INTRODUCTION

1 About 3
2 Contributors 5
3 Features 7
  3.1 Parallel Scalability 7
  3.2 Applications 8
4 Citation 11
5 MiMiC Framework 15
  5.1 Requirements 15
  5.2 Build and Installation 16
6 External Programs 21
  6.1 CPMD 21
  6.2 GROMACS 23
7 MiMiCPy 25
  7.1 Requirements 25
  7.2 Basic Installation 25
  7.3 Python Virtual Environments 26
  7.4 Conda Environments 27
8 First Steps: Acetone in Water 29
  8.1 System Preparation 30
  8.2 Classical Equilibration 38
  8.3 QM/MM MD with MiMiC 45
9 MiMiCPy Handbook 65
  9.1 Example System: Acetone in Water 66
  9.2 PrepQM 67
  9.3 FixTop 77
  9.4 CPMDid 80
  9.5 Debugging with CPMD2Coords and Geom2Coords 84
  9.6 As a Python Library 86
10 Frequently Asked Questions (FAQ) 89
11 How to Get Help 91
  11.1 Where should you post? 91
MiMiC is a framework for multiscale modeling in computational chemistry. The aim is to enable flexible and efficient implementations of multiscale simulation methods with support for multiple subsystems through coupling external programs.

The MiMiC framework consists of the MiMiC library and the MiMiC communication library. Additionally, the MiMiCPy package can be used to assist in input preparation. MiMiC-based QM/MM runs efficiently in parallel using a combination of MPI and OpenMP parallelization. It is freely available under the LGPLv3+ license.

Getting started:

1. Install MiMiC framework
2. Patch and install CPMD and GROMACS
3. Install MiMiCPy
4. Follow the introductory tutorial
5. Get help and support
6. Join our discussion group to receive news and updates

Current releases:

- MiMiC Library v0.2.0
- MiMiC Communication Library v2.0.2
- MiMiCPy v0.2.1

Get the source at: [GitLab](https://gitlab.com)
MiMiC is a framework for the development of multiscale models and for running multiscale simulations related to computational chemistry. The goal is to enable flexible and efficient implementations of multiscale models with support for multiple subsystems that each can be described at different resolutions and levels of theory.

Spatial resolutions may range from a quantum mechanical description of electrons, over atomistic molecular mechanics and coarse-grained modeling, to continuum models. However, MiMiC itself does not calculate any subsystem contributions (e.g., energy and forces) on its own except potentially for contributions arising from the interactions between subsystems. Instead, MiMiC relies on external programs using a multiple-program multiple-data (MPMD) model with loose coupling between programs. This allows high flexibility while at the same time retaining a high degree of computational efficiency by letting the external programs run concurrently and by exploiting the fact that different programs have been optimized for their own specific use case.

Specifically, MiMiC couples a main driver (which runs a molecular dynamics simulation or performs some other type of calculation) to a set of external programs, each of which concurrently computes the contributions that are relevant to a specific subsystem using their own optimal parallelization strategy. Crucial for this strategy is the efficient communication between programs which is achieved through the use of a lightweight communication library (MiMiC communication library aka MCL), which was developed specifically for this purpose. MCL comes with a simple C and Fortran API that avoids major intervention in the source code of the external programs. The underlying data transfer uses MPI, which gives immediate access to high-speed interconnects such as InfiniBand, but it can be seamlessly expanded to other communication mechanisms without altering the API.

For more details about the MiMiC framework, and the theory and performance of a MiMiC-based QM/MM implementation, we refer to the following two articles:


The MiMiC framework is developed mainly by the groups of J. M. H. Olsen, P. Carloni, and U. Röthlisberger. The initial idea for MiMiC came about in a meeting between J. M. H. Olsen, S. Meloni, and U. Röthlisberger at EPFL in early 2015. The MiMiC concept was further developed in 2015 by J. M. H. Olsen with contributions from S. Meloni and U. Röthlisberger.

The plans for MiMiC were presented to T. Laino and A. Curioni from IBM Research Europe and V. Bolnykh, E. Ippoliti, and P. Carloni from Forschungszentrum Jülich in a meeting hosted by IBM Research Europe in Zürich on November 25th, 2015. Shortly after, E. Ippoliti and V. Bolnykh visited EPFL, where the work on the MiMiC framework started. During the visit, the implementation was designed by J. M. H. Olsen, V. Bolnykh, and S. Meloni, with contributions from E. Ippoliti and U. Röthlisberger. The work on the MiMiC framework continued as a collaboration from that time.

J. M. H. Olsen and V. Bolnykh wrote the MiMiC library, and V. Bolnykh wrote the MiMiC communication library (MCL). J. M. H. Olsen and V. Bolnykh wrote the CPMD interface, and V. Bolnykh wrote the GROMACS interface.

The following is a list of the main authors and other contributors including their affiliation(s) at the time of their contribution.

Main authors:

- Jógvan Magnus Haugaard Olsen
  - Technical University of Denmark
  - UiT The Arctic University of Norway
  - École Polytechnique Fédérale de Lausanne
- Viacheslav Bolnykh
  - Forschungszentrum Jülich
  - RWTH Aachen University
  - The Cyprus Institute

Other contributors:

- Simone Meloni
  - University of Ferrara
  - École Polytechnique Fédérale de Lausanne
- Ursula Röthlisberger
  - École Polytechnique Fédérale de Lausanne
- Emiliano Ippoliti
  - Forschungszentrum Jülich
• Paolo Carloni  
  – Forschungszentrum Jülich  
  – RWTH Aachen University  
• Martin P. Bircher  
  – École Polytechnique Fédérale de Lausanne  
• Till Kirsch  
  – Johannes Gutenberg-Universität Mainz  
• Bharath Raghavan  
  – Forschungszentrum Jülich  
  – RWTH Aachen University  
• Florian Schackert  
  – Forschungszentrum Jülich  
  – RWTH Aachen University  
• Nitin Malapally  
  – Forschungszentrum Jülich  
• Davide Mandelli  
  – Forschungszentrum Jülich  
• Andrea Levy  
  – École Polytechnique Fédérale de Lausanne  
• Sophia Johnson  
  – École Polytechnique Fédérale de Lausanne  
• Andrej Antalik  
  – École Polytechnique Fédérale de Lausanne  
• François Mouvet  
  – École Polytechnique Fédérale de Lausanne  
• Sonata Kvedaravičiūtė  
  – Technical University of Denmark  
• David Carrasco de Busturia  
  – Technical University of Denmark
MiMiC currently supports using CPMD as the main MD driver and for calculating subsystem contributions it has interfaces to CPMD and GROMACS. Using the currently supported programs it is possible to run electrostatic-embedding QM/MM MD simulations where CPMD can be used as the QM engine and GROMACS as the MM engine. MiMiC also supports using the PLUMED library for performing enhanced-sampling QM/MM MD simulations.

### 3.1 Parallel Scalability

The MiMiC-based QM/MM implementation has displayed strong scalability well beyond ten thousand cores in a single QM/MM simulation while maintaining an overall parallel efficiency of at least 70%.

![Scaling performance graph](image)

Fig. 1: Scaling performance within Born-Oppenheimer molecular dynamics for a system containing a large Cl/H+ antiporter protein embedded in a lipid membrane bilayer solvated in water. In this system, 19 atoms out of a total of 150,925 atoms were treated at the B3LYP QM level. Simulations were run on the JUWELS cluster at the Juelich Supercomputing Center.
3.2 Applications

MiMiC-based QM/MM MD simulations combined with metadynamics have been successfully applied to investigate complex biochemical processes. Below are some examples.

3.2.1 Molecular basis of CLC antiporter inhibition by fluoride

CLC channels and transporters conduct or transport various kinds of anions, with the exception of fluoride, which acts as an effective inhibitor. This study identified the high affinity of both F and E148 for protons as the basis of the transport inhibition of the CLC anion/proton exchangers from E. coli.  

Sub-nanosecond QM/MM simulations of the E. coli anion/proton exchanger ClC-ec1 showed that fluoride binds incoming protons within the selectivity filter, with excess protons shared with the gating glutamate E148. Depending on E148 conformation, the competition for the proton can involve either a direct F/E148 interaction or the modulation of water molecules bridging the two anions. The direct interaction locks E148 in a conformation that does not allow for proton transport, and thus inhibits protein function.

3.2.2 Mechanisms underlying proton release in CLC-type F/H+ antiporters

The CLC family of anion channels and transporters includes Cl/H+ exchangers (blocked by F) and F/H+ exchangers (or CLCFs). CLCFs contain a glutamate (E318) in the central anion-binding site that is absent in CLC Cl/H+ exchangers. The X-ray structure of the protein from Enterococcus casseliiflavus (CLCF-eca) shows that E318 tightly binds to F when the gating glutamate (E118; highly conserved in the CLC family) faces the extracellular medium. This study illustrated how glutamate insertion into the central anion-binding site of CLCF-eca permits the release of H+ to the cytosol as HF, thus enabling a net 1:1 F/H+ stoichiometry.

Classical and MiMiC-based QM/MM metadynamics simulations were used to investigate proton transfer and release by CLCF-eca. Results show that after up to down movement of protonated E118, both glutamates combine with F to form a triad, from which protons and F anions are released as HF.

Fig. 2: E. coli CLC antiporter (CLC-ec1) in a lipid bilayer in the simulation setup. Water and counterions are not shown for clarity.
Fig. 3: Free energy profiles (in kcal/mol) emerging from QM/MM metadynamics simulations. (A) Free energy associated with the proton transfer (PT) process between fluoride and E148 modulated by two water molecules. The schematic representation of the two free energy minima is displayed on the top of the figure. Here, E148 is in the up conformation. (B) Free energy profile associated with the direct PT process between F and E148 in the down conformation. The minimum is reached at HF and HO (E148) distances of 1.5 and 1.0 Å, respectively. No barrier separates the state in which the proton is bound to the fluoride (HF 1.0 Å). (C, D) Free energy as a function of E148’s $\chi_1$ dihedral angle for E148 protonated (C) and deprotonated (D). The relative positions of the E148 carboxyl group above or below the backbone unit define the up or down conformations. The transition from one conformation to the other is indicated by dashed red lines.
Fig. 4: E. casseli flavus F/H+ antiporter (CLCF-eca) embedded in a lipid bilayer in the simulation setup. Water and counterions are not shown for clarity. The inset show HF interacting with the two glutamates E118 and E338.

Fig. 5: (A) Free energy landscape of conversion of the E318FE118 triad into the intermediate conformation with E318 and E118 in direct contact. (B) Free energy profile of HF release into the intracellular solution. The E318 carboxyl oxygen atoms are labeled O1 and O2. H-bonds are shown as red dashed lines and free energies are in kcal/mol.
Please cite these papers if you used MiMiC in your work:


And this paper if you used MiMiCPy:


In addition, in the interest of reproducibility, we ask that you also cite the following software (remembering to update the version number and DOI according to the version that was used):


- B. Raghavan, and F. K. Schackert, *MiMiCPy (v0.2.1)*, GitLab, 2023. DOI: 10.5281/zenodo.7746282. See https://mimic-project.org/.

For your convenience here are BibTeX entries that you can use:

```latex
@article{mimic-1,
  title = {{MiMiC: A Novel Framework for Multiscale Modeling in Computational Chemistry}},
  author = {Olsen, J{o}rgvan Magnus Haugaard and Bolnykh, Viacheslav and Meloni, Simone and Ippoliti, Emiliano and Bircher, Martin P. and Carloni, Paolo and Rothlisberger, Ursula},
  journal = {J. Chem. Theory Comput.},
  volume = {15},
  number = {6},
  pages = {3810–3823},
  month = {jun},
  year = {2019},
  url = {https://doi.org/10.1021/acs.jctc.9b00093},
  doi = {10.1021/acs.jctc.9b00093}
}
```

(continues on next page)
@article{mimic-2,
  title = {{Extreme Scalability of DFT-Based QM/MM MD Simulations Using MiMiC}},
  author = {Bolnykh, Viacheslav and Olsen, J{o}gvan Magnus Haugaard and
            Meloni, Simone and Bircher, Martin P. and Ippoliti, Emiliano and
            Carloni, Paolo and Rothlisberger, Ursula},
  journal = {J. Chem. Theory Comput.},
  volume = {15},
  number = {10},
  pages = {5601-5613},
  month = {oct},
  year = {2019},
  url = {https://doi.org/10.1021/acs.jctc.9b00424},
  doi = {10.1021/acs.jctc.9b00424}
}

@article{mimicpy,
  title = {{MiMiCPy: An Efficient Toolkit for MiMiC-Based QM/MM Simulations}},
  author = {Raghavan, Bharath and Schackert, Florian K. and Levy, Andrea and
            Johnson, Sophia K. and Ippoliti, Emilliano and Mandelli, Davide and
            Olsen, J{o}gvan Magnus Haugaard and Rothlisberger, Ursula and
            Carloni, Paolo},
  journal = {J. Chem. Inf. Model.},
  volume = {63},
  number = {5},
  pages = {1406-1412},
  month = {mar},
  year = {2023},
  url = {https://doi.org/10.1021/acs.jcim.2c01620},
  doi = {10.1021/acs.jcim.2c01620}
}

@misc{mimic:0.2.0,
  author = {Olsen, J{o}gvan Magnus Haugaard and Bolnykh, Viacheslav and
            Meloni, Simone and Ippoliti, Emiliano and Carloni, Paolo and
            Rothlisberger, Ursula},
  title = {{MiMiC: A Framework for Multiscale Modeling in Computational Chemistry (v0.2.0)}},
  publisher = {GitLab},
  year = {2022},
  doi = {10.5281/zenodo.7304688},
  note = {See https://mimic-project.org/}
}

@misc{mcl:2.0.1,
  author = {Bolnykh, Viacheslav and Olsen, J{o}gvan Magnus Haugaard and
            Meloni, Simone and Ippoliti, Emiliano and Carloni, Paolo and
            Rothlisberger, Ursula},
  title = {{MiMiC Communication Library (v2.0.1)}},
  publisher = {GitLab},
  year = {2022},
  (continues on next page)
doi = {10.5281/zenodo.5024023},
note = {See https://mimic-project.org/}
}

@misc{mimicpy:0.2.1,
author = {Raghavan, Bharath and Schackert, Florian K.},
title = {{MiMiCPy (v0.2.1)}},
publisher = {GitLab},
year = {2023},
doi = {10.5281/zenodo.7746282},
note = {See https://mimic-project.org/}
}
The following installation guide provides quick and concise instructions to get MiMiC running as quickly as possible. The MiMiC framework consists of two components, namely the MiMiC library and the MiMiC communication library (MCL). In the following, we provide the requirements for both components and the instructions for building and installing each component.

Along with the MiMiC framework, we also provide the utility tool MiMiCPy, which is a Python-based collection of tools that automates the setup of MiMiC-based simulations. See MiMiCPy for installation instructions.

Throughout this guide, we primarily use Git to obtain the MiMiC framework sources, but we also provide optional download links. In case you opt for the latter option, just note that the directory names extracted from downloaded files might differ.

### 5.1 Requirements

A list of minimum requirements to build the MiMiC framework is given below. Optional dependencies and supported external programs are also listed.

#### 5.1.1 Minimum Requirements

- CMake 3.12
- Fortran compiler (Fortran 2008 standard)
- C++ compiler (C++17 standard)
- MPI library (MPI 2.1 standard)

#### 5.1.2 Optional Dependencies

- Shared-memory parallelism: Fortran compiler with OpenMP 3.0 support
- Building unit tests: pFUnit (MiMiC) and GoogleTest and GoogleMock (MCL)
5.1.3 Supported External Programs

See *External Programs* for installation instructions.

- CPMD 4.3 (patch required)
- GROMACS 2021 (patch required)
- PLUMED 2 (through CPMD)

5.2 Build and Installation

The two components of the MiMiC framework are configured, built, tested, and installed using the CMake build system. The following instructions will guide you through the process from download to installation.

Before we start, it is important to decide where MiMiC should be installed. In this guide, we will use `.local` in user's home directory, but feel free to choose whichever location you prefer:

```
INSTALL_DIR=${HOME}/.local
```

In the following, we will refer to this variable to set `CMAKE_INSTALL_PREFIX` which determines the root of install directory. This means that after invoking `make install`, everything will be installed in this directory using standard GNU locations, i.e., `lib` (or `lib64`), `include`, `share`, etc.

Moreover, to be consistent with the compilers used for the different programs you are going to build, you can set the environment variables `FC`, `CC`, and `CXX` to your chosen Fortran, C, and C++ compilers, respectively. These are then detected by CMake during the build configuration process.

Finally, we recommend that you download the sources in a separate directory. Here we will create and use a directory in the user's home directory, but again feel free to use whichever location you prefer:

```
SOURCE_DIR=${HOME}/mimic_sources
mkdir -p ${SOURCE_DIR}
```

### Building with tests

We strongly recommend to build with tests if you are planning to use the builds for production simulations. Note that some tests require either `mpirun` or `mpiexec` which may not be available on all HPC clusters that may instead use, e.g., `srun`. If neither `mpirun` nor `mpiexec` is available, then those tests may fail, but that does not necessarily mean that the build will not work. If you are in doubt consult the *Help section*.

**Warning:** Instructions in this guide are written for releases of MiMiC and MCL. Note that the development branch may be unstable or contain breaking changes that are not compatible with released versions of supported programs. Therefore we do not recommend that you use it unless you really know what you are doing.
5.2.1 MiMiC Communication Library

It is necessary to build and install MCL first, as the MiMiC library depends on it. Start by changing directory to the common source directory:

```
cd ${SOURCE_DIR}
```

Then start by downloading and unpacking the release version of MCL:

```
wget https://gitlab.com/MiMiC-projects/CommLib/-/archive/2.0.2/CommLib-2.0.2.tar.gz
tar -zxvf CommLib-2.0.2.tar.gz
```

Alternatively, you can use git to clone it directly from the MCL repository. Next, enter the MCL root directory, create a build directory and enter it:

```
cd CommLib-2.0.2
mkdir build
cd build
```

From here, you configure the build using CMake, and then build and install:

```
cmake -DBUILD_SHARED_LIBS=YES -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} ..
make
make install
```

Additionally, you might also want to build and run tests. To do so, you need to add `-DBUILD_TESTS=YES` in the configure step shown above and run `make test` before `make install`. The prerequisite for this is the GoogleTest testing and mocking framework. If you do not have it, you can build it in a few steps. However, this has to be done before you configure MCL.

