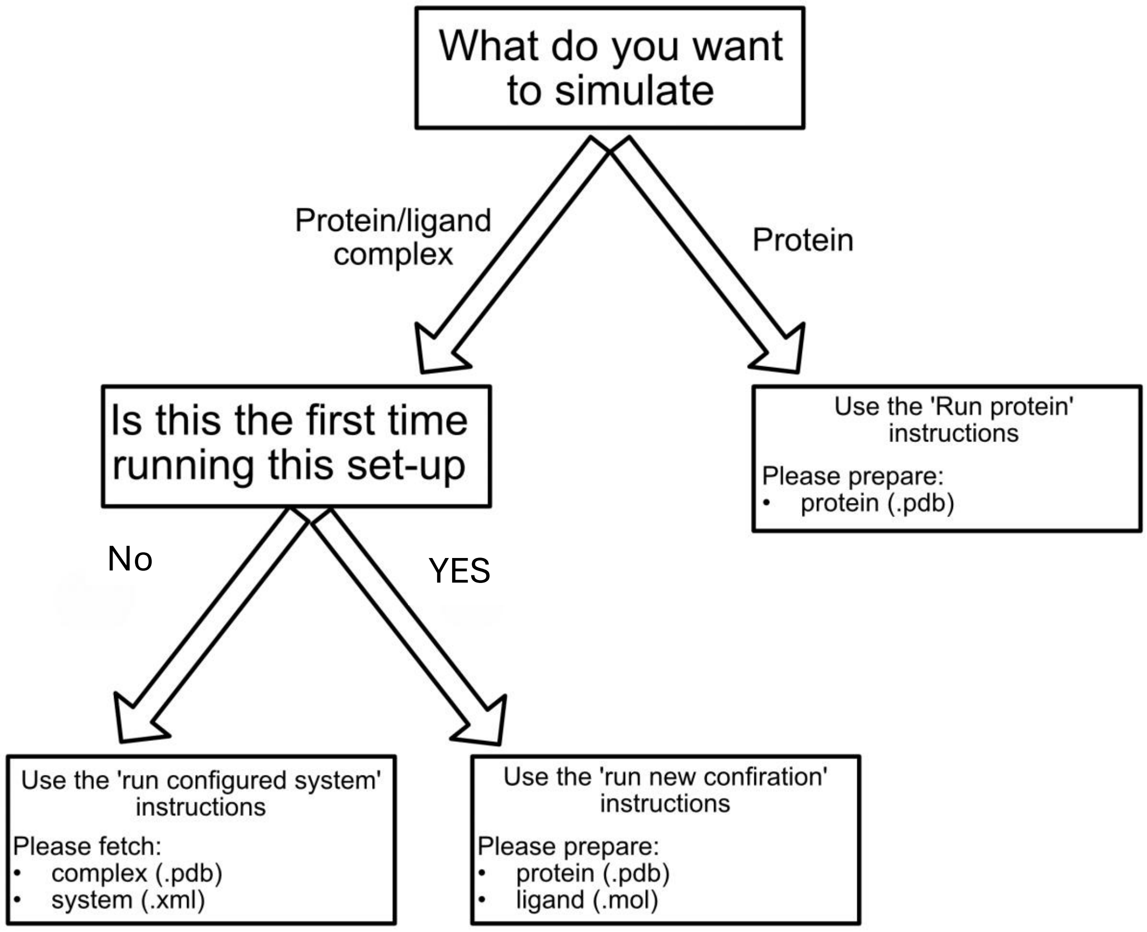

What do you want to simulate
Protein/ligand complex
Protein
Is this the first time running this set-up
Use the 'Run protein' instructions
Please prepare:
• protein (.pdb)
No
YES
Use the 'run configured system' instructions
Please fetch:
• complex (.pdb)
• system (.xml)
Use the 'run new confiration' instructions
Please prepare:
• protein (.pdb)
• ligand (.mol)

Figure 1 Flowchart of simulation options with MDsim.py

Running a new configuration

```
$ Python MDsim.py -a Complex -p protein.pbd -l ligand.mol -o output
```

Here we will build a completely new confirmation starting from a protein and (docked) ligand.

Please provide:

Short ID	Long ID	Provide argument
-a	--simulation	Complex
-p	--protein	Protein .pbd file
-l	--ligand	Ligand mol file
-o	--output	Name the output files

Other parameters can be assigned based on the specific requirements of the simulation (see below).