**Installing GoogleTest**

If you already have a functioning build of GoogleTest just set these environment variables:

```
export GTEST_ROOT=<path to your GTest build>
export GMOCK_ROOT=<path to your GMock build>
```

Otherwise, you can clone the GoogleTest repository, build, and install by following these steps:

```
cd ${SOURCE_DIR}
wget -O googletest-1.12.1.tar.gz https://github.com/google/googletest/archive/refs/tags/release-1.12.1.tar.gz
cd googletest-release-1.12.1
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} ..
make
make install
```

**Failing tests**

Note that tests `mpitransport` and `mpmdtransport` require either `mpirun` or `mpiexec` which may not be available on all HPC clusters that may instead use, e.g., `srun`. If neither `mpirun` nor `mpiexec` is available, then those tests will fail, but that does not necessarily mean that the MCL build will not work. Moreover, not all MPI implementations
support both modes of communication offered by MCL. In our experience, most newer MPI implementations support
MPMD, which is tested by mpmdtransport, while the server-client mechanism, which is tested by mpirtransport, is
less common. There may also be others factors that result in those tests failing even if the MCL build is fully functional.
If you are in doubt consult the Help section.

5.2.2 MiMiC Library

Once we have installed the communication library, we can proceed to the compilation of MiMiC. Move to the source
directory:

```
cd ${SOURCE_DIR}
```

And then you download and unpack the release version of MiMiC:

```
wget https://gitlab.com/MiMiC-projects/MiMiC/-/archive/0.2.0/MiMiC-0.2.0.tar.gz
tar -zxvf MiMiC-0.2.0.tar.gz
```

Alternatively, you can use git to clone the repository. Enter the MiMiC root directory and from there create a build
directory and enter it:

```
cd MiMiC-0.2.0
mkdir build
cd build
```

Then you configure the build by running CMake and then build and finally install:

```
cmake -DENABLE_OPENMP=YES -DBUILD_SHARED_LIBS=YES -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} .
make
make install
```

As with MCL, we recommend that you also build and run the tests suite. This is done in the same manner by adding a
CMake flag -DBUILD_TESTS=YES when configuring and calling make test before the installing. The only difference
is that MiMiC uses pFUnit testing framework, which you can install following the instructions below.

Installing pFUnit

First clone pFUnit repository to the source directory:

```
cd ${SOURCE_DIR}
wget -O pFUnit-4.6.1.tar.gz https://github.com/Goddard-Fortran-Ecosystem/pFUnit/archive/refs/tags/v4.6.1.tar.gz
tar -zxvf pFUnit-4.6.1.tar.gz
```

Enter the directory, create and enter build directory, configure, build, and install:

```
cd pFUnit-4.6.1
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} ..
make tests
make install
```
If you encounter an error related to git submodules, try using a more recent version of git.

---

**Failing tests**

Note that tests `test_main` and `test_data_collect` require either `mpirun` or `mpiexec` which may not be available on all HPC clusters. If neither is available, then those tests will fail, but that does not necessarily mean that the MiMiC build will not work. Both tests also rely on the MPMD communication mechanism and will fail if it is not supported. There may also be others factors that result in those tests failing even if the MiMiC build is fully functional. If you are in doubt consult the *Help section*. 

---

5.2. Build and Installation
Currently, the only way to enable MiMiC in the supported programs comes in the form of patches, which must be applied to the original source codes.

The following instructions mostly follow the compilation of the original software. Therefore, if you come across any problems unrelated to MiMiC, please consult their respective manuals.

Following the instructions for MiMiC, we will install the programs in the `.local` directory in the user’s home, but feel free to choose whichever location you prefer:

```bash
INSTALL_DIR=${HOME}/.local
```

Set the environment variables `FC`, `CC`, and `CXX` to your chosen Fortran, C, and C++ compilers, respectively. We will also use the same directory for the sources as in the instructions for MiMiC, but use whichever location you prefer:

```bash
SOURCE_DIR=${HOME}/mimic_sources
```

### 6.1 CPMD

In your `$SOURCE_DIR` directory, download and unpack CPMD 4.3:

```bash
cd $SOURCE_DIR
wget -O cpmd-4.3.tar.gz https://github.com/CPMD-code/CPMD/archive/refs/tags/4.3.tar.gz
tar -zxvf cpmd-4.3.tar.gz
```

Download the MiMiC patch for CPMD, copy it to the CPMD root directory, and apply the patch:

```bash
cp MiMiC_CPMD-4.3.sh ./CPMD-4.3/
cd CPMD-4.3
bash MiMiC_CPMD-4.3.sh
```

If you want to enable support for PLUMED in CPMD, then follow the instructions in *Enabling support for PLUMED* before continuing.

To make the install path available within CPMD’s configuration script you first have to export the `INSTALL_DIR` variable:

```bash
export INSTALL_DIR
```

Choose a configuration file from the `configure` directory that is most suitable for your target system (you may need to edit it to make it work). Append the following two lines to the selected configuration file:
They should be the last two lines in the configuration file. If you use GCC then add also 
-fallow-argument-mismatch -ffree-line-length-none to FFLAGS.

Attention: Make sure that your selected configuration supports MPI, as this is required by MiMiC. There are no further checks and you would discover the incompatibility during runtime.

Now configure the build using the configure.sh script, switch to the build directory, and compile:

```bash
bash ./configure.sh -DEST=build -omp -mimic <configuration-file-name>
```

```bash
cd build
make
```

Optionally, as a final touch you can copy the CPMD executable to INSTALL_DIR:

```bash
mkdir -p ${INSTALL_DIR}/bin
cp ${SOURCE_DIR}/CPMD-4.3/build/bin/cpmd.x ${INSTALL_DIR}/bin/
```

Pseudopotentials

Pseudopotentials are not distributed with CPMD, so do not forget to download them either from the CPMD website or your favourite pseudopotential repository.

### 6.1.1 Enabling support for PLUMED

To enable support for PLUMED in CPMD, there are a few additional steps. First you need to have PLUMED installed. If you already have a working PLUMED installation, you can skip the PLUMED installation part.

**Note:** We have tested PLUMED v2.6.5, v2.7.4, v2.8.0, and v2.8.1, but it is likely that other versions of PLUMED 2 are also compatible with the CPMD patch provided in the MiMiC tarball.

**Installing PLUMED**

Here we provide a quick installation guide for PLUMED. For more detailed information please consult the official PLUMED documentation.

First move to the source directory, download PLUMED v2.8.1 from the official PLUMED website, and unpack it:

```bash
cd ${SOURCE_DIR}
tar -zxvf plumed-2.8.1.tgz
```

Then enter the PLUMED source directory, configure, build, and install:

```bash
cd plumed-2.8.1
./configure CXX=<c++-mpi-wrapper> --prefix=${INSTALL_DIR}
```

(continues on next page)
Finally, set the following environment variables:

```bash
export PATH=${INSTALL_DIR}/bin:${PATH}
export LIBRARY_PATH=${INSTALL_DIR}/lib:${LIBRARY_PATH}
export LD_LIBRARY_PATH=${INSTALL_DIR}/lib:${LD_LIBRARY_PATH}
export PLUMED_KERNEL=${INSTALL_DIR}/lib/libplumedKernel.so
export PLUMED_VIMPATH=${INSTALL_DIR}/lib/plumed/vim
export PYTHONPATH=${INSTALL_DIR}/lib/plumed/python:${PYTHONPATH}
```

First, ensure that the CPMD source has been patched with the MiMiC patch by following the instructions above. Then download the PLUMED patch for CPMD, copy it to the CPMD root directory, and apply it:

```bash
cp PLUMED_CPMD-4.3.sh ./CPMD-4.3/
cd CPMD-4.3
bash PLUMED_CPMD-4.3.sh
```

Next, use PLUMED to patch CPMD:

```bash
plumed patch -n CPMD-4.3
plumed patch -p -e CPMD-4.3
```

This step creates the Plumed.inc, Plumed.h, and Plumed.cmake files which are necessary to compile CPMD with PLUMED support.

Finally, add the -plumed option in the configuration step.

## 6.2 GROMACS

In your `${SOURCE_DIR}` directory, download and unpack GROMACS 2021.6 from the official GROMACS website:

```bash
cd ${SOURCE_DIR}
tar -zxvf gromacs-2021.6.tar.gz
```

Download the MiMiC patch for GROMACS, copy it to the GROMACS directory, and apply it:

```bash
cp MiMiC_GROMACS-2021.patch gromacs-2021.6/
cd gromacs-2021.6
patch -p1 < MiMiC_GROMACS-2021.patch
```

Create a build directory, enter it, configure the build by running CMake, build, and finally install:

```bash
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DGMX_DOUBLE=ON -DGMX_MPI=ON -DGMX_MIMIC=ON 
  -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} -DCMAKE_PREFIX_PATH=${INSTALL_DIR} ..
make
make install
```
You are free to use any other flags, but those shown here are required for a successful build of MiMiC-enabled GROMACS.
MiMiCPy is the companion utility package for MiMiC, with a collection of tools for preparing and debugging MiMiC input files. It is written in Python and can be installed from the Python Package Index (PyPI) using the package installer for Python (pip). MiMiCPy can be installed directly in the current Python installation or within a virtual environment. The process is described in detail below.

## 7.1 Requirements

The minimum requirements for running MiMiCPy are given below. MiMiCPy also provides plugins for the molecular visualization packages VMD and PyMOL. The plugins can be installed anytime after the installation of MiMiCPy.

### 7.1.1 Minimum Requirements

- Python 3.5+
- Pandas 0.24+
- NumPy 1.12+

### 7.1.2 Optional Dependencies

- VMD
- PyMOL

## 7.2 Basic Installation

To install the latest stable version of MiMiCPy from the PyPI, run the following command:

```bash
$ python -m pip install mimicpy
```

This will also install the required dependencies if they are not already available. Recent versions of *pip* will install packages by default in a local user directory when root access is not available. The path depends on the system, but the base directory defaults to `~/.local` on Linux. Scripts are then placed in `~/.local/bin` which is often already added to the `PATH` environment variable. In case it is not, or you do not know the base directory, run this command to add it to the `PATH` environment variable:

```bash
PATH=$(python -c 'import site; print(site.USER_BASE + "/bin")'):$PATH
```
To check that MiMiCPy has been correctly installed, you can ask MiMiCPy to print the help via the command:

```
$ mimicpy --help
```

After successfully installing MiMiCPy, you can optionally install the PyMOL/VMD plugins by running the plugin installer bundled with MiMiCPy:

```
$ mimicpy_plugin_installer -pymoldir /path/to/plugin/ -vmddir /path/to/plugin/
```

The path to the plugin is usually either the path to the PyMOL/VMD installation, or the user home directory. This command will create (or append to, if it already exists) a .pymolrc.py and/or a .vmdrc (vmd.rc on Windows) file in the given paths (in this case /path/to/plugin/). If these files are in the current directory, the user’s home directory, or the installation directory of the visualization package, they will be read and loaded by PyMOL and VMD on startup. This makes the PrepQM command available to them.

Installing MiMiCPy from the PyPI is the recommended option. However, if you would like to install the most recent version of MiMiCPy under development, clone the source using git (or download it directly):

```
$ python -m pip install git+https://gitlab.com/MiMiC-projects/MiMiCPy.git
```

## 7.3 Python Virtual Environments

As mentioned in the previous section, pip will often add install packages locally. In these cases, the MiMiCPy executables may not be available without adding them to PATH. This tedious process can be circumvented by creating virtual environments, which is the recommended procedure for installing Python packages. Python has built-in support for virtual environments that can be created and activated as follows:

```
$ python -m venv mimicpy_venv
$ . mimicpy_venv/bin/activate
```

This will create and activate a virtual environment that will be located in the folder mimicpy_venv, which is created in the directory from where you run the commands. You can choose, in principle, any name for the environment but avoid using names that clash with Python package names. The virtual environment can be deactivated by running the following command:

```
$ deactivate
```

To install MiMiCPy, run pip from within the virtual environment (i.e., after activating it):

```
$ python -m pip install mimicpy
```

And the VMD/PyMOL plugins:

```
$ mimicpy_plugin_installer -pymoldir /path/to/plugin/ -vmddir /path/to/plugin/
```

To use MiMiCPy within VMD/PyMOL, activate the virtual environment before starting up VMD/PyMOL.
7.4 Conda Environments

Conda is package and environment management system that allows a user to create, export, list, remove, and update environments that have different versions of Python and/or Python packages installed in them. If you are not familiar with conda, check the conda installation instructions for details about the installation and the conda environments documentation for more information about conda environments in general.

If you have conda installed in your system, you can set up a conda environment for MiMiCPy with the following commands (in this example, Python 3.7 is used):

```
$ conda create --name mimicpy python=3.7 ipython
$ conda activate mimicpy
```

This will create and activate a conda environment. The conda environment can be deactivated by running the command:

```
$ conda deactivate
```

To install MiMiCPy in your conda environment, you can use pip as explained above. To this aim, you first have to install pip in your conda environment by running:

```
$ conda activate mimicpy
$ conda install pip
```

and then proceed as explained previously if you want to install MiMiCPy without VMD and/or PyMOL support.

Instead, if you want to install MiMiCPy with VMD and/or PyMOL support, you first need to install VMD and/or PyMOL in the conda environment. This can be easily achieved with the following commands:

```
$ conda install vmd
$ conda install -c schrodinger pymol
```

When installing MiMiCPy with VMD or PyMOL support, make sure to pass the path corresponding to the VMD and/or PyMOL installation directory in the conda environment:

```
$ pip install mimicpy
$ mimicpy_plugin_installer -pymoldir /path/to/pymol/ -vmddir /path/to/vmd/
```
CHAPTER
EIGHT

FIRST STEPS: ACETONE IN WATER

Introductory tutorial to perform QM/MM simulations using MiMiC on a small system, composed of an acetone molecule in water.

Main steps:
1. Calculate the dipole moment of acetone in vacuum
2. Solvate the system and equilibrate it with classical MD
3. Prepare the input for MiMiC and perform a QM/MM MD simulation at room temperature
4. Calculate the average dipole moment from selected snapshots of the production run
5. Estimate the difference between the dipole moment in vacuum and the one at room temperature to understand the effects of solvent and temperature

This tutorial shows how to run a QM/MM MD simulation with MiMiC from scratch, including the equilibration of the system with classical MD. If you are already familiar with these preliminary steps, you can directly start from the Section QM/MM MD with MiMiC. All the files generated during these preliminary steps useful to start a QM/MM simulation are provided.

This tutorial assumes the user to have running versions of MiMiC, CPMD and GROMACS.

• Instructions to install the MiMiC Framework (the MiMiC library and the MiMiC communication library) can be found in the Installation/MiMiC Framework section.

• Instructions to install the CPMD and GROMACS with MiMiC enabled can be found in the Installation/External Programs section.

• Instructions to install the MiMiCPy can be found in the Installation/MiMiCPy section.

Tutorial based on:

• CPMD tutorial for the BioExcel Summer School on Biomolecular Simulations 2018 in Pula (Italy) from Bonvin Lab.

• MiMiC tutorial from 2018 by Viacheslav Bolnykh and Emiliano Ippoliti

• Updates on the MiMiC tutorial (February 2021) and MiMiCPy documentation by Bharath Raghavan

• CECAM Flagship School: Multiscale Molecular Dynamics with MiMiC (July 18-22, 2022 at CECAM HQ, EPFL, Lausanne, Switzerland)

Main contributors to the updated version of this tutorial:

• Andrea Levy

• Sophia Johnson

• Bharath Raghavan
8.1 System Preparation

Acetone is the organic compound with the formula (CH3)2CO, representing the simplest example of ketone. It is a colorless, volatile, flammable organic solvent naturally occurring in plants, trees, forest fires, vehicle exhaust, and as a breakdown product of animal fat metabolism. It dissolves in water and is used to make plastic, fibers, drugs, and other chemicals, and also to dissolve other substances.

For this tutorial we will assume that you downloaded the corresponding git repository, hence you will have the provided folders and files in the same locations.

curl https://gitlab.com/MiMiC-projects/MiMiC-docs/-/archive/main/MiMiC-docs-main.tar.gz?...path=docs/tutorials/AcetoneTutorial -o AcetoneTutorial.tar.gz
tar --strip-components=3 -zxvf AcetoneTutorial.tar.gz

The above commands downloaded a copy of the exercise repository AcetoneTutorial in the current folder. Navigate the repository and create your own directory to run this exercise

cd AcetoneTutorial
mkdir solution
cd solution

8.1.1 Dipole Moment in Vacuum with CPMD

We will start by using CPMD to calculate the dipole moment of acetone in vacuum: ensure that you have CPMD loaded into your coding environment.

Create a folder where we are going to perform the calculations with CPMD:

mkdir 1_dipole_vac
cd 1_dipole_vac

Geometry optimization

In order to calculate the dipole moment of the acetone in vacuum we first have to optimize the geometry of the acetone molecule. The input file is provided in the data folder for this tutorial. Copy this file in the current folder:

cp ../../data/opt_geo.inp .

The geometry together with instructions for CPMD to run the optimization can be found in this file. CPMD input files are organized in sections that start with &NAME and end with &END. Inside sections, specific keywords are present. The sequence of the sections does not matter, nor does the order of keywords (except in some special case reported in the manual). All keywords have to be in upper case, otherwise they will be ignored.

In particular, the input file opt_geo.in, used to perform a geometry optimization, contains the following sections:

- &INFO: optional section, allowing the user to insert comments about the calculation, which will be repeated in the output file.
• &CPMD: required section, specifying the instructions for the calculation. In this case, a geometry optimization (OPTIMIZE GEOMETRY) will be performed. The XYZ suboption specifies that the final structure will be also saved in xyz format in a file called GEOMETRY.xyz, and the ‘trajectory’ of the optimization in a file named GEO_OPT.xyz. Convergence criteria for wavefunction and geometry convergence are also specified in this section.

• &SYSTEM: required section, containing simulation cell and plane wave parameters. The keywords SYMMETRY, CELL and CUTOFF are required and define the (periodic) symmetry, shape and size of the simulation cell, as well as the plane wave energy cutoff (i.e. the size of the basis set), respectively. The keyword ANGSTROM specifies that the atomic coordinates, the supercell parameters and several other parameters are read in Angstrom (pay attention: default units for CPMD are atomic units (a.u.), which are always used internally).

• &DFT: optional section, reporting the exchange and correlation functional and related parameters. In this case, the BLYP functional is used (local density approximation is the default).

• &ATOMS: required section, containing atoms and pseudopotentials and related parameters. The input for a new atom type is started with a * in the first column. This line further contains the file name where to find the pseudopotential information, and several other possible labels such as KLEINMAN–BYLANDER in our case, which specifies the method to be used for the calculation of the nonlocal parts of the pseudopotential (this approximation makes the nonlocal parts calculation extremely fast but in general it also keeps also high accuracy). The next line contains information on the nonlocality of the pseudopotential: the maximum l-quantum number can be specified with LMAX= l where 1 is S for l=0, P for l=1 or D for l=2. The following line contains the number of atoms of the current type, followed by the coordinates for this atomic species.

For more details see CPMD manual and CPMD documentation.

We will perform the optimization at DFT level, using the BLYP functional. This is indicated in the &DFT section of the input file:

```plaintext
&DFT
  XC_DRIVER
  FUNCTIONAL GGA_XC_BLYP
&END
```

To run CPMD, you need to provide pseudopotentials as second input in the cpmd.x command (see below). Pseudopotentials are not distributed with CPMD, so do not forget to download them either from the CPMD website or your favourite pseudopotential repository. The pseudopotentials needed for this tutorial are provided in the AcetoneTutorial/data/pseudo repository.

The selection of proper pseudopotentials and the corresponding plane wave cutoff are crucial to obtain good results with a CPMD calculation. The pseudopotentials to use for describing different atomic species are specified in the CPMD input files in the &ATOMS section. The corresponding pseudopotential files need to be provided as input when running CPMD.

Pseudopotential files usually have .psp extension and contain different sections. In particular, the &INFO section contains a Pseudopotential Report with relevant information, and information for the pseudopotential and the pseudo wavefunction are contained in the &POTENTIAL and &WAVEFUNCTION, respectively. The columns in the &POTENTIAL section provide useful information for the choice of LMAX: the highest possible value of depends on the highest angular momentum for which a ‘channel’ was created in the pseudopotential. In the &POTENTIAL section, the first column is the radius, the next ones are the orbital angular momenta (s, p, d, f…). It is possible to use LMAX values as high as there is data for in the pseudopotential file, but you since higher values correspond to an increase in calculation time, it is not required to use the maximum possible LMAX unless the physics of the problem requires this.

For more details see CPMD manual. Pseudopotential files for different atoms can be obtained from the CPMD documentation section.

Run the optimization using the following command (one single line), making sure to specify the correct path for the AcetoneTutorial folder, instead of path-to-tutorial-folder
The calculation can be monitored using the command

```
tail -f opt_geo.out
```

When the optimization is completed, different output files are generated. In particular, the RESTART.1 file is a binary file containing the necessary information to restart a new calculation from the final step of the current run, i.e. the optimized geometry in this case. You can check what RESTART file is the last one written (the one you want to use for a new calculation) by checking the LATEST file.

In particular, the input file `opt_geo.inp`, where geometry optimization is performed, contains the following sections:

- The header, where it is possible to see when the run was started, which version of CPMD was used, and when it was compiled

```
PROGRAM CPMD STARTED AT: 2022-05-09 11:23:07.287
SETCNST| USING: CODATA 2006 UNITS

VERSION 4.3-4610

COPYRIGHT
IBM RESEARCH DIVISION
MPI FESTKOERPERFORSCHUNG STUTTGART

The CPMD consortium
Home Page: http://www.cpmd.org
Mailing List: cpmd-list@cpmd.org
E-mail: cpmd@cpmd.org

*** Sep 30 2019 -- 14:15:56 ***
```

- Some technical information about the environment where the job was run, such as machine, user, directory, input file, and process id.
- The &INFO section from the input file
- A summary of some of the parameters read in from the &CPMD section, or the default values if not set, and the exchange correlation functionals used.
- Information about atoms (and their coordinates in a.u.), electrons and states in the system, together with pseudopotentials used and the respective settings (these are copied from the pseudopotential .psp files).
- Information about how the calculation is distributed through the cores. In this case for example, only two cores are used.
A summary of the settings read from the &SYSTEM section of the input file (or their default values), including some derived parameters such as density cutoff and number of plane waves.

Generation of the initial guess for the wavefunction optimization. In this case, a superposition of atomic wavefunctions using a minimal Slater basis:

---

**GENERATE ATOMIC BASIS SET**

**C SLATER ORBITALS**

2S  \( \text{ALPHA}= 1.6083 \)  \( \text{OCCUPATION}= 2.00 \)

2P  \( \text{ALPHA}= 1.5679 \)  \( \text{OCCUPATION}= 2.00 \)

**O SLATER ORBITALS**

2S  \( \text{ALPHA}= 2.2458 \)  \( \text{OCCUPATION}= 2.00 \)

2P  \( \text{ALPHA}= 2.2266 \)  \( \text{OCCUPATION}= 4.00 \)

**H SLATER ORBITALS**

1S  \( \text{ALPHA}= 1.0000 \)  \( \text{OCCUPATION}= 1.00 \)

---

After other initialization steps, such as the initialization of the Hessian matrix, it is possible to start with the geometry optimization. For each step, a wavefunction optimization is performed, and then the positions of the atoms are updated according to the forces acting on them. The columns give information about the step number (NFI), information about the convergence, such as the largest off-diagonal component (GEMAX), the average of the off-diagonal components (CNORM), the total energy (ETOT) and its change (DETOT), and finally the CPU time (TCPU):

---

**GEOMETRY OPTIMIZATION**

---

<table>
<thead>
<tr>
<th>NFI</th>
<th>GEMAX</th>
<th>CNORM</th>
<th>ETOT</th>
<th>DETOT</th>
<th>TCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EWALD</td>
<td>SUM IN REAL SPACE OVER</td>
<td>1* 1* 1 CELLS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.104E-02</td>
<td>4.475E-03</td>
<td>-35.653984</td>
<td>-3.565E+01</td>
<td>0.71</td>
</tr>
<tr>
<td>2</td>
<td>1.843E-02</td>
<td>1.418E-03</td>
<td>-36.325881</td>
<td>-6.719E-01</td>
<td>0.73</td>
</tr>
<tr>
<td>3</td>
<td>1.806E-02</td>
<td>7.307E-04</td>
<td>-36.396587</td>
<td>-7.071E-02</td>
<td>0.72</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>34</td>
<td>1.372E-07</td>
<td>6.696E-09</td>
<td>-36.430486</td>
<td>-1.138E-11</td>
<td>0.77</td>
</tr>
<tr>
<td>35</td>
<td>6.576E-08</td>
<td>4.820E-09</td>
<td>-36.430486</td>
<td>-3.681E-12</td>
<td>0.77</td>
</tr>
</tbody>
</table>

**ATOM COORDINATES GRADIENTS (~FORCES)**

<table>
<thead>
<tr>
<th>NFI</th>
<th>GEMAX</th>
<th>CNORM</th>
<th>ETOT</th>
<th>DETOT</th>
<th>TCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C 0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>-4.390E-02</td>
<td>7.606E-02</td>
</tr>
<tr>
<td>2</td>
<td>C 2.5834</td>
<td>0.0000</td>
<td>-0.9134</td>
<td>-3.852E-02</td>
<td>4.907E-02</td>
</tr>
<tr>
<td>3</td>
<td>C -1.2917</td>
<td>-2.2373</td>
<td>-0.9134</td>
<td>6.140E-02</td>
<td>9.279E-03</td>
</tr>
<tr>
<td>4</td>
<td>O 0.0000</td>
<td>0.0000</td>
<td>2.3055</td>
<td>3.161E-02</td>
<td>9.404E-03</td>
</tr>
<tr>
<td>5</td>
<td>H 2.5834</td>
<td>0.0000</td>
<td>-2.9713</td>
<td>-1.090E-02</td>
<td>9.404E-03</td>
</tr>
<tr>
<td>6</td>
<td>H 3.5535</td>
<td>1.6803</td>
<td>-2.2274</td>
<td>4.775E-04</td>
<td>3.662E-03</td>
</tr>
<tr>
<td>7</td>
<td>H 3.5535</td>
<td>-1.6803</td>
<td>-2.2274</td>
<td>4.775E-04</td>
<td>3.662E-03</td>
</tr>
<tr>
<td>8</td>
<td>H -1.2917</td>
<td>-2.2373</td>
<td>-2.9713</td>
<td>5.511E-03</td>
<td>9.404E-03</td>
</tr>
</tbody>
</table>

(continues on next page)
The calculation stops after the convergence criterion is reached for the \textit{GEMAX} value, and then the positions and forces of the atoms are reported. After that another wavefunction optimization starts.

- Finally, at the end of the geometry optimization, the final geometry is reported, together with other information, with the breakdown of the total energy among the different components

---

**ELECTRONIC GRADIENT:**

- \textbf{MAX. COMPONENT} = 7.84233E-08
- \textbf{NORM} = 8.03717E-09

**NUCLEAR GRADIENT:**

- \textbf{MAX. COMPONENT} = 4.96207E-04
- \textbf{NORM} = 2.31465E-04

---
• The last section of the output reports the memory allocation and the timing of the run.

For more details see CPMD manual and CPMD documentation.

**Dipole moment calculation**

Now we want to calculate the dipole of the acetone for the optimized structure at ‘zero temperature’ just found by using the RESTART file to retrieve the relevant information from the previous calculation. We need a new input file, which is similar to the one used for the optimization, with few important differences:

1. The &CPMD section now asks CPMD to compute properties, using the WAVEFUNCTION and COORDINATES from the RESTART file written in the LATEST file (those files were generated in the previous calculation)

```
&CPMD
  PROPERTIES
  RESTART WAVEFUNCTION COORDINATES LATEST
&END
```

2. The properties to calculated are specified in the &PROP section

```
&PROP
  DIPOLE MOMENT
&END
```

Copy the input in the current folder and run the dipole moment calculation (again, specifying the correct path for the AcetoneTutorial folder, instead of path-to-tutorial-folder):

```
cp ../../data/dipole.inp .
cpmd.x dipole.inp path-to-tutorial-folder/AcetoneTutorial/data/pseudo > dipole.out &
```

This calculation will be much faster than the previous one, and will produce different output files. In particular, the dipole.out file contain a CALCULATE DIPOLE MOMENT section, reporting the dipole moment along x,y,z axes and the total, both in atomic units and Debye. This is what you should see:
8.1.2 System Preparation with GROMACS

Now we will move to the system preparation using classical (force field based) molecular dynamics. Create a folder where we are going to prepare the structure inputs and the topology for this tutorial:

```bash
mkdir ../2_system_prep
cd ../2_system_prep
```

### Structure File

The structure file for acetone is provided in the `data` folder. Copy this file in the current folder:

```bash
cp ../../data/acetone.gro .
```

`.gro` files contain molecular structure in Gromos87 format, in this case for the acetone molecule:

```
Acetone molecule gas phase
10
1ACT  C1  1  5.041  5.034  5.035
...
10.00000 10.00000 10.00000
```

The first line is the title (free string), the second line is the number of atoms (integer), and each of the following lines refers to one atom, reporting its residue number, residue name, atom name, atom number, position (x,y,z, in nm), and velocity (optional). The very last line contains the box vectors in x,y,z dimensions.

For more details see GROMACS documentation.
Topology

Also the topology files are provided in the data folder for this tutorial. Copy these files in the current folder:

```
cp ../../data/acetone.top .
cp ../../data/acetone.itp .
```

.top files contain the topology for the system, i.e. the parameters used for the force field, in this case for the acetone molecule (that we will solvate). Topology files contain different sections:

```
; Include forcefield parameters
#include "amber03.ff/forcefield.itp"
#include "acetone.itp"

; Include SPC water topology
#include "amber03.ff/spc.itp"
```

Here different .itp files are included. Those files are where the parameters actually are stored. In this case we include parameters from the AMBER force field (#include "amber03.ff/forcefield.itp"), containing standard parameters for proteins and nucleic acids, parameters for acetone (#include "acetone.itp"), from the .itp file provided that we have just copied in the current folder, and parameters for water. Different water models exist (for more details on this topic see for this resource), and we choose to use the SPC rigid model, which is a three-site water model, i.e. contains three interaction points corresponding to the three atoms of the water molecule.

```
[ system ]
; Name
Acetone molecule
```

The [ system ] section, which needs to be placed after any other level, except the [ molecules ] one (here the previous sections are not shown, but are contained in the different .itp files). This section contains the name of the system (a string).

```
[ molecules ]
; Compound   #mols
ACT          1
```

The [ molecules ] section defines the total number of molecules i the system. Here, for the moment, we have only one acetone molecule. Each name present in this section must correspond to a name given with [ moleculetype ] previously in the topology (in our case, in the .itp files). The order of the blocks of molecule types and the numbers of molecules must match the coordinate file (acetone.gro in this case).

For more details see GROMACS documentation.

Solvation

After this preparatory step, we will now start to actually use GROMACS. Ensure that you either have GROMACS loaded into your coding environment.

We will solvate the acetone molecule, i.e. add water molecules to our structure. To do this, we can use the solvate command of GROMACS.

```
gmx solvate -cp acetone.gro -cs spc216.gro -p acetone.top -o solvated.gro -box 2.5 2.5 2.
```
Where we have specified the water model (redundant here, since the spc216.gro model is the default one for the solvate command) and the size of the box, namely a cubic box of size 2.5nm. The solvated structure is saved as solvated.gro, and the topology file has been updated. A backup file with the previous topology is saved as well, named acetone.top.1#

Doublecheck the new topology: the last lines should appear as

<table>
<thead>
<tr>
<th>ACT</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOL</td>
<td>506</td>
</tr>
</tbody>
</table>

with a single line per molecule type. If this is not the case and a single line is present after running the solvate command, i.e.

| ACT | 1SOL | 506 |

edit the topology as explained above with one line per molecule type!

The solvate command solvates a solute configuration in a box filled with solvent molecules. Different options can be used:

- -cp structure file for the solute, such as a protein
- -cs structure file for the solvent (where spc216.gro is the default one, i.e. SPC water model)
- -p topology file for the system
- -box box size (x,y,z vectors) in nm

This command will produce a new structure file containing the solvated system, and will also update the topology file (keeping a backup version of the input topology). In particular, the [ molecules ] section of the topology will now contain

```
[ molecules ]
; Compound #mols
ACT       1
SOL       506
```

Specifying that 506 solvent molecules, with molecule type SOL, which is defined in the water topology file that we already included in the main topology file.

For more details see GROMACS documentation.

## 8.2 Classical Equilibration

We are interested in studying the effects of the presence of water molecules on the dipole of acetone molecule, but the solvated system that we just created is based on the structure of acetone in vacuum. Hence, we need to equilibrate the system before continuing with QM/MM MD. In any case, the first step before running a QM/MM simulation is a classical force field-based molecular dynamics equilibration.

The equilibration we will perform is composed of two main steps:

1. Energy minimization on the initial structure which is needed to minimize the energy with the selected force field of the starting structure of the solvated system. This step will, for example, improve the orientation of hydrogen bonds.
2. Equilibration during which we will heat the system, using the minimized structure, by increasing temperature from 0K to 300K (room temperature).
8.2.1 Energy Minimization

Create a folder where we are going to perform the energy minimization step and copy the provided mini.mdp file, containing the parameters for the minimization:

```bash
mkdir ../3_mini
cd ../3_mini
cp ../../../data/mini.mdp .
```

mdp files contain parameters to set up a molecular dynamics run with a given system. We later process this file with the grompp command (explained later) to generate a binary .tpr file. The .mdp file parameters depend on the specific process and system so each time we move to a different MD process we’ll need a new .mdp input file (for example when we transition to QM/MM). Below we provide a sample .mdp file for an energy minimization run with comments following ; symbols:

```plaintext
; General MD parameters for an energy minimization process
integrator = steep ; MD Algorithm selection (steep = steepest descent minimization).
emtol = 100.0 ; stop minimization when the maximum force < 100.0 kJ/mol/nm
emstep = 0.01 ; energy step size
nsteps = 50000 ; maximum number of (minimization) steps to perform

; Parameters describing how to find the neighbors of each atom and how to calculate their interactions
nstlist = 1 ; frequency to update the neighbor list and long range forces. Lowest possible value is 0 which equates to a list that is created and then never updated
cutoff-scheme = Verlet ; type of cut-off scheme. 'Verlet' generates a pair list with buffering
ns_type = grid ; method to determine neighbor list (simple, grid)
coulombtype = PME ; treatment of long range electrostatic interactions. This example shows a Particle-Mesh Ewald selection ('PME'). For more options see GROMACS documentation link below.
rcoulomb = 1.0 ; short-range electrostatic cut-off in nm
rvdw = 1.0 ; short-range Van der Waals cut-off in nm
pbc = xyz ; periodic boundary conditions used in all directions (xyz). Other options could be 'no' to ignore the box or 'xy' to apply pbc to x and y directions only.
```

For more details see GROMACS documentation.