The script will generate the following output filles:

File name	Description
Output_complex.pdb	The protein/ligand complex
Output_minimised.pdb	The protein/ligand complex with minimised energies
Output_system.xml	The simulation system, can be used to reduce the time needed to run the same complex several times (e.g. at different time intervals)
Output_traj.dcd	The different states of the complex

* with output replaced by the name provided in -–output

Running an already configured configuration

If you have previously already ran a simulation, you can use the outputted complex and system files. This way you can save some time by bypassing the initial set-up of the complex and system.

```
$ Python MDsim.py -a Complex -c output_complex.pdb -s output_system.xml -o output
```

Please provide:

Short ID	Long ID	Provide argument
-a	--simulation	Complex
-c	--complex	Complex .pbd file (obtained during previous run)
-s	--system	System .xml file, (obtained during previous run)
-o	--output	Name the output files

Other parameters can be assigned based on the specific requirements of the simulation (see below).

The script will generate the following output filles:

File name	Description
Output_complex.pdb	The protein/ligand complex
Output_minimised.pdb	The protein/ligand complex with minimised energies
Output_traj.dcd	The different states of the complex

* with output replaced by the name provided in -–output

Running a protein alone

If you only want to simulate a protein, without any ligand use the following instructions.

```
$ Python MDsim.py -a Protein -p protein.pdb -o output
```

Please provide:

Short ID	Long ID	Provide argument
-a	--simulation	Protein
-p	--protein	Protein pbd file
-o	--output	Name the output files

Other parameters can be assigned based on the specific requirements of the simulation (see below).

The script will generate the following output filles:

File name	Description
Output_protein.pdb	The protein
Output_minimised.pdb	The protein with minimised energies
Output_traj.dcd	The different states of the complex

* with output replaced by the name provided in –output

Table 1 Overview of all MDsim.py input parameters.

Short ID	**Long ID**	**Default**	**Description**
Required			
-a	--simulation		What simulation do you wat to run
-p	--protein		Input protein pdb file
-l	--ligand		Input ligand mol file
-c	--complex		Preconfigured complex (pbd)*
-s	--system		Preconfigured system (xml)*
-o	--output		Name for output files
Optional			
-e	--steps	5000	Number of simulation steps (steps*stepsize = duration (ps))
-z	--step-size	0.002	Time of between step (steps*stepsize = duration (ps))
-f	--friction-coeff	1	
-i	--interval	1000	Reporting interval
-t	--temperature	300	Temperature in Kelvin
	--cleanup-protein	False	Clean-up the simulation protein
	--solvate	False	Add solvent box**
	--padding	10	Padding of the solvent box (Å)***
	--water-model	tip3p	Water model used for solvating
	--positive-ion	Na+	Positive ions for solvation
	--negative-ion	Cl-	Negatives ions for solvation
	--ionic-strength	0	Ionic strength for solvation
	--pH	7	pH of simulation environment when adding H-atoms during cleanup
	--no-neutralize	False	Don't add ions to neutralise**
	--equilibration-steps	200	Number of equilibration steps
-pf	--protein-force-field	amber/ff14SB.xml	Protein force field
-lf	--ligand-force-field	gaff-2.11	Ligand force field
-wf	--water-force-field	amber/tip3p-standard.xml	Water force field
	--restrain	None	Restrain atoms using a harmonic force
	--restrain-strength	200	Strength of the harmonic force
	--custom-bond	None	Create a bond between given atoms
	--bond-strength	259407	Strength of the created custom bond

* Only required when no protein and ligand are supplied.
** These don't take an argument, so —-solvate/--no-neutralize is sufficient!
*** The solvent box is a cube of (radius protein sphere + padding)

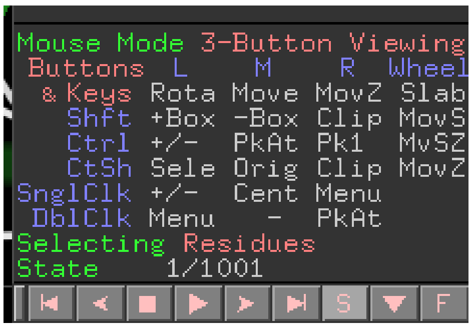
4. Analysing the output

Upon completion we will have generated 4 output-files: output_complex.pdb, output_system.xml, output_minimized.pdb and output_traj.dcd. The system and minimized files can be used to run again as configured configuration. The complex and traj file contain the actual simulation information.