The .mdp files can be converted into binary files using the GROMACS preprocessor command grompp, which requires different inputs. Prior to running, make sure that ACT and SOL are on separate lines in molecules section of the acetone.top file.

```bash
gmx grompp -f mini.mdp -c ../../../2_system_prep/solvated.gro -p ../../../2_system_prep/acetone.top -o mini.tpr
```

The grompp command converts our solvated.gro file from a molecular topology file into a binary file, mini.tpr, according to the parameters specified in the mini.mdp input file. Different options can be used:

- `-f` read the MD parameter inputs from the .mdp file
- `-c` structure file in .gro or .g96 or .pdb or .brk or .ent or .esp or .tpr
- `-p` topology file in .top format
MiMiC

- n [optional] index file in .ndx format
- r or -rb [optional] restraints in any of the formats possible with the -c flag, could even be the same file as provided for the -c flag
- t [optional] trajectory in .trr or .cpt or .tng format to read starting coordinates if needing to restart a process
- e [optional] energy file in .edr format to provide Nose-Hoover and/or Parrinello-Rahman coupling variables

Additionally, grompp uses a built-in preprocessor which can support the following keywords:

```
#ifndef VARIABLE
#define VARIABLE
#endif
#include "filename"
```  
And these keywords can be enacted and changed in the associated .mdp file by including statements like:

```
define = -DVARIABLE1 -DVARIABLE2
include = -<file_path>
```  

There are additional options to control the output files. For more details see GROMACS documentation.

Now that we generated the mini.tpr file, we are actually able to run the minimization using the mdrun command.

```
gmx mdrun -deffnm mini &
```  
- -deffnm uses the following string for all the file options (input, output, error, log, etc)
- -s portable xdr run input file in .tpr format
- -o output a full precision trajectory in a .trr or .cpt or .tng format
- -c structure file in .gro or .g96 or .pdb format
- -ntomp number of OpenMP threads per MPI rank

For more details see GROMACS documentation.

This procedure will generate different output files. In particular, the progress of the minimization can be monitored using the .log file, using the command

```
tail -f mini.log
```  

When the minimization will be completed, a message will be printed in the log file, stating how many steps have been required to converge the minimization, together with other interesting quantities to look at, such as the potential energy of the final configuration and the maximum force (and the atom on which it acts). The final configuration can be found in the mini.gro file, and will become the starting point for the heating phase.
### 8.2.2 Heating - Classical Equilibration

The next step is to thermalize the system classically and to adjust the pressure in the box to reach standard conditions. Create a folder where we are going to perform the heating step and copy there the `heat.mdp` input file provided. We need a new `.mdp` file because we are performing a different type of simulation and therefore need different parameters.

```bash
mkdir ../4_heat
cd ../4_heat
cp ../../data/heat.mdp .
```

The `heat.mdp` requires information about the temperature and pressure control. You’ll see that we need to define more parameters than we did for the simple energy minimization process. Below we provide a sample `.mdp` file for a classical equilibration run with comments following ; symbols:

```plaintext
; Run parameters
integrator = md ; MD Algorithm selection (leap-frog integrator)
dt = 0.002 ; time step in ps. Here we define a 2 fs time step
nsteps = 100000 ; number of steps in the simulation. Here: 2 fs/step * 100000 steps = 200 ps

; Output control
nstxout = 500 ; save coordinates every 1.0 ps
nstvout = 500 ; save velocities every 1.0 ps
nstenergy = 500 ; save energies every 1.0 ps
nstlog = 500 ; update log file every 1.0 ps

; Bond parameters
continuation = no ; when turned on, this option specifies NOT to apply constraints to start or to reset shells.
+exact continuation.
+have this option set to `no` so that the constraints are applied at the start of the run.
constraint_algorithm = lincs ; choice of constraint solver for any non-SETTLE holonomic constraints. Here we select a LINear Constraint Solver (lincs).
constraints = all-bonds ; which bonds are constrained. `all bonds` means all bonds convert to constraints (even heavy atom-H bonds).
lincs_iter = 1 ; max number of iterations for the lincs solver which controls the accuracy of LINCSS
lincs_order = 4 ; number of matrices in the expansion for the matrix inversion during constraint solving. This is also related to accuracy

; Neighbor searching
cutoff-scheme = Verlet ; type of cut-off scheme. `Verlet` generates a pair list with buffering
ns_type = grid ; method to determine neighbor list (simple, grid)
nslist = 10 ; frequency to update the neighbor list and long range forces. Lowest possible value is 0 which equates to a list that is created and then never updated. This option is largely irrelevant with a Verlet cut-off scheme.
rcoulomb = 1.2 ; short-range electrostatic cutoff in nm
rvdw = 1.2 ; short-range van der Waals cutoff in nm
```

(continues on next page)
; Electrostatics
coulombtype = PME ; treatment of long range electrostatic interactions. This example shows a Particle-Mesh Ewald selection (`PME`). For more options see GROMACS documentation link below.
pme_order = 4 ; order of PME interpolation. Here we define a cubic interpolation.
fourierspacing = 0.16 ; grid spacing for FFT in nm. For optimizing the relative load of the particle-particle interactions and the mesh part of PME, it is useful to know that the accuracy of the electrostatics remains nearly constant when the Coulomb cut-off and the PME grid spacing are scaled by the same factor.

; Temperature coupling
tcoupl = V-rescale ; selection of thermostat/temperature control scheme. Here we select a modified Berendsen thermostat.
tc-grps = System ; define groups to couple to separate thermal baths. Here, with two coupling groups for system and solvent (basically everything else except the system) we can run a more accurate/efficient equilibration process.
tau_t = 0.1 ; time constant for coupling with the bath in ps. A value of -1 specifies no coupling.
ref_t = 300 ; reference temperature, one for each group defined, in K

; Pressure coupling
pcoupl = Berendsen ; selection of barostat/pressure control scheme. Here we select a modified Berendsen barostatting scheme.
pcoupltype = isotropic ; type or isotropy of pressure coupling. Here we employ a uniform scaling of box vectors with tau-p. When this option is selected a compressibility value and reference pressure value are required.
tau_p = 2.0 ; time constant for pressure coupling, in ps
ref_p = 1.0 ; reference pressure, in bar
compressibility = 4.5e-5 ; isothermal compressibility of water at your target conditions, bar^-1

; Periodic boundary conditions
pbc = xyz ; periodic boundary conditions used in all directions (`xyz`). Other options could be `no` to ignore the box or `xy` to apply pbc to x and y directions only.

; Dispersion correction
DispCorr = EnerPres ; dispersion corrections to account for the cut-off applied for vdw. Here `EnerPres` tells our system to apply long range dispersion corrections for both Energy and Pressure outputs.

; Velocity generation
gen_vel = yes ; generate velocities with a random seed according to a Maxwell distribution
gen_temp = 300 ; temperature for the Maxwell distribution
gen_seed = -1 ; generate a random seed. A `-1` value uses a pseudo-random seed

For more details see GROMACS documentation.

As before, we first need to create the .tpr binary file using the GROMACS preprocessor command grompp, where the starting
and then run the equilibration simulation:

```bash
gmx mdrun -deffnm heat &
```

As before, the progress of the heating phase can be monitored using the `tail` command:

```
tail -f heat.log
```

and at the end a report will be printed to the file, and a `heat.gro` file containing the equilibrated configuration will be generated.

GROMACS provides several options for temperature control during a molecular dynamics simulation. You can add the option in the `.mdp` file with the keyword `tcoupl` followed by one of the options below:

- `-no` no temperature control
- `-nose-hoover` the Nose-Hoover thermostat defines an extended Lagrangian to manage the thermostat variables. Nose-Hoover chains are the typical choice for NVE production runs due to their stability.
- `-berendsen` the Berendsen thermostat adds a small coupling factor which moves the system towards a reference temperature. This is a “historical” thermostat mainly present to be able to reproduce previous simulations, but it is strongly recommend not to use it for new production runs
- `-andersen` the Andersen thermostat randomizes velocities of some particles at each time step
- `-andersen-massive` the Andersen massive thermostat randomizes velocities of some particles but not at each time step
- `-v-rescale` temperature coupling using velocity rescaling with a stochastic term. This thermostat is similar to Berendsen coupling, but the stochastic term ensures that a proper canonical ensemble is generated.

We must define other thermostat variables in the `.mdp` file such as reference temperature (`ref-t`) and time constant for coupling (`tau-t`). For more details see GROMACS documentation.

### 8.2.3 Equilibration Check

Is the system well equilibrated? One way to assess that is the GROMACS `energy` tool. This allows to monitor the behavior along the trajectory of the quantities you are interested in. Select the quantities you want to check using the corresponding number, and type a `0` to exit the command:

```bash
gmx energy -f heat.edr -o heat_check.xvg
```

for example, to look at total energy (11) and temperature (13), type:

```
> 11 13 0
```

This will perform a statistical analysis over all the steps stored in the `.edr` file and plot in particular the average and the estimated error on the terminal. You should obtain something similar to:

```
Statistics over 100001 steps [ 0.0000 through 200.0000 ps ], 2 data sets
All statistics are over 1001 points

<table>
<thead>
<tr>
<th>Energy</th>
<th>Average</th>
<th>Err.Est.</th>
<th>RMSD</th>
<th>Tot-Drift</th>
</tr>
</thead>
</table>
```

(continues on next page)
Check that the average temperature is close to 300K, or in general to the temperature you set as ref-t, without too large of fluctuations.

Note that the above command has also generated a heat_check.xvg file, which is helpful to plot and visually inspect data. This file is organized in columns based on the quantities selected before. In our case, the first column contains the time in ps (we asked GROMACS to store the energy every 500 steps, i.e. 1 ps using a timestep of 0.002 ps, setting nstenergy 500 in the heat.mdp file).

You can use any program to visualize this, in particular using gnuplot the commands needed to plot Total-Energy (column 2) and Temperature (column 3), are:

```gnuplot
set multiplot layout 1, 2
set title "Total Energy"
set xlabel "Time (ps)"
set ylabel "Energy (kJ/mol)"
plot 'heat_check.xvg' u 1:2 w l lt 2 title "Total Energy"
set title "Temperature"
set xlabel "Time (ps)"
set ylabel "Temperature (K)"
plot 'heat_check.xvg' u 1:3 w l lt 1 title "Temperature"
```

Here you can see an example of the evolution of these two quantities during the heating phase. In particular, both energy and temperature should oscillate around fixed values, in particular equal to 300K (ref-t) for the temperature. These two plots, together with the average values and the errors estimated with the energy tool, confirm that the system is well equilibrated. We are now finally ready to run QM/MM MD!
Now that the preliminary steps are completed, and the system has been equilibrated at classical level, we are finally ready to run QM/MM MD! In particular, we want to run a QM/MM simulation in which the acetone molecule is treated quantum mechanically, while the surrounding water molecules are treated at classical MM level.

All the files generated during the preliminary steps useful to start a QM/MM simulation are provided in the AcetoneTutorial/data/ folder.

In case you skipped the first sections and you are starting the tutorial from this point, run this preliminary commands to download the corresponding git repository and create a solution folder, hence you will have the provided folders and files in the same locations.

```
curl https://gitlab.com/MiMiC-projects/MiMiC-docs/-/archive/main/MiMiC-docs-main.tar.gz?path=docs/tutorials/AcetoneTutorial -o AcetoneTutorial.tar.gz
tar --strip-components=3 -zxvf AcetoneTutorial.tar.gz
cd AcetoneTutorial
mkdir solution
cd solution
```

You can now either copy your files a new folder inside the solution folder (if you have followed the tutorial so far), or use the ones provided (if you are starting the tutorial now, or you are not confident about the files you generated).

```
mkdir 5_mimic_prep
cd 5_mimic_prep

cp ../../data/acetone_equilibrated.gro ./heat.gro
cp ../../data/acetone_equilibrated.top ./acetone.top
cp ../../data/acetone.itp .
```

### 8.3.1 Input Preparation with MiMiCPy

MiMiC uses CPMD together with GROMACS in order to run a QM/MM simulation. The Python package MiMiCPy is the companion library to MiMiC in the preparation of QM/MM simulations. It comes with a set of command lines tools to prepare MiMiC input scripts. Additionally, plugins for PyMOL and VMD are also provided.

Instructions to install the MiMiCPy can be found in the Installation/MiMiCPy section.

#### Index file and GROMACS tpr

The CPMD input file and the GROMACS .tpr binary file have to be prepared at the same time, making sure that both CPMD and GROMACS receive the same QM atoms information. This can be easily achieved with the MiMiCPy package. As explained in the introduction, we assume you already installed mimicpy.

In particular, the MiMiCPy prepqm script allows the user to easily generate a CPMD input script (default name cpmd.inp) and a GROMACS index file (default name index.ndx). The latter can be used to generate the GROMACS .tpr file needed for the QM/MM simulation with MiMiC. Run the MiMiCPy prepqm script:

```
mimicpy prepqm -top acetone.top -coords heat.gro
```

The command will open an interactive session with the following message printed out:
Some atom types had no atom number information. They were guessed as follows:

<table>
<thead>
<tr>
<th>Atom Type</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>c3</td>
<td>C</td>
</tr>
<tr>
<td>o</td>
<td>O</td>
</tr>
<tr>
<td>hc</td>
<td>H</td>
</tr>
</tbody>
</table>

MiMiCPy requires the atomic element information for each atom to generate the CPMD input script. Some non-standard atom types in the GROMACS topology file do not contain this information. MiMiCPy guesses them based on the combination of atomic mass, name and type.

(Always verify that the guessed atomic elements for each atom type are meaningful, as it is essential for CPMD to run correctly)

MiMiCPy guesses non-standard atom types based on the combination of atomic mass, name and type. The automatic behavior of guessing atomic elements can be toggled on or off using the -guess option. If False is passed and non-standard atoms are present, MiMiCPy will exit with an error message instead of attempting to guess the elements. If you are not satisfied with the element information guessed, a file containing the list of all non-standard atom types with the correct element information can be passed to mimicpy prepqm with the -nsa option. For example, if MiMiCPy guessed the wrong atomic number for atom type c3 and o, we could create a file called atomtypes.dat with the following text:

```
o 0
 c3 C
```

and pass it to MiMiCPy when running prepqm

```
mimicpy prepqm -top acetone.top -coords heat.gro -nsa atomtypes.dat
```

At this point, the user can provide instructions to add and/or deleted atoms to the QM region in an interactive session. The instructions specified after add correspond to the selection query that identifies the atoms to be added to the QM region. MiMiCPy provides a selection language similar to PyMOL and VMD for selecting QM atoms. In this case, we want to add the ACT molecule to the QM region with the following command:

```
> add resname is ACT
> q
```

MiMiCPy now generated the CPMD input script cpmd.inp and a GROMACS index file index.ndx. The latter contains the indeces of the QM atoms in the GROMACS topology, and needs to be used to generate the GROMACS .tpr file needed for the QM/MM simulation with MiMiC.

If you would prefer, the same instruction add resname is ACT can be stored as a file sele.dat and passed to mimicpy prepqm for faster execution. In the case the selection is provided as input file:

```
cp ../../data/sele.dat .
```

In this case (not necessary now that you've already completed the prepqm step with the interactive session), the prepqm script can be run with an additional input -sele:
As explained above, this command generates the CPMD input script `cpmd.inp` and a GROMACS index file `index.ndx`.

The syntax for the selection query involves the following general structure:

```
keyword logical_operator value
```

where `keyword` can include `resname`, `resid`, `name`, `type`, `id`, and `mol`. `logical_operator` can be `is`, `not`, `>`, `<`, `>=`, or `<=`.

Different selection queries can be joined by using `and` or `or` operators, and grouped with brackets, such as

```
add (resname is RES) and (name is X)
```

which will add atom named X of residue RES. With the same selection syntax atoms can also be deleted from the QM region, using the `delete` command instead of `add`.

To check the current QM region, the `view` command can be used. When the desired QM atoms are selected, type `q` to exit.

A binary `.tpr` file can be generated using GROMACS preprocessor (`gmx grompp`), analogously to what was is usually done with GROMACS (as done in the Classical Equilibration section). Copy the `.mdp` file provided and run GROMACS preprocessor, passing the index file just generated. It is important that the label for QMMM-groups matches the index file label (here qmatoms)!

```
cp ../../data/mimic.mdp .
gmx grompp -f mimic.mdp -c heat.gro -p acetone.top -o mimic.tpr -n index.ndx
```

The same coordinate and topology file used to run the earlier MiMiCPy command should be passed to gromacs grompp as well.

The two steps that we have just performed, i.e. generating an index file with MiMiCPy and using that to run the GROMACS preprocessor, can be combined in a single MiMiCPy command. In particular, if you selected the QM atoms in the `sele.dat` file, the `mimic.mdp` file can be passed to `mimicpy prepqm` command and this will automatically generate not only the `index.ndx` file, but also the `mimic.tpr` file.

**Before running the next command, make sure GROMACS has been sourced, i.e. is available on the command line.** In particular, you may need to set the environment variable `GMXLIB` with the following command, where the path to use the the one where the GROMACS share/gromacs folder is located

```
export GMXLIB="/path-to-gromacs-installation/share/gromacs/"
```

Now you can run mimicpy:

```
mimicpy prepqm -top acetone.top -coords heat.gro -sele sele.dat -mdp mimic.mdp -gmx gmx -tpr mimic.tpr
```

With this command, MiMiCPy would generate not only the `cpmd.inp` and `index.ndx` files as before, but also `mimic.tpr` using `gmx grompp`, like the GROMACS command we just ran. Note that in order to execute this MiMiCPy command you will need to have correctly sourced GROMACS and loaded all necessary modules as this command includes a call to GROMACS to perform the `grompp` command.


**CPMD input**

Together with the `indexndx` and/or `mimic.tpr` file, MiMiCPy generated also the CPMD input script `cpmd.inp`. This is a barebones CPMD input file, containing the most essential commands to run a MiMiC simulation.