- Open PyMOL and load the output_complex.pdb file

- Load the trajectory using the load_traj command
Note: make sure to load the trajectory with the same name as the load complex. This can be done using:
> load_traj output_traj.dcd, output_complex

- Now the different time points are loaded as states in PyMOL, visible in the bottom right.



```
Mouse Mode 3-Button Viewing
 Buttons  L    M    R  Wheel
  & Keys Rota Move MovZ Slab
    Shft +Box -Box Clip MovS
    Ctrl +/-  PkAt Pk1  MvSZ
    CtSh Sele Orig Clip MovZ
SnglClk +/-  Cent Menu
 DblClk Menu  -   PkAt
Selecting Residues
State      1/1001
```

- Now look at the protein in each state, you will notice it tents to move jump around. If a ligand was docked, this may even leave the docked position. This is because of an internal position shift if the protein or ligand left the solvent box, but has no effect on the simulation itself. To resolve this issue will load the complex and trajectory in another python script.

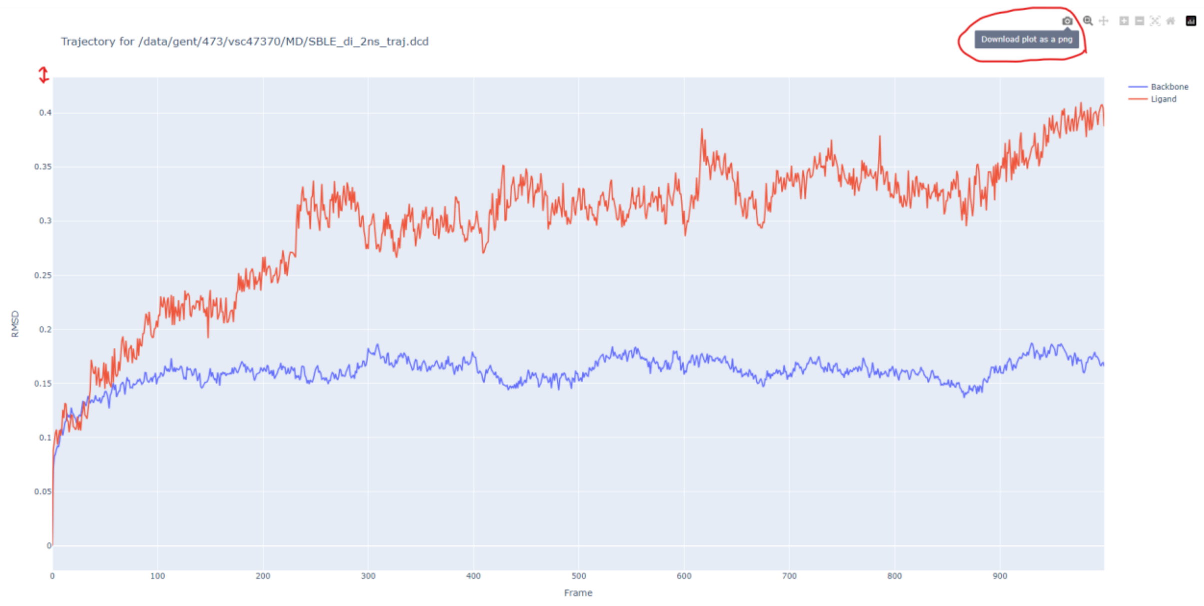
- Return to the terminal and run

```
$ Python analyse.py -p output_complex.pdb -t output_traj.dcd -o output -r
-n 4
```

Table 2 Overview of analyse.py input and output.

Short ID	**Long ID**	**Provide argument**
Input		
-p	--protein	Complex/protein pbd file
-t	-- trajectory	Trajectory dcd file
-o	--output	Name the output files
-r	--remove-waters	Remove waters, salts, ect (optional)
-n	--renumber	Renumber the protein (give the desired first number)
Output		
	File name	**Description**
	Output.pdb	File with atom positions
	Output.dcd	File with different states
	Output.svg	Graph of the RMSD

- When done, a new browser tab will open with the RMSD graph. This plot the RMSD of the ligand and protein backbone of each frame to the initial input conformation (frame 0). On the HPC, a HTML file will be outputted instead of opening a browser tab.



- When needed the scale can be modified by dragging the top up or down (red arrow)
- It is advised to download this graph, as it is more informative then the .svg. This is done in top right corner.
  Note: the html file can also be safe, these are however quit large and thus not advised if not necessary .
- Now again load the pdb and dcd files in PyMOL.
  Note: first load the pdb and afterwards the dcd with load_traj
- You have now successfully made a MD simulation. Congratulations!