A CPMD input file for a QM/MM simulation is similar to the CPMD input file for a standard QM calculation that has been previously described (Introduction section: Dipole Moment in Vacuum with CPMD). However, there are some main differences that should always be taken into account for the new QM/MM interface of CPMD:

- In the &CPMD section the keyword MIMIC needs to be added.
- A new &MIMIC section is mandatory. This section is the output from the MiMiCPy preprocessor, the 3 required keywords PATHS, BOX and OVERLAPS are provided. In this section, other keywords can be added. The most relevant keywords for the &MIMIC section are:
  - **PATHS**, the number of the additional layers beyond the QM one treated by CPMD. In the case of QM/MM, this is 1. In the following lines, for each layer the absolute path where the corresponding files are stored has to be specified (in our case the path where the GROMACS .tpr file is located). **Always check that this path is consistent and is not indented - otherwise the simulation will get crash!**
  - **BOX**, the size of the classical box, in a.u.
  - **OVERLAPS**, i.e. how many (first line) and which are the atoms (following lines) from one of the codes that should be treated by another one. In our case in this section the QM atom needs to be specified. The format for the lines specifying the overlapping atoms is a sequence of four numbers representing: the code (CPMD = 1, GROMACS = 2), the atom ID, the cpde ID that has to treat it, and the atom ID in that code.
  - **LONG-RANGE COUPLING**, which turns on (if present) long-ranged electrostatic coupling to minimize the computational cost of the simulation. Interactions with MM atoms within CUTOFF DISTANCE (20 in this case) will be treated using explicit Coulomb potential. For atoms that are outside the cutoff, a multipolar expansion of the electrostatic potential of an electronic grid will be used.
  - **FRAGMENT SORTING ATOM-WISE UPDATE**, which specifies the sorting frequency, i.e. the number of steps between two consecutive sorting of the atoms into the short and the long-ranged groups (100 in this case).
  - **MULTIPOLE ORDER**, which dictates the order at which the multipolar expansion is truncated (3 in this case).
- In the &ATOMS section, the QM atoms has to be specified as in the full QM calculations.
- The keyword ANGSTROM in the &SYSTEM section cannot be used. Lengths have to be expressed in a.u.
- The option ABSOLUTE in the keyword CELL cannot be used. The correct syntax for the size of an orthorhombic box A x B x C is A B/A C/A 0 0 0
- The QM system in a QM/MM calculation can only be dealt as isolated system, i.e. without explicit PBC since there is the MM environment all round it. Even though the option ISOLATED SYSTEM (or 0) is used for the SYMMETRY keyword, the calculation is, in fact, still done in a periodic cell: we are still using a plane wave basis set to expand the wavefunction of the QM part. Since the acetone molecule has a dipole moment, we have to take care of the long-range interactions between periodic images and there are methods (activated with the keyword POISSON SOLVER in the &SYSTEM section) implemented in CPMD to compensate for this effect. In particular, TUCKERMAN Poisson solver is used, since it has been proven to be the most effective one with typical systems studied in biology. Decoupling of the electrostatic images in the Poisson solver requires increasing the box size over the dimension of the molecule: practical experience shows that 3-3.5 Å between the outmost atoms and the box walls is usually sufficient for typical biological systems.

Details of the CPMD input to perform QM/MM calculations with MiMiC are described in the above drop-down menu. In particular, few things need to be noted/fixed:

- The path specified in the PATHS keyword of the &MIMIC section refers to the path of the directory from which GROMACS will run, i.e. where the mimic.tpr file is located. By default, MiMiCPy uses the current di-
rectory. Alternatively, a path can be specified with the -path option when running mimicpy prepqm. Always check that this path is consistent and is not indented - otherwise the simulation will get crash!

- The size of the QM region is specified in the CELL keyword in the &SYSTEM section. The cell size cannot be written in the ABSOLUTE format, and the correct syntax for the size of an orthorhombic box $A \times B \times C$ is

$$CELLA B/A C/A \theta \theta \theta$$

where $\theta \theta \theta$ refer to the cosines of the angles between the vector, i.e. perpendicular vectors in this case. By default, MiMiCPy writes a cell size exactly bounding the QM region. This is often not enough to contain the plane waves of the QM region. For this reason, a padding needs to be added: this can be done using the -pad option when running mimicpy prepqm, i.e. an additional distance between the QM atoms and the walls of the QM box (in nm). A standard value for the Poisson solver is 3-3.5 Å, but in general the question of the right padding should be considered and chosen wisely depending on the system under study. MiMiCPy adds this on both sides in each direction; therefore, with a -pad of 0.3, a value of 0.7 nm is actually added in each direction to the size of the QM system

- The charge of the QM region is specified in the CHARGE keyword in the &SYSTEM section. This is calculated by MiMiCPy by summing the MM charges read from the force field data. This is usually enough, but it may need to be cross checked. Alternatively, a charge value to be written in the output can be specified using the -q option when running mimicpy prepqm.

- The QM atoms are specified in the &ATOMS section. Default values are written by MiMiCPy, but pseudopotential information needs to be adapted. This can be done by hand or passed a file pp_info.dat (provided) to the -pp option of mimicpy prepqm. This file should contains information like the pseudopotential filenames for a specific atom, the same for the link atom version and other labels like LMAX and LOC.

Keep the LMAX values consistent with what we used for the dipole moment calculation in vacuum, namely $S$ for H and P for C and O. By default, all these have been set to $S$ by the preprocessor if the -pp option is not used (see below).

The extra sections not in cpmd.inp file outputted by MiMiCPy can be added in by hand. A more efficient way is to store all the extra parameters in a separate template file template.inp (provided). In the example provided, these are the parameters required for a full MiMiC simulated annealing (excluding the parameters in &ATOMS and &MIMIC sections outputted by mimicpy prepqm). Other options can be specified, they can be listed by typing mimicpy prepqm --help.

You can edit manually the cpmd.inp file, or invoke MiMiCPy again, this time passing all the option that we mentioned above: the template.inp file via the -inp option to provide a template for the generation of the complete input file, the -pad option, to properly define the size of the QM region. We now use the same procedure explained in the previous section, passing also the selection of the QM atoms as input (using the sele.dat file), and we will also use the -gmx option, to directly invoke the grompp GROMACS preprocessor.

Before running the next command, make sure GROMACS has been sourced, i.e. is available on the command line. In particular, you may need to set the environment variable GMXLIB with the following command, where the path to use the one where the GROMACS share/gromacs folder is located

```
export GMXLIB="/path-to-gromacs-installation/share/gromacs/
```

Now you can run:

```bash
cp ../../data/sele.dat .
cp ../../data/pp_info.dat .
cp ../../data/template.inp .
mimicpy prepqm -top acetone.top -coords heat.gro -sele sele.dat -inp template.inp -pp...
     -pp_info.dat -pad 0.35 -out annealing.inp -mdp mimic.mdp -gmx gmx -tpr mimic.tpr
```

With the last command, we overwrote the index.ndx and the mimi.tpr files previously generated. This last command shows you how you can easily generate those two files, together with a complete CPMD input for MiMiC, using MiMiCPy. In particular, this time MiMiCPy inserts the &ATOMS section, etc. into template.inp, generating the
annealing.inp CPMD input file (whose name has been specified with the option -out). This is a full CPMD input file ready to be used for running MiMiC.

### 8.3.2 QM/MM calculations

In the previous sections, we obtained the equilibrated coordinates at room conditions using classical MD. This will be the starting point for a QM/MM simulation. However, we first need to equilibrate the system at QM/MM level since the two levels of theory are different. Therefore, it is necessary to firstly optimize the geometry of the system. Once we have a QM/MM minimized structure, we can heat the system to room temperature and run a QM/MM MD simulation.

#### Annealing

A minimal energy structure at QM/MM level can be obtained with a simulated annealing (this is the reason of the keyword ANNEALING IONS in the input file), i.e. we run a Born-Oppenheimer MD where gradually removing kinetic energy from the nuclei by multiplying velocities with a factor (set to 0.95 in our input, so 5% of the kinetic energy will be removed at each step). Create a new folder where we will perform the annealing and copy all the files needed for the simulation:

```bash
mkdir ../6_annealing
cd ../6_annealing
cp ../5_mimic_prep/mimic.tpr .
cp ../5_mimic_prep/annealing.inp .
```

Update the path in the PATH keyword in the &MIMIC section - no white spaces!

The &CPMD section contains few keyword which have not been explained in detail yet:

- **MOLECULAR DYNAMICS BO**, to perform a molecular dynamics run. BO stands for a Born-Oppenheimer type of MD. An alternative option is CP, which stands for a Car-Parrinello type MD.
- **ANNEALING IONS**, specifies to perform a simulated annealing simulation, where kinetic energy is gradually removed from the nuclei by multiplying velocities with a factor specified in the following line (0.99 here, so 1% of the kinetic energy removed at each step).
- **NEW CONSTRAINTS**, which switches on the new constraint solver specifically designed for the MiMiC interface
- **TEMPERATURE**, the initial temperature for the atoms (in Kelvin), specified in the following line (300 in this case). Here we choose the temperature at which we equilibrated the system at force field level.
- **TIMESTEP**, time step (in a.u.), specified in the following line (10 in this case). The choice of the timestep is crucial to have a stable simulation, and at the same time to optimize the time for the computation. For BO MD, a time step of 10 a.u. (~ 0.24 fs) is usual.
- **MAXSTEP**, maximum number of steps for MD to be performed, specified in the following line.
- **TRAJECTORY SAMPLE**, the frequency at which to save atomic positions, velocities and optionally forces. It is specified with a number, corresponding to the number of steps, specified in the following line. If this is set to 0, no TRAJECTORY file will be written.
- **STORE**, the frequency at which to update the RESTART file. It is specified with a number, corresponding to the number of steps, specified in the following line.
- **RESTFILE**, the number of distinct RESTART files generated during CPMD runs, specified in the next line.

A QM/MM simulation with the MiMiC interface will require running at the same time two different (parallel) processes, one for CPMD and one for GROMACS. The user is totally free to choose the best partitioning for the nodes/cores at one’s own disposal, but in this tutorial we require two cores for CPMD and one core for GROMACS: in the largest majority of the cases one core for GROMACS is enough. Run the annealing (all commands as one line), making sure to
specify the correct path to the AcetoneTutorial folder, instead of path-to-tutorial-folder, to provide CPMD the pseudopotentials.

To run CPMD, you need to provide pseudopotentials as second input in the cpmd.x command (see below). Pseudopotentials are not distributed with CPMD, so do not forget to download them either from the CPMD website or your favourite pseudopotential repository. The pseudopotentials needed for this tutorial are provided in the AcetoneTutorial/data/pseudo repository.

First we set all the OpenMP threads to 1 to ensure each core is executing effectively. In this run of MiMiC, we will use three cores, 2 for CPMD and 1 for GROMACS.

```bash
export OMP_NUM_THREADS=1
```

```bash
mpirun -n 2 cpmd.x annealing.inp path-to-tutorial-folder/AcetoneTutorial/data/pseudo > annealing.out
--annealing.out : -n 1 gmx mdrun -deffnm mimic -ntomp 1 &
```

While the simulation runs you can monitor the decreasing temperature (third column of the ENERGIES file):

```bash
tail -f ENERGIES
```

The calculation will perform 300 steps (MAXSTEP in the annealing.inp file). These should be enough to reach a low temperature, about 1-2K. Once the annealing is completed, the last configuration will be stored in the RESTART.1 file (the last restart file written is anyway specified in the LATEST file). Wait until CPMD finishes writing the file and then verify that CPMD and GROMACS processes are stopped.

The annealing.out file reports some new sections with respect to what we have seen in the Dipole Moment in Vacuum with CPMD section (which was a pure QM calculation):

- The energy report shows the different energetic contributions, for example:

```
(K+E1+L-N+X-MM+QM/MM) TOTAL ENERGY = -44.53592240 A.U.
(K=E1+L+X-MM) TOTAL QM ENERGY = -36.46982014 A.U.
(MM) TOTAL MM ENERGY = -8.02852010 A.U.
(QM/MM) TOTAL QM/MM ENERGY = -0.03758217 A.U.
(K) KINETIC ENERGY = 28.12240741 A.U.
(E1=A-S+F) ELECTROSTATIC ENERGY = -27.50067179 A.U.
(S) ESELF = 29.92067103 A.U.
(R) ESR = 1.78558591 A.U.
(L) LOCAL PSEUDOPOTENTIAL ENERGY = -29.86421972 A.U.
(N) N-L PSEUDOPOTENTIAL ENERGY = 3.61298065 A.U.
(X) EXCHANGE-CORRELATION ENERGY = -10.84031668 A.U.
GRADIENT CORRECTION ENERGY = -0.57543727 A.U.
```

- After the force initialization section, BO MD begins: for each MD step, the wavefunction is re-converged. The MD step number is indicated as NFI (number of finite iterations), and for MD each step different quantities are reported: temperature, calculated as the kinetic energy divided by the degrees of freedom (TEMPP), quantum DFT Kohn-Sham electron energy, equivalent to the potential energy in classical MD (EKS), total energy in a classical MD (ECLASSIC), mean squared displacement of the atoms from the initial coordinates (DIS), and the time took to calculate this step (TCPU). Since in BO MD the wavefunction is re-converged, for each step information about the convergence are reported.

<table>
<thead>
<tr>
<th>NFI</th>
<th>TEMPP</th>
<th>EKS</th>
<th>ECLASSIC</th>
<th>DIS</th>
<th>TCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>297.1</td>
<td>-44.53592</td>
<td>-43.09507</td>
<td>0.844E-04</td>
<td>16.34</td>
</tr>
<tr>
<td>INF</td>
<td>GEMAX</td>
<td>ETOT</td>
<td>DETOT</td>
<td>TCPU</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8.907E-04</td>
<td>-36.507197</td>
<td>0.000E+00</td>
<td>3.02</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.218E-04</td>
<td>-36.507428</td>
<td>-2.304E-04</td>
<td>1.32</td>
<td></td>
</tr>
</tbody>
</table>
When the simulation is completed (due to the maximum number of steps/computational time reached or “soft” EXIT message received), a summary of averages and root mean squared deviations for some of the monitored quantities is reported. This is useful to detect unwanted energy drifts or too large fluctuations in the simulation:

<table>
<thead>
<tr>
<th></th>
<th>INFR</th>
<th>GEMAX</th>
<th>ETOT</th>
<th>DETOT</th>
<th>TCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.942E-04</td>
<td>-36.507225</td>
<td>0.000E+00</td>
<td>3.02</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.189E-04</td>
<td>-36.507455</td>
<td>-2.304E-04</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.159E-04</td>
<td>-36.507472</td>
<td>-1.613E-05</td>
<td>1.35</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.981E-04</td>
<td>-36.507475</td>
<td>-3.273E-06</td>
<td>1.33</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9.894E-05</td>
<td>-36.507475</td>
<td>-5.344E-07</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3.474E-05</td>
<td>-36.507476</td>
<td>-1.739E-07</td>
<td>1.42</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2.159E-05</td>
<td>-36.507476</td>
<td>-9.483E-08</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.275E-05</td>
<td>-36.507476</td>
<td>-5.612E-08</td>
<td>1.12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5.119E-06</td>
<td>-36.507476</td>
<td>-2.584E-08</td>
<td>1.10</td>
<td></td>
</tr>
</tbody>
</table>

To have a visual check that the annealing proceeded as expected, you can have a look at the temperature and the physical energy to check they correctly stabilized. You can use any program to visualize this, in particular using gnuplot the commands needed to plot TEMPP (column 3) and ECLASSIC (column 5), are:

```bash
gnuplot
set multiplot layout 1, 2
```

* AVERAGED QUANTITIES *

<table>
<thead>
<tr>
<th>MEAN VALUE (&lt;x&gt;)</th>
<th>DEVIATION (&lt;x^2&gt;-&lt;x&gt;^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IONIC TEMPERATURE</td>
<td>26.38</td>
</tr>
<tr>
<td>DENSITY FUNCTIONAL ENERGY</td>
<td>-45.672099</td>
</tr>
<tr>
<td>CLASSICAL ENERGY</td>
<td>-45.544157</td>
</tr>
<tr>
<td>NOSE ENERGY ELECTRONS</td>
<td>0.000000</td>
</tr>
<tr>
<td>NOSE ENERGY IONS</td>
<td>0.000000</td>
</tr>
<tr>
<td>ENERGY OF CONSTRAINTS</td>
<td>0.000000</td>
</tr>
<tr>
<td>ENERGY OF RESTRAINTS</td>
<td>0.000000</td>
</tr>
<tr>
<td>BOGOLOUBOV CORRECTION</td>
<td>0.000000</td>
</tr>
<tr>
<td>ION DISPLACEMENT</td>
<td>1.27023</td>
</tr>
<tr>
<td>CPU TIME</td>
<td>35.9728</td>
</tr>
</tbody>
</table>
Here you can see an example of the evolution of these two quantities during the test:

Test - NVE MD

In general, it is a good idea to verify that the final configuration obtained after the annealing is a physically 'reasonable' minimum energy configuration and that the BO MD has not brought the system in a very improbable configuration. A good test is to run a simulation in an NVE ensemble monitoring temperature (\textit{TEMPP}) and physical energy (\textit{ECLASSIC}): if after some steps these two quantities stabilize, then it is possible to be confident that the \textsc{RESTART.1} file previously obtained contains a good minimum energy structure. On the other hand, if energy and/or temperature continuously increase, that means that a good structure has not yet been obtained, and another annealing procedure is required, usually starting from a different configuration (for example, after heating the system at 300 K in order to move the system away from that “wrong” energy potential basin), or changing the annealing parameters (for example the annealing factor in \textsc{ANNEALING IONS}).

The test can be performed by the following procedure:

- Create a new folder and copy in it all the files needed (we will modify the input file used for the annealing):

```bash
mkdir ../7_test
cd ../7_test
cp ../6_annealing/mimic.tpr .
cp ../6_annealing/RESTART.1 ./RESTART
cp ../6_annealing/annealing.inp ./test.inp
```
• Modify test.inp file so that the &CPMD section appears as:

```plaintext
&CPMD
  RESTART COORDINATES VELOCITIES WAVEFUNCTION
  MIMIC
  MOLECULAR DYNAMICS BO
  NEW CONSTRAINTS
  ISOLATED MOLECULE
  TIMESTEP
    10.0
  MAXSTEP
    1000
  TRAJECTORY SAMPLE
    0
&END
```

Update the path in the PATH keyword in the &MIMIC section - no white spaces!

• Run the test

```plaintext
mpirun -n 2 cpmd.x test.inp path-to-tutorial-folder/AcetoneTutorial/data/pseudo > test.out
→ test.out : -n 1 gmx mdrun -deffnm mimic -ntomp 1 &
```

• Monitor the simulation:

```plaintext
tail -f ENERGIES
```

• When the simulation is completed, you can have a look at the temperature and the physical energy to check they correctly stabilized. You can use any program to visualize this, in particular using gnuplot the commands needed to plot TEMPP (column 3) and ECLASSIC (column 5), are:

```plaintext
gnuplot
```

```plaintext
set multiplot layout 1, 2
set title "Temperature"
set xlabel "MD step"
set ylabel "Temperature (K)"
plot 'ENERGIES_1000' u 1:3 w l lt 1 title "TEMPP"
set title "Physical Energy"
set xlabel "MD step"
set ylabel "Energy (a.u.)"
plot 'ENERGIES_1000' u 1:5 w l lt 2 title "ECLASSIC"
```

Here you can see an example of the evolution of these two quantities during the test:
Heating

If the test is successful (or if you skipped it), we can come back to the configuration obtained by the annealing procedure and start heating the system up to the room temperature. One way to do this is to increase the target temperature of a thermostat (coupled to the system) linearly at each step by performing a usual BO MD. A simple Berendsen-type thermostat can be used in the heating phase: it does not fully preserve the correct canonical ensemble but we are not interested to this feature at this stage, while it is numerically fast and more stable than alternative algorithms.

You can heat the system by performing the following procedure:

- Create a new folder and copy in it all the files needed (we will modify the input file used for the annealing):

```
mkdir ../8_heating
cd ../8_heating
cp ../6_annealing/mimic.tpr .
cp ../6_annealing/RESTART.1 ./RESTART
cp ../6_annealing/annealing.inp ./heating.inp
```

- Modify `heating.inp` file so that the `&CPMD` section appears as:

```
&CPMD
RESTART COORDINATES VELOCITIES WAVEFUNCTION
MIMIC
MOLECULAR DYNAMICS BO
NEW CONSTRAINTS
ISOLATED MOLECULE
TEMPERATURE RAMP
T_init 340.0 20
BERENDSEN IONS
T_init 5000
TIMESTEP
10.0
```

(continues on next page)
Update the path in the `PATH` keyword in the `&MIMIC` section - no white spaces! Modify the `T_init` value in the input file: With respect to the input file used to perform the test, two additional keywords are added in the `&CPMD` section: 1. `TEMPERATURE` with the option `RAMP`, where 3 numbers have to be specified on the line below the keyword: initial temperature (`T_init`, in K), target temperature (in K) and the ramping speed (in K per atomic time unit), to get the change per time step you have to multiply it with the value of `TIMESTEP`. In particular, we set a target temperature slightly higher than our real target of 300 K, since the thermostat may require a long time before actually reaching the target temperature specified. 2. `BERENDSEN` with the option `IONS`, where 2 numbers have to be specified on the line below the keyword: the target temperature for the termmostat, which in our case is the initial one (`T_init`) and the time constant of the thermostat (in a.u.). The suggested value of 5000 a.u., corresponding to ~0.12 ps, is a reasonable value for the system.

The value of `T_init` corresponds to the final temperature of the annealing procedure, from which we are starting the heating. You can read it from the `ENERGIES` file generated in the annealing (third column) using the command `tail -1 ../6_annealing/ENERGIES`, and set it accordingly in the `heating.inp` file.

- Run the heating

```
mpirun -n 2 cpmd.x heating.inp path-to-tutorial-folder/AcetoneTutorial/data/pseudo > heating.out : -n 1 gmx mdrun -deffnm mimic -ntomp 1 &
```

- Monitor the temperature during the simulation (third column):

```
tail -f ENERGIES
```

- If the desired temperature is not reached at the end of the simulation, you can continue the heating adding two keywords in the `&CPMD` section in the `RESTART` line of `heating.inp`:

  1. `LATEST`, which ensures to use the lastest written `RESTART` file (whose name is contained in the `LATEST` file)

  2. `ACCUMULATORS`, which ensures that new energy data and trajectory will be appended to your existing files.

Run the job in the same working directory as before.

```
RESTART COORDINATES VELOCITIES WAVEFUNCTION LATEST ACCUMULATORS
```

- When the simulation is completed, you can have a look at how the different quantities evolved during the heating phase. In particular, you can check how the temperature rises. You can use any program to visualize this, in particular using `gnuplot` the commands needed to plot `TEMPP` (column 3) are:

```
gnuplot

```set title "Temperature"
set xlabel "MD step"
set ylabel "Temperature (K)"
plot 'ENERGIES' u 1:3 w l lt 1 title "TEMPP"
```

Here you can see an example of the rise in temperature during the heating. In particular, to obtain this plot we run a longer heating than the `MAXSTEP` we set previously, and we had to to restart it from time to time to be able to reach the desired temperature of 300K (as explained above):
Note that the heating phase can be challenging: it is not always easy to get the temperature to stabilize to the desired value. The temperature specified in the RAMP will be eventually reached, but it may require a large amount of time, i.e. many more steps than the ones performed. What you can do in practice, if your system seems to be stabilized at a lower temperature than desired, is to restart the heating procedure from that intermediate point. You will then set the temperature for that configuration as $T_{\text{init}}$, and you can try to set a higher target temperature or a faster ramping speed.

Anyway, even if to run your own system you will of course need to complete the heating phase, for this tutorial we suggest to move to the next section even if you did not manage to obtain a well equilibrated configuration at QM/MM level: we will provide one for you, so you can see also how to start a production run.

**Production Run**

Now that the system is well equilibrated, we are finally ready to run a QM/MM MD at room conditions. You can run the production by performing the following procedure:

- Create a new folder and copy in it all the files needed (we will modify the input file used for the heating):

  ```bash
  mkdir ../9_production
  cd ../9_production
  cp ../8_heating/mimic.tpr .
  cp ../8_heating/heating.inp ./prod.inp
  cp ../8_heating/RESTART.1 ./RESTART
  ```

- Modify `prod.inp` file so that the &CPMD section appears as:

  ```
  &CPMD
  RESTART COORDINATES VELOCITIES WAVEFUNCTION GEOFILE
  MIMIC
  MOLECULAR DYNAMICS BO
  ```

(continues on next page)
Update the path in the PATH keyword in the &MIMIC section - no white spaces!

With respect to the input file used to perform the heating, we made some modifications:

1. The GEOFILE option has been added to the RESTART keyword. To perform the restart, this option will read old ionic positions and velocities from the file GEOMETRY. Note that a RESTART file need to be present as well, to read informations about the system not present in the geometry file, such as the atom elements.

2. The keywords DIPOLE DYNAMICS have been added which captures dipole information during the production run every NSTEP iteration in MD and saves it in an output file named DIPOLE. The NSTEP value is read from the next line if the keyword SAMPLE is present. But without this keyword and value, the default is 1 (every time step). Here we simply capture the dipole information at every step. We will use the data stored in the DIPOLE file in Dipole Calculation - method 1 section.

3. BERENDSEN thermostat for ions has been replaced with NOSE, corresponding with Nose-Hoover chains. As mentioned before, Berendsen a fast and stable thermostat, but does not properly samples the canonical ensemble. On the other hand, Nose-Hoover preserves the Maxwell distribution of the velocities and allows sampling the correct canonical ensemble, providing a NVT ensemble for a system in equilibrium. After the NOSE IONS keyword, 2 numbers are specified: the target temperature for the termostat, which in our case is 300 K, and the thermostat frequency in cm⁻¹, here 4000 cm⁻¹. Concerning the choice of the thermostat frequency, at which the energy transfer from/to the thermostat happens, it is important not to select a resonance vibrational frequency of your system.

4. The MAXSTEP has been increased to 10000 steps. Considering the timestep of choise (10 a.u.), this corresponds to a total simulation time of 100000 a.u., i.e. ~2.4 ps.

5. The trajectory is stored every 100 steps (TRAJECTORY SAMPLE option), and during the simulation 10 restart files will be saved. This is selected choosing to update the RESTART file every 1000 steps (STORE options), and saving 10 distinct RESTART files (RESTFILE option). In this way, a sequence of files RESTART.1, RESTART.2, ..., RESTART.10 will be produced during the dynamics. We will calculate the dipole moment of the acetone molecule for each of these configurations.

• We want to start the production from a reasonably heated configuration. This can be done extracting the coordinates and velocities of a step of choice, where the temperature has reached the desired value. We will save this information in a GEOMETRY file, which will be used as starting point thanks to the GEOFILE option we added to the RESTART keyword.

Even in case you did not manage to obtain a well-equilibrated structure and you will use the GEOMETRY file that we provide as starting point for the production (instructions below) it is useful to read the process described here to extract
a GEOMETRY file from a TRAJECTORY since you will be likely perform this step when working with MiMiC with your systems.

We provide a useful script to do that, called geofile_extract.py, even if this can in principle be easily done manually from the TRAJECTORY file (but it is not so practical with large systems). In order to extract the coordinates and velocities, we need to choose a step at which doing it. We can do it using the ENERGIES file produced during the heating process. It is important to keep in mind that how often coordinates and velocities have been stored. In our case, we set TRAJECTORY SAMPLE 1, i.e. we stored coordinates and velocities each steps, so we are sure every step we see in the ENERGIES file will have a corresponding set of positions and velocities in the TRAJECTORY file (but this may not be the case). Inspect the ENERGIES file of the heating phase and select a step number (first column) where the temperature (third column) is around 300K

vi ../8_heating/ENERGIES

You will see something similar to

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Energy</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>3468</td>
<td>0.000000</td>
<td>-44.654326513</td>
<td>-43.281408611</td>
</tr>
<tr>
<td>3461</td>
<td>0.000000</td>
<td>-44.654381345</td>
<td>-43.281433649</td>
</tr>
<tr>
<td>3462</td>
<td>0.000000</td>
<td>-44.654929512</td>
<td>-43.281886524</td>
</tr>
<tr>
<td>3463</td>
<td>0.000000</td>
<td>-44.654516592</td>
<td>-43.281777503</td>
</tr>
<tr>
<td>3464</td>
<td>0.000000</td>
<td>-44.659494353</td>
<td>-43.281703457</td>
</tr>
<tr>
<td>3465</td>
<td>0.000000</td>
<td>-44.659494858</td>
<td>-43.281654974</td>
</tr>
<tr>
<td>3466</td>
<td>0.000000</td>
<td>-44.657542937</td>
<td>-43.281683519</td>
</tr>
<tr>
<td>3467</td>
<td>0.000000</td>
<td>-44.658778661</td>
<td>-43.281625821</td>
</tr>
<tr>
<td>3468</td>
<td>0.000000</td>
<td>-44.659784323</td>
<td>-43.281709241</td>
</tr>
<tr>
<td>3469</td>
<td>0.000000</td>
<td>-44.660327312</td>
<td>-43.281597549</td>
</tr>
<tr>
<td>3470</td>
<td>0.000000</td>
<td>-44.660867919</td>
<td>-43.281680067</td>
</tr>
<tr>
<td>3471</td>
<td>0.000000</td>
<td>-44.661014498</td>
<td>-43.281778036</td>
</tr>
<tr>
<td>3472</td>
<td>0.000000</td>
<td>-44.660929834</td>
<td>-43.281685442</td>
</tr>
<tr>
<td>3473</td>
<td>0.000000</td>
<td>-44.659819427</td>
<td>-43.281552342</td>
</tr>
<tr>
<td>3474</td>
<td>0.000000</td>
<td>-44.659891785</td>
<td>-43.281436718</td>
</tr>
<tr>
<td>3475</td>
<td>0.000000</td>
<td>-44.659944852</td>
<td>-43.281527436</td>
</tr>
<tr>
<td>3476</td>
<td>0.000000</td>
<td>-44.659808283</td>
<td>-43.281614893</td>
</tr>
<tr>
<td>3477</td>
<td>0.000000</td>
<td>-44.659771974</td>
<td>-43.281683256</td>
</tr>
<tr>
<td>3478</td>
<td>0.000000</td>
<td>-44.658536628</td>
<td>-43.281586316</td>
</tr>
<tr>
<td>3479</td>
<td>0.000000</td>
<td>-44.658376691</td>
<td>-43.281476389</td>
</tr>
<tr>
<td>3480</td>
<td>0.000000</td>
<td>-44.652008380</td>
<td>-43.281561146</td>
</tr>
<tr>
<td>3481</td>
<td>0.000000</td>
<td>-44.658086971</td>
<td>-43.281451494</td>
</tr>
<tr>
<td>3482</td>
<td>0.000000</td>
<td>-44.654808885</td>
<td>-43.281359274</td>
</tr>
<tr>
<td>3483</td>
<td>0.000000</td>
<td>-44.654779094</td>
<td>-43.281262446</td>
</tr>
<tr>
<td>3484</td>
<td>0.000000</td>
<td>-44.654083779</td>
<td>-43.281166348</td>
</tr>
<tr>
<td>3485</td>
<td>0.000000</td>
<td>-44.654688861</td>
<td>-43.281099031</td>
</tr>
<tr>
<td>3486</td>
<td>0.000000</td>
<td>-44.653819367</td>
<td>-43.281047580</td>
</tr>
<tr>
<td>3487</td>
<td>0.000000</td>
<td>-44.653841883</td>
<td>-43.281374633</td>
</tr>
<tr>
<td>3488</td>
<td>0.000000</td>
<td>-44.662514440</td>
<td>-43.281459238</td>
</tr>
<tr>
<td>3489</td>
<td>0.000000</td>
<td>-44.662591999</td>
<td>-43.281348429</td>
</tr>
<tr>
<td>3490</td>
<td>0.000000</td>
<td>-44.662562754</td>
<td>-43.281342645</td>
</tr>
<tr>
<td>3491</td>
<td>0.000000</td>
<td>-44.662515229</td>
<td>-43.281318385</td>
</tr>
<tr>
<td>3492</td>
<td>0.000000</td>
<td>-44.681984335</td>
<td>-43.281402027</td>
</tr>
<tr>
<td>3493</td>
<td>0.000000</td>
<td>-44.681558512</td>
<td>-43.281294185</td>
</tr>
<tr>
<td>3494</td>
<td>0.000000</td>
<td>-44.631168356</td>
<td>-43.281376185</td>
</tr>
<tr>
<td>3495</td>
<td>0.000000</td>
<td>-44.608762843</td>
<td>-43.281263061</td>
</tr>
<tr>
<td>3496</td>
<td>0.000000</td>
<td>-44.608377675</td>
<td>-43.281293095</td>
</tr>
<tr>
<td>3497</td>
<td>0.000000</td>
<td>-44.649497732</td>
<td>-43.281248973</td>
</tr>
<tr>
<td>3498</td>
<td>0.000000</td>
<td>-44.649818620</td>
<td>-43.281235224</td>
</tr>
<tr>
<td>3499</td>
<td>0.000000</td>
<td>-44.649846884</td>
<td>-43.281214590</td>
</tr>
<tr>
<td>3500</td>
<td>0.000000</td>
<td>-44.648586165</td>
<td>-43.281195636</td>
</tr>
</tbody>
</table>

In this example, step 3477 seems a good choice as it is near 300K and shows stability in that range for a while. You could manually extract the coordinates and the TRAJECTORY file, and convert it in the GEOMETRY format. Otherwise, you can use the geofile_extract.py from the MiMiC Helper Script GitLab folder to extract the corresponding geometry.

wget https://gitlab.com/MiMiC-projects/mimic_helper/-/raw/main/scripts/geofile_extract.py
python3 geofile_extract.py ../8_heating/TRAJECTORY ../8_heating/ENERGIES

8.3. QM/MM MD with MiMiC
This will extract step XXX (which we would fill in as 3477 in our example) from the TRAJECTORY file, and check the corresponding temperature in the ENERGIES file. You will see in the output information about the temperature of the step selected, and the location of the corresponding geometry file.

This will generate a geometry file GEO_TTTK, where TTT is the temperature of the configuration. Copy the extracted geometry file, renaming it GEOMETRY, in the production run folder to use it as starting point for the production run

```bash
cp ../8_heating/GEO_TTTK ./GEOMETRY
```

- In case you did not manage to obtain a well-equilibrated configuration at QM/MM level, you can start the production run using the GEOMETRY provided. Copy that in the current folder with the following command

```bash
cp ../../data/GEO_295K ./GEOMETRY
```

- Now we have all the elements to run the production

```bash
mpirun -n 2 cpmd.x prod.inp path-to-tutorial-folder/AcetoneTutorial/data/pseudo > prod.out : -n 1 gmx mdrun -deffnm mimic -ntomp 1 &
```

- You can monitor different quantities of interest during the production run, for example the temperature (TEMPP, third column) and the physical energy (ECLASSIC, fifth column) of the system:

```bash
tail -f ENERGIES
```

- When the simulation is completed, you can have a look at how these two quantities evolved during the production phase. You can use any program to visualize this, in particular using gnuplot the commands needed to plot TEMPP (column 3) are: TEMPP (column 3) and ECLASSIC (column 5), are:

```bash
gnuplot
```

```bash
set multiplot layout 1, 2
set title "Temperature"
set xlabel "MD step"
set ylabel "Temperature (K)"
plot 'ENERGIES' u 1:3 w l lt 1 title "TEMPP"
set title "Physical Energy"
set xlabel "MD step"
set ylabel "Energy (a.u.)"
plot 'ENERGIES' u 1:5 w l lt 2 title "ECLASSIC"
```

Here you can see an example of the evolution of these two quantities over the first 5000 steps of the production phase:
A very good practice when you perform a simulation is to look at your trajectory through some visualization tool. This in particular can be extremely helpful when you notice some strange behavior of some physical quantity: the most of the problems are immediately identified by visual inspection. To visualize a the trajectory you can for example use VMD. There are several options to do so:

1. By setting the option `TRAJECTORY XYZ` in the &CPMD section the output file `TRAJEC.xyz` is produced. This can be directly loaded into VMD

   ```
   vmd TRAJEC.xyz
   ```

2. In principle, all the information needed by VMD are present in two distinct files: the `GEOMETRY.xyz` file contains the information about the atom types, and the `TRAJECTORY` file contains the coordinates at different timesteps. You can load the `GEOMETRY.xyz` file in VMD and input in it the `TRAJECTORY`.

   ```
   vmd GEOMETRY.xyz
   ```

   Then from the VMD menu right click on the `GEOMETRY.xyz` molecule, select Load Data Into Molecule and load the `TRAJECTORY` file, selecting CPMD in the Determine file type dropdown menu. Be aware that in this way the first frame will be the configuration in the `GEOMETRY.xyz`, while the actual trajectory will be visualized from.

3. If you did not set the option `TRAJECTORY XYZ` in the &CPMD section, you can always convert the `TRAJECTORY` file into the `xyz` format, making it easier the loading into VMD. For example, you can use the python script `traj_xyz.py` in the MiMiC Helper Script GitLab folder.
wget https://gitlab.com/MiMiC-projects/mimic_helper/-/raw/main/scripts/traj_xyz_convert.py
python3 traj_xyz_convert.py TRAJECTORY GEOMETRY.xyz

this will generate a TRAJECTORY.xyz file in the current folder. This can easily loaded into VMD as in option 1.

**Dipole Calculation**

As a last step we want to evaluate the dipole moment of acetone in water from QM/MM simulations. In contrast to what was done in the *Dipole Moment in Vacuum with CPMD* section, now our estimate for the dipole moment of acetone will take into account both the temperature and entropic effects due to the solvent environment. In gas phase we observed a dipole moment around 1.2 a.u. or 3.1 Debye.

We propose two methods to compute the dipole moment, you can try both of them and check which one agrees better with published values for solvated acetone at room temperature. Some published values are between 1.5-2.0 a.u. (see for example this study by Pereyra, R. G., et al.).

**Dipole Calculation - method 1**

One way to monitor the dipole moment over the production is by analyzing the data in the DIPOLE output file (produced thanks to the keyword DIPOLE DYNAMICS added to the input file). The first column of this file contains the step, columns 2 to 4 are the electronic, ionic and total contribution to the dipole moment. Due to later changes in CPMD code, columns 5 to 7 contain the same information of columns 2 to 4. *All the dipole moments are divided by the volume of the quantum box.*

We can analyze the dipole moment along our QM/MM MD simulations with a simple procedure (note that all this operation can be easily done manyally, but for simplicity we provide command lines to perform these mathematical calculations):

- Compute the QM box volume from the values in the CELL keyword of the &CPMD section

```bash
grep 'CELL' prod.inp -A 1 | tail -1 | awk '{print $1*$1*$2*$1*$3}'
```

where the A, B/A, and C/A sides of the box are extracted from the line following the keyword CELL (by finding it using `grep` and obtaining the following line with `tail`), and are summed together using the command `awk`.

- Average the values of the second column, multiplied by the cell volume. The following command line will print out the average and the standard deviation of the second column of the file DIPOLE. Run the following line, substituting the volume of the cell box found before to XXX

```bash
awk '{CELL_VOLUME=XXX}{if($2!=""){count++;sum+=$(2*CELL_VOLUME)}};y+=($2*CELL_VOLUME)^2} END{y=sqrt((y)/NR-(sum/NR)^2);sq=sq?sq:0;print "Mean = "sum/count ORS "S.D = ",sq}' DIPOLE
```

Now, you can compare the dipole moments obtained in gas phase (Section *Dipole Moment in Vacuum with CPMD*) and in solution (from BO QM/MM-MD). In gas phase we observed a dipole moment around 1.2 a.u. or 3.1 Debye.
Dipole Calculation - method 2

Another way to calculate the dipole is the following procedure, where we want to evaluate the dipole moment from the information previously collected during the production run, and subsequently to calculate their average value:

- Create a new folder and copy in it all the files needed (we will modify the input file used for the annealing):

```
mkdir ../10_dipole
cd ../10_dipole
cp ../9_production/mimic.tpr .
cp ../9_production/prod.inp ./dipole.inp
```

- Modify dipole.inp file so that the &CPMD section appears as:

```
&CPMD
MIMIC
RESTART WAVEFUNCTION COORDINATES LATEST
PROPERTIES
RESTFILE
0
&END
```

where the keyword RESTFILE with value 0 guarantees that CPMD does not write any RESTART file at the end of the dipole calculation, which is not needed now, and avoids to overwrite any RESTART file copied from the 9_production folder. To perform a dipole calculation you also need to add the following section (as we did before to calculate the dipole moment in vacuum):

```
&PROP
DIPOLE MOMENT
&END
```

Update the path in the PATH keyword in the &MIMIC section - no white spaces!

- The 10 calculations will be performed using each RESTART.X file, and this can be easily done editing the LATEST file (this is why the LATEST option has been added). For this reason, before running each dipole calculation, you need to replace the name of the RESTART.X in the first line of the LATEST file. This can be easily done with the suggested command:

```
for i in {1..10}; do
    ln -fs ../9_production/RESTART.$i RESTART
    mpirun -n 2 cpmd.x dipole.inp path-to-tutorial-folder/AcetoneTutorial/data/→pseudo > dipole_$i.out : -n 1 gmx mdrun -deffnm mimic -ntomp 1
done
```

- When the last process is completed, you can extract each value of the dipole moment from the output files by looking at each output file:

```
grep -A 5 DIPOLE dipole_X.out | tail -6 | head -4
```

Now, you can compare the dipole moments obtained in gas phase (Section Dipole Moment in Vacuum with CPMD) and in solution (from BO QM/MM-MD). In gas phase we observed a dipole moment around 1.2 a.u. or 3.1 Debye.
This is the “handbook” to the MiMiCPy package, functioning as an introduction and quick reference manual. MiMiCPy allows for quick input preparation of MiMiC input scripts, and it is used throughout the rest of the MiMiC tutorial.

Main feature described here:

1. Main features of MiMiCPy PrepQM subcommand
2. Boundary atoms selection in MiMiCPy
3. VMD/PyMOL support for PrepQM subcommand
4. Main features of FixTop subcommand
5. Main features of CPMDid subcommand
6. Brief explanation of CPMD2Coords and Geom2Coords
7. Accessing MiMiCPy as a Python library

This tutorial assumes the user to have running versions of MiMiC, CPMD and GROMACS.

- Instructions to install the MiMiC Framework (the MiMiC library and the MiMiC communication library) can be found in the Installation/MiMiC Framework section.
- Instructions to install CPMD and GROMACS with MiMiC enabled can be found in the Installation/External Programs section.
- Instructions to install MiMiCPy can be found in the Installation/MiMiCPy section.

Each section of this tutorial is dedicated to a different subcommand and is in principle independent from the others. Based on your needs, after the Example system: Acetone in Water section you can start from any of the following ones.

Main contributors to this handbook:

- Andrea Levy
- Bharath Raghavan
- Sophia Johnson
9.1 Example System: Acetone in Water

For this tutorial we will assume that you downloaded the corresponding git repository, hence you will have the provided folders and files in the same locations.

curl https://gitlab.com/MiMiC-projects/MiMiC-docs/-/archive/main/MiMiC-docs-main.tar.gz?path=docs/tutorials/MiMiCPyTutorial -o MiMiCPyTutorial.tar.gz
tar --strip-components=3 -zxvf MiMiCPyTutorial.tar.gz

The above commands downloaded a copy of the exercise repository MiMiCPy in the current folder. Navigate the repository and create your own directory to run this exercise:

```bash
cd MiMiCPyTutorial
mkdir solution
```

The structure and topology files for acetone are provided in the data folder. Copy this file to the current folder:

```bash
cp ../data/acetone_equilibrated.gro .
cp ../data/acetone_equilibrated.top .
cp ../data/acetone.itp .
```
9.2 PrepQM

As the name suggests, prepqm is the main tool to prepare MiMiC QM/MM input files. In short, it accepts MM topology and coordinate files, and after inputting the QM selection, outputs the CPMD input file and GROMACS index file necessary to create the .tpr run file.

Create a folder in the MiMiCPyTutorial/solution directory where you will test the PrepQM command with different keywords:

```bash
mkdir prepqm
cd prepqm
```

9.2.1 Minimal input

The minimal input for PrepQM consists of the MM topology (.top file) and the coordinates (.gro or .pdb files). Run the MiMiCPy prepqm script using the files provided:

```bash
mimicpy prepqm -top ../acetone_equilibrated.top -coords ../acetone_equilibrated.gro
```

When running MiMiCPy PrepQM command, make sure GROMACS has been sourced, i.e. is available on the command line. In particular, you may need to set the environment variable `GMXLIB` with the following command, where the path to use the one where the GROMACS share/gromacs folder is located

```bash
export GMXLIB='/path-to-gromacs-installation/share/gromacs/'
```

The PrepQM command will open an interactive session with the following message printed out (more details on the interactive selection environment can be found in the MiMiCPy documentation):

```
***** MiMiCPy *****
For more information type mimicpy [subcommand] --help

=====> Running prepqm <=====

**Reading topology**

Some atom types had no atom number information. They were guessed as follows:

+---------------------+
| Atom Type | Element |
+---------------------+
| c          | C       |
+---------------------+
| c3         | C       |
+---------------------+
| o          | O       |
+---------------------+
| hc         | H       |
```

(continues on next page)
MiMiCPy requires the atomic element information for each atom to generate the CPMD input script. Some non-standard atom types in the GROMACS topology file do not contain this information (atom types c, c3, o, and hc). MiMiCPy guesses them based on the combination of atomic mass, name and type. Always verify that the guessed atomic elements for each atom type are meaningful, as it is essential for CPMD to run correctly.

MiMiCPy guesses non-standard atom types based on the combination of atomic mass, name and type. The automatic behavior of guessing atomic elements can be toggled on or off using the -guess option. If False is passed and non-standard atoms are present, MiMiCPy will exit with an error message instead of attempting to guess the elements. If you are not satisfied with the element information guessed, a file containing the list of all non-standard atom types with the correct element information can be passed to `mimicpy prepqm` with the -nsa option. For example, if MiMiCPy guessed the wrong atomic number for atom type c3 and o, we could create a file called atomtypes.dat with the following text:

```
0 0
C3 C
```

and pass it to MiMiCPy when running prepqm

```
mimicpy prepqm -top acetone_equilibrated.top -coords acetone_equilibrated.gro -nsa
˓→atomtypes.dat
```

At this point, the user can provide instructions to add and/or delete atoms to the QM region in an interactive session. The instructions specified after add correspond to the selection query that identifies the atoms to be added to the QM region. MiMiCPy provides a selection language similar to PyMOL and VMD for selecting QM atoms. In this example, we want to add the acetone molecule to the QM region, hence we’ll select the ACT residue with the following command:

```
> add resname is ACT
```

The syntax for the selection query involves the following general structure:

```
> keyword logical_operator value
```

where `keyword` can include `resname`, `resid`, `name`, `type`, `id`, and `mol`. `logical_operator` can be `is`, `not`, `>`, `<`, `>=`, or `<=`.

Different selection queries can be joined by using `and` or `or` operators, and grouped with brackets, such as

```
> add (resname is RES) and (name is X)
```

which will add atom named X of residue RES. With the same selection syntax atoms can also be deleted from the QM region, using the `delete` command instead of `add`.

To check the current QM region, the `view` command can be used. When the desired QM atoms are selected, type `q` to exit.

More details on the interactive selection environment can be found in the MiMiCPy documentation.

We can check the current QM region to console using `view`
which will produce the following output

<table>
<thead>
<tr>
<th>type</th>
<th>resid</th>
<th>is_bound</th>
<th>resname</th>
<th>name</th>
<th>charge</th>
<th>element</th>
<th>mass</th>
<th>mol</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>C1</td>
<td>0.8102</td>
<td>C</td>
<td>12.0</td>
<td>ACT</td>
<td>1.212</td>
<td>1.264</td>
</tr>
<tr>
<td>2</td>
<td>c3</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>C2</td>
<td>-0.4767</td>
<td>C</td>
<td>12.0</td>
<td>ACT</td>
<td>1.222</td>
<td>1.135</td>
</tr>
<tr>
<td>3</td>
<td>c3</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>C3</td>
<td>-0.4767</td>
<td>C</td>
<td>12.0</td>
<td>ACT</td>
<td>1.326</td>
<td>1.362</td>
</tr>
<tr>
<td>4</td>
<td>o</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>O1</td>
<td>-0.5864</td>
<td>O</td>
<td>16.0</td>
<td>ACT</td>
<td>1.110</td>
<td>1.295</td>
</tr>
<tr>
<td>5</td>
<td>hc</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>H1</td>
<td>0.1216</td>
<td>H</td>
<td>1.0</td>
<td>ACT</td>
<td>1.197</td>
<td>1.046</td>
</tr>
<tr>
<td>6</td>
<td>hc</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>H2</td>
<td>0.1216</td>
<td>H</td>
<td>1.0</td>
<td>ACT</td>
<td>1.321</td>
<td>1.121</td>
</tr>
<tr>
<td>7</td>
<td>hc</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>H3</td>
<td>0.1216</td>
<td>H</td>
<td>1.0</td>
<td>ACT</td>
<td>1.154</td>
<td>1.135</td>
</tr>
<tr>
<td>8</td>
<td>hc</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>H4</td>
<td>0.1216</td>
<td>H</td>
<td>1.0</td>
<td>ACT</td>
<td>1.393</td>
<td>1.314</td>
</tr>
<tr>
<td>9</td>
<td>hc</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>H5</td>
<td>0.1216</td>
<td>H</td>
<td>1.0</td>
<td>ACT</td>
<td>1.299</td>
<td>1.457</td>
</tr>
<tr>
<td>10</td>
<td>hc</td>
<td>1</td>
<td>0</td>
<td>ACT</td>
<td>H6</td>
<td>0.1216</td>
<td>H</td>
<td>1.0</td>
<td>ACT</td>
<td>1.360</td>
<td>1.373</td>
</tr>
</tbody>
</table>

Now that we verified that all the atoms of the acetone molecule are correctly included in the QM region, you can terminate the interactive session using `quit` (or `q`)

which will terminate the interactive session and print the following output

```
Wrote Gromacs index file to index.ndx
Wrote new CPMD input script to cpmd.inp
MDP file not passed! Skipping generation of GROMACS TPR file
=====> Done <=====  
```

As stated in the last output, MiMiCPy has generated a GROMACS index file, named `index.ndx` by default (a custom name can be set using the output option `-ndx` when running `mimicpy prepqm`) and a CPMD input script, named `cpmd.inp` by default (custom name with `out` option).

The `index.ndx` file contains the GROMACS IDs of the QM atoms, in this case they are the first ten atoms in the topology:

```
; Generated by MiMiCPy
[ QMatoms ]
    1  2  3  4  5  6  7  8  9  10
```

The `cpmd.inp` file contains some basic sections necessary in a CPMD input in order to run QM/MM simulations with MiMiC. In particular, the section generated include `&MIMIC`, `&CPMD`, `&SYSTEM`, and `&ATOMS`.

To run a QM/MM simulation with MiMiC, a GROMACS binary `.tpr` file is also needed. This can be generated using GROMACS preprocessor (`gmx grompp`), using the `index.ndx` file generated by MiMiCPy. Copy the `.mdp` file provided and run GROMACS preprocessor.

```
  cp ../../data/mimic.mdp .
gmx grompp -f mimic.mdp -c ../acetone_equilibrated.gro -p ../acetone_equilibrated.top -n .._index.ndx -o mimic.tpr
```

The above GROMACS command generated the `mimic.tpr` file, which can be used by MiMiC to perform QM/MM MD.
The two steps that we have just performed, i.e. generating an index (and CPMD input) file with MiMiCPy and using that to run the GROMACS preprocessor, can be conveniently combined in a single MiMiCPy command. This, together with other useful keywords, will be shown in the next section.

9.2.2 PrepQM keywords

As mentioned above, the MM topology and coordinate files are the minimal inputs for PrepQM. However, for practical uses the output files generated will not be enough to directly start a MiMiC QM/MM simulation. In this section, some useful keywords are explained. For each of them, if an input file is needed, it is provided as example in the data folder. Those will be copied later and used to run PrepQM a second time, generating the necessary input to start a MiMiC QM/MM MD simulation.

• `-sele`: the list of selection commands (the add resname is ACT selection used before in the interactive session) can be saved in a file and provided as input.

• `-mdp`: the GROMACS MD parameters file can be passed and MiMiCPy will automatically generate the GROMACS .tpr file using the gmx grompp command (avoiding the two-step process used before). The output name for the .tpr file can be specified using the additional input option `-tpr`, and the name of the executable for running `grompp` can be passed with the `-gmx` option `gmx`, `gmx_mpi`, or `gmx_mpi_d`.

• `-pp`: pseudopotential information written in the &ATOMS section of the CPMD input is by default set to KLEINMAN-BYLANDER with LMAX=S, and needs to be manually corrected. Otherwise, pseudopotential information can be written in a file and passed using the `-pp` option. This file should contain information like the pseudopotential filenames for a specific atom, the same for the boundary atom version and other labels like LMAX and LOC. The data for each element is given in separate lines, and each line has the following format: `<element> <pp filename> <bound pp filename> <labels> <lmax> <loc>`, where a `-` can be used to skip any of this information.

• `-inp`: by default, `mimicpy prepqm` outputs a barebones CPMD script, where many sections need to be manually filled to be able to start a QM/MM simulation. Missing CPMD sections and commands can be saved in a template script, passed to PrepQM. These sections will be copied into the generated CPMD input.

• `-pad`: the size of the QM region is specified in the CPMD input with the CELL keyword in the &SYSTEM section. By default, MiMiCPy writes a cell size exactly bounding the QM atoms, which is often not enough to contain the plane waves of the QM region. For this reason, a padding needs to be added, i.e. an additional distance between the QM atoms and the walls of the QM box (specified in nm). A standard value for the Poisson solver is 3-3.5 Å, but in general the question of the right padding should be considered and chosen wisely depending on the system under study. MiMiCPy adds this on both sides in each direction; therefore, with a `-pad` of 0.35, a value of 0.7 nm is actually added in each direction to the size of the QM system.

• `-q`: by default, MiMiCPy calculates the total charge of the QM region (specified with the CHARGE keyword in the &SYSTEM section of the CPMD input) using the MM charges of the QM atoms from the force field data. This is usually enough, but it may need to be cross checked. Alternatively, a charge value to be written in the output can be specified using the `-q` option.

• `-path`: the path specified in the PATHS keyword of the &MIMIC section refers to the path of the directory from which GROMACS will be run, i.e. where the mimic.tpr file is located. By default, MiMiCPy uses the current directory. Alternatively, a path can be specified with the `-path` option.

The above keywords are the most useful ones when using MiMiCPy PrepQM in practice. Additional keywords are available, and can be found in the MiMiCPy PrepQM documentation.

Now that you are more familiar with the main PrepQM keywords, let’s run the `mimicpy prepqm` command to add them. The input files needed for this tutorial are provided.

```
cp ../../data/sele.dat .
cp ../../data/mimic.mdp .
```
When running MiMiCPy PrepQM command, make sure GROMACS has been sourced, i.e. is available on the command line. In particular, you may need to set the environment variable `GMXLIB` with the following command, where the path to use the one where the GROMACS share/gromacs folder is located:

```
export GMXLIB='/path-to-gromacs-installation/share/gromacs/'
```

The next command shows you how you can easily generate the GROMACS index and .tpr files, together with a complete CPMD input for MiMiC, using MiMiCPy. In particular, this time MiMiCPy generates the `cpmd_mimic.inp` CPMD input file (whose name has been specified with the option `-out`), which is a full CPMD input file ready to be used for running MiMiC (the input provided is in particular to run a BO MD in the NVE ensemble). Compare this input file with the previously generated `cpmd.inp`.

```
mimicpy prepqm -top ../acetone_equilibrated.top -coords ../acetone_equilibrated.gro -sele sele.dat -inp template.inp -pp pp_info.dat -pad 0.35 -out cpmd_mimic.inp -mdp -gmx gmx -tpr mimic.tpr
```

### 9.2.3 Boundary atoms

In the example the QM region was quite easy to select, in particular it was a single molecule (the acetone molecule) with no covalent bond with any other MM atom of the system. This is not always the case, especially when dealing with biological systems, where usually some of the amino acids are treated at QM level, while the rest of the protein is treated at MM level. The atoms at a boundary point between QM and MM regions, when a covalent bond is present between them, are referred to as **boundary** atoms. One approach to describe boundary atoms is to use the boundary-atom scheme, where open-valence QM atoms are described through a special optimized monovalent pseudopotential with the required valence change in place of a classical atom. In the CPMD input, ‘regular’ QM atoms and boundary atoms are in separate blocks of the &ATOMS section, since they are described by different pseudopotential.

The syntax for selecting boundary atoms in MiMiCPy PrepQM is similar to the one for other QM atoms, with the exception of the keyword used, which `add-bound` instead of just `add`, followed by the `<selection>`.

To give an example, we will partition our acetone molecule in a quite weird way, but this should give you a clear example of the different selection of ‘regular’ and boundary atoms. For clarity, we will just use the minimal input for PrepQM, i.e. a MM topology and a coordinate file.

```
mkdir boundaryatoms
cd boundaryatoms
mimicpy prepqm -top ../acetone_equilibrated.top -coords ../acetone_equilibrated.gro -sele sele.dat -inp template.inp -pp pp_info.dat -pad 0.35 -out cpmd_mimic.inp -mdp -gmx gmx -tpr mimic.tpr
```

Let’s pretend we want to treat the oxygen atom and one of the two CH3 groups at QM level, and the other CH3, together with the water molecules, at MM level. The C-C bond will be the covalent linkage between the QM and MM region and the C3 carbon will be the boundary atom.
A possible way to make this selection in PrepQM is the following:

```
> add name is O1
> add name is C1
> add name is C2
> add name is H1
> add name is H2
> add name is H3
> add-bound name is C3
```

where the first six commands add the atoms to the QM region, and the last one selects C3 as boundary atom.

With this selection, C1 and C2 carbons in the &ATOMS section of the generated CPMD input file will appear as

```
%C_MT_BLYP.psp KLEINMAN-BYLANDER
```

while C3 as

```
%C_MT_BLYP_BOUNDARY.psp KLEINMAN-BYLANDER
```

i.e. they will be treated with different pseudopotentials.

Be aware that the input generated in this illustrative example would cause you trouble if you try to use it for a QM/MM MD run. This is because with the splitting of the acetone molecule used to illustrate how to treat boundary atoms, the MM region contains some atoms from a non-standard residue, i.e. the atoms of acetone left outside the QM region. CPMD will not have the species information for these atoms, which can lead to segmentation fault errors. This can be solved using MiMiCPy FixTop command (illustrated in a separate section of this tutorial).
### 9.2.4 Boundary atom automatic guess

MiMiCPy can also automatically guess boundary atoms. This feature can be activated by setting the `-bound` option to `True` (default `False`). With this option, MiMiCPy can automatically mark the QM atoms at the QM/MM interface as boundary atoms.

As before, let’s pretend to want to treat O1, C1, C2, H1, H2, and H3 at QM level, without manually specifying the boundary atoms this time (where we use the `-ndx` and `-out` options to save the index and CPMD input files avoiding to overwrite the previously generated ones for comparison):

```bash
mimicpy prepqm -top ../../acetone_equilibrated.top -coords ../../acetone_equilibrated.gro -bound True -ndx index_boundtrue.ndx -out cpmd_boundtrue.inp
```

Where the `-bound` option is set to `True`.

Performing the same selection as before for the atoms in the QM atoms, i.e.

```
> add name is O1
> add name is C1
> add name is C2
> add name is H1
> add name is H2
> add name is H3
```

now yields to this output:

```
Boundary atoms were automatically set. The following atoms were marked as boundary atoms:

+----------------+
| Atom | Residue |
+----------------+
| 1 C1 | 1 ACT |
+----------------+

Wrote Gromacs index file to index_boundtrue.ndx
Total charge of QM region is 0.11189999999999997, Rounding to integer
Wrote new CPMD input script to cpmd_boundtrue.inp
MDP file not passed! Skipping generation of GROMACS TPR file

===== Done =====
```

The C1 carbon has been automatically detected and set as boundary atom. MiMiCPy generated two files: `index_boundtrue.ndx` and `cpmd_boundtrue.inp`. You should see no difference with the index.ndx and cpmd.inp files generated in the *Boundary atoms* section.

### 9.2.5 VMD and PyMOL support

MiMiCPy also provides plugins for the molecular visualization packages VMD and PyMOL. These can be optionally activated during the MiMiCPy installation process.

The next two sections will perform the same operations, once with VMD and once with PyMOL.

Make sure to have installed MiMiCPy enabling the VMD and/or PyMOL support during the installation procedure described in the Installation/MiMiCPy section.

MiMiCPy requires Python >= 3.5, pandas >= 0.24.0 and numpy >= 1.12.0. The plugins have been tested with PyMOL version 2.3.4 and VMD version 1.9.4a38, although other versions are expected to work. If any incompatibilities are
found, please post an issue on GitLab.

VMD

Open the provided acetone_equilibrated.gro structure with VMD

```
vmd ../../acetone_equilibrated.gro
```

The `prepqm` command can be used in the TK console of VMD. For this, what is needed is a selection of QM atoms (and any boundary atom, if present) to pass to the `prepqm` plugin. In this case, if you performed the preceding steps of this tutorial, the selection would be quite straightforward. However, a good practice is to create your selection as a graphical representation, visually inspect it to ensure everything is correctly selected, and proceed with the TkConsole.

In order to be able to see the different selections for ‘regular’ QM atoms and boundary atoms, as done in the Boundary atoms section, we will pretend to want to treat the oxygen atom and one of the two CH3 groups at QM level, and the other CH3, together with the water molecules, at MM level. The C-C bond will be the covalent linkage between the QM and MM region and the C3 carbon will be the boundary atom.

Open the Graphical Representations window from the VMD Main by clicking Graphics > Representations. ... You can create a new representation by clicking Create Rep. We want to include in this first representation the ‘regular’ QM atoms. You can do this by typing the following VMD selection as Selected Atoms

```
name O1 C1 C2 H1 H2 H3
```

An easy way to visually inspect if your selection corresponds to what you wanted is to change Drawing Method and/or Coloring Method. In this way you will immediately see the atoms that will be assigned to the QM region.

Add a second representation for the boundary atoms by clicking again Create Rep and now selecting

```
name C3
```

You can see below an example, where the atoms corresponding to the first selections are colored in red, while the boundary atom is colored in yellow.

Now that we know what atoms to select, it is possible to proceed with the `prepqm` plugin: open the TK Console by clicking Extensions > TK Console from the VMD Main window and type the following commands (which will
define the selections for the QM atoms and the boundary atom):

```plaintext
set sel [atomselect top "name is O1 C1 C2 H1 H2 H3"]
set sel_bound [atomselect top "name is C3"]
```

where `set` command creates a selection, followed by the name of the selection (sel or sel_bound in this case) and between square brackets we used the `atomselect` method followed by the `molecule_id` (top in this case, since we have only a molecule which is also the top one) and the `selection_text` (with the same VMD syntax as the one used in the Graphical Representations window).

Once the selections are created, it is possible to call the `prepqm` plugin in the TK Console and pass the selections

```plaintext
prepqm -top ../../acetone_equilibrated.top -sele $sel -molid 0 -sele_bound $sel_bound -
   →out cpmd_vmd.inp -ndx index_vmd.ndx
```

where `-sele` is the selection of the QM atoms, `-molid` is the molecule ID (you can see that from VMD menu), `-sele_bound` is the selection for the boundary atoms. The remaining commands are analogous the usual MiMiCPy PrepQM command we have used. Details about the syntax of the PrepQM plugin in VMD can be found in the MiMiCPy documentation.

You will see in the TK Console a similar output to the one shown when running PrepQM from terminal

```
Some atom types had no atom number information.
They were guessed as follows:

+---------------------+  
| Atom Type | Element |
+---------------------+  
| c           | C       |
+---------------------+  
| c3          | C       |
+---------------------+  
| o            | O       |
+---------------------+  
| hc           | H       |
+---------------------+  

Wrote Gromacs index file to index_vmd.ndx
Total charge of QM region is -0.3648, Rounding to integer
Wrote new CPMD input script to cpmd_vmd.inp
```

You can now close VMD and you will find in your folder the two new files generated: `index_vmd.ndx` and `cpms_vmd.inp`. You should see no difference with the `index.ndx` and `cpms.inp` files generated in the `Boundary atoms` section.

### PyMOL

Open the provided `acetone_equilibrated.gro` structure with PyMOL

```plaintext
pymol ../../acetone_equilibrated.gro
```

The `prepqm` command can be used in the console of PyMOL. For this, what is needed is a selection of QM atoms (and any boundary atom, if present) to pass to the `prepqm` plugin. In this case, if you performed the preceding steps of this tutorial, the selection would be quite straightforward. However, a good practice is to create your selection as a graphical representation, visually inspect it to ensure everything is correctly selected, and proceed with the selection via console.
In order to be able to see the different selections for ‘regular’ QM atoms and boundary atoms, as done in the *Boundary atoms* section, we will pretend to want to treat the oxygen atom and one of the two CH3 groups at QM level, and the other CH3, together with the water molecules, at MM level. The C-C bond will be the covalent linkage between the QM and MM region and the C3 carbon will be the boundary atom.

You can create a new selection by typing it in the PyMOL command line. We want to include in this first representation the ‘regular’ QM atoms. You can do this by typing the following PyMOL selection

```py
selection (name O1+C1+C2+H1+H2+H3)
set_name sele, qm_atoms
```

An easy way to visually inspect if your selection corresponds to what you wanted is to change how it is shown and/or colored (for example just by using `color qm_atoms, red` for the selection just created). In this way you will immediately see the atoms that will be assigned to the QM region.

Add a second selection for the boundary atoms as before, now selecting

```py
selection (name C3)
set_name sele, boundary_atoms
```

You can see below an example, where the atoms corresponding to the first selections are colored in red, while the boundary atom is colored in yellow.

Now that we know what atoms to select, it is possible to proceed with the `prepqm` plugin: type in the PyMOL command line the following command:

```bash
prepqm ../../acetone_equilibrated.top, qm_atoms, boundary_atoms, out=cpmd_pymol.inp, ...
ndx=index_pymol.ndx
```
where \texttt{qm_atoms} is the selection created for the QM atoms, \texttt{bound_atoms} is the one for the boundary atoms. The remaining commands are analogous the usual MiMicPy PrepQM command we have used. Details about the syntax of the PrepQM plugin in PyMOL can be found in the MiMiCPy documentation.

You will see in the PyMOL console a similar output to the one shown when running PrepQM from the command line.

Some atom types had no atom number information. They were guessed as follows:

<table>
<thead>
<tr>
<th>Atom Type</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>c3</td>
<td>C</td>
</tr>
<tr>
<td>o</td>
<td>O</td>
</tr>
<tr>
<td>hc</td>
<td>H</td>
</tr>
</tbody>
</table>

Wrote Gromacs index file to \texttt{index_pymol.ndx}

Total charge of QM region is -0.3648, Rounding to integer

Wrote new CPMD input script to \texttt{cpmd_pymol.inp}

You can now close PyMOL and you will find in your folder the two new files generated: \texttt{index_vmd.ndx} and \texttt{cpmd_vmd.inp}. You should see no difference with the \texttt{index.ndx} and \texttt{cpmd.inp} files generated in the Boundary atoms section.

### 9.3 FixTop

As mentioned at the end of the Boundary atoms section, not having species information for some of the atoms in the MM region will cause CPMD to crash because the atomic element for those atoms is not known by CPMD. This can happen when non-standard residues are included in the MM region and the corresponding topology files do not contain species information (which is usually the case since these are not needed by GROMACS, but by CPMD only) and is especially true if an MM atom, without species information, participates in a QM-MM bond. As the name suggests, FixTop command can be used to fix the topologies and is specifically designed for the aforementioned cases.

Create a folder in the MiMiCPyTutorial/solution directory where you will test the FixTop command:

```
mkdir fixtop
cd fixtop
```

As in the Boundary atoms section, let’s pretend that in the system composed of an acetone molecule in water we want to treat the oxygen atom and one of the two CH3 groups at QM level, and the other CH3, together with the water molecules, at MM level. The C-C bond will be the covalent linkage between the QM and MM region and the C3 carbon will be the boundary atom.
In particular, no species information is given for the atoms of the acetone molecule that want to be treated at MM level (C3,H4,H5,H6) in the GROMACS topology. Let’s copy the acetone topology in the current folder to have a look

```
cp ../../data/acetone_equilibrated.top .
cp ../../data/acetone.itp .
```

In GROMACS, species information are specified in the [ atomtype ] section of a force field parameters. In this example, AMBER03 force field is used to treat MM atoms (as specified in the .top file by the amber03.ff/forcefield.itp). However, parameters for acetone are not included. This is why we need the additional acetone.itp file, as usual when dealing with non-standard residues in GROMACS. This file contains additional parameters, but not the atomic species, which is needed by CPMD. These parameters are stored in the ffnonbonded.itp file of the amber force field.

It is good practice to create a local copy of the force-field directory to avoid overwriting the original. Based on the location of your GROMACS installation, the absolute path will differ, but force fields are located in the following directory: `<path-to-gromacs>/share/gromacs/top/` and in particular, this directory contains the amber03.ff/ folder. For this tutorial you can copy this folder in the current directory or simply use the file provided in the data folder (we only include the ffnonbonded.itp file, which is sufficient for this tutorial).

```
cp ../../data/ffnonbonded.itp .
```

This is how this file looks like:

```
[ atomtypes ]
; name at.num mass charge ptype sigma epsilon
H0  1  1.008 0.0000 A  2.47135e-01  6.56888e-02
Br  35 79.90 0.0000 A  3.95559e-01  1.33888e+00 ; Converted from parm99.dat
C   6  12.01 0.0000 A  3.39967e-01  3.59824e-01
```

Non-standard atomtypes are not contained in this file, and in particular this is the case for c3 and hc (the atom types for the carbon and hydrogen atoms we are interested in).
FixTop is able to guess the atomic species of all atoms where it is missing in the topology file (this includes those in MM region, which PrepQM will not fix), and prints a consolidated \texttt{[atomtypes]} section into a GROMACS \texttt{.itp} file.

Let’s fix this by running FixTop, here shown with a minimal input, i.e. only the topology

\begin{verbatim}
mimicpy fixtop -top acetone_equilibrated.top
\end{verbatim}

This command will print the following message on the console, informing the users about which atom types have been guessed by FixTop and which element has been assigned to them. It’s always good to doublecheck that the guess performed by MiMiCPy is correct.

\begin{verbatim}
***** MiMiCPy *****

For more information type mimicpy [subcommand] --help

=====> Running fixtop <=====

**Reading topology**

Some atom types had no atom number information. They were guessed as follows:

+---------------------+   
| Atom Type | Element |   
+---------------------+   
| c          | C       |   
+---------------------+   
| c3         | C       |   
+---------------------+   
| o          | O       |   
+---------------------+   
| hc         | H       |   
+---------------------+

**Writing fixed