4.1 Finding ligand interacting residues

- Separate the ligand from the ligand by selecting the ligand sele -> A -> extract object

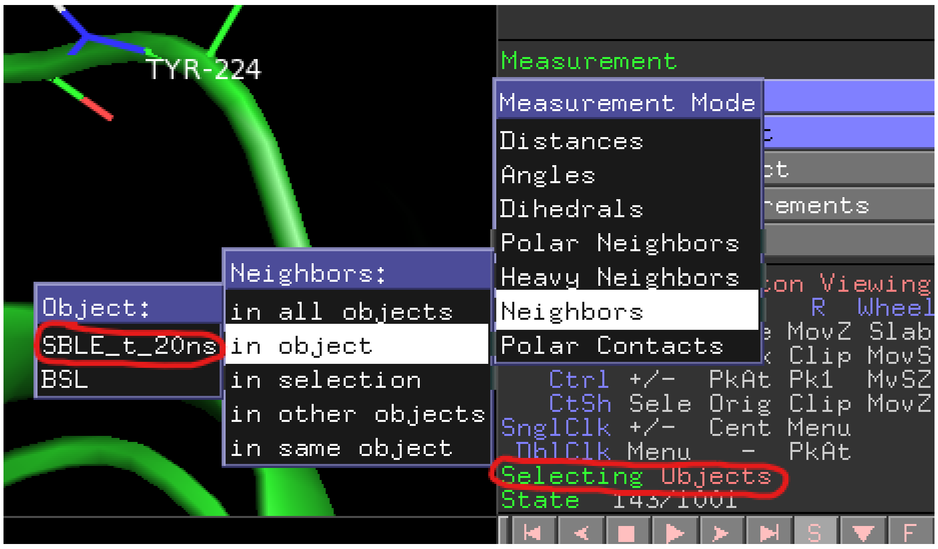
- Rename the obj01 to <name ligand> (e.g. BSL)

Now we can indicate the interactions

- Wizard -> measurement

- change distance to neighbour -> in object -> <name protein>

- Change the selection to ‘object’ and select the ligand



Now yellow dotted lines appear, making the interaction between the residues and the ligand. Now we'll show the interacting ligands

- type in the consol:

> select <name of new selection>, <name of ligand> around <distance in Å>

e.g. > select Neighbours, BSL around 4

- Now show them and display the identifiers

S -> line/sticks

L -> residues

Note: PyMOL is also available on the VSC-HPC.

5. Fantastic errors and how to solve them

3.1 Undefined chiral centres

The .pdbqt format removes all double bounds. This can create chiral centra at place where theren't any before. To solve the problem, you'll need to re-add the double bound again manually.

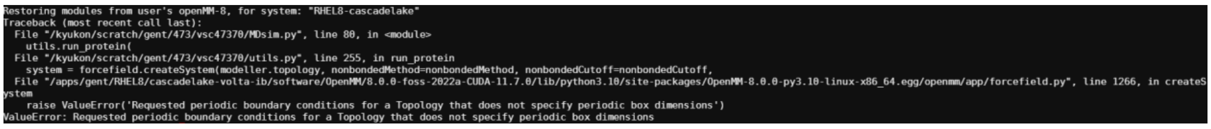
3.2 Endless export of RSMD plot when analysing

Make sure you have version '0.1.0post1' of the *kaleido* package installed. Higher versions tend to get stuck in an endless loop when trying to export the plot.

3.3 No template found for res X (…). Closest match is …, but X atoms are missing/ bonds are different.

As it says, the residue does not match the template. This can because there are either to extra or missing atoms. You can manually inspect the structure and remove/add the needed atoms or re-try the protein preparation with openmm-setup. If both fail another preparation script is available at https://github.com/tdudgeon/simple-simulate-complex?tab=readme-ov-file (prepareProtein.py).

3.4 No periodic box defined



```
Restoring modules from user's openMM-8, for system: "RHEL8-cascadelake"
Traceback (most recent call last):
  File "/kyukon/scratch/gent/473/vsc47370/MDsim.py", line 80, in <module>
    utils.run_protein(
  File "/kyukon/scratch/gent/473/vsc47370/utils.py", line 255, in run_protein
    system = forcefield.createSystem(modeller.topology, nonbondedMethod=nonbondedMethod, nonbondedCutoff=nonbondedCutoff,
  File "/apps/gent/RHEL8/cascadelake-volta-ib/software/OpenMM/8.0.0-foss-2022a-CUDA-11.7.0/lib/python3.10/site-packages/OpenMM-8.0.0-py3.10-linux-x86_64.egg/openmm/app/forcefield.py", line 1266, in createS
ystem
    raise ValueError('Requested periodic boundary conditions for a Topology that does not specify periodic box dimensions')
ValueError: Requested periodic boundary conditions for a Topology that does not specify periodic box dimensions
```

When running protein simulations, make sure you solvate your system. This can be done via the openmm-setup or include –solvate as parameter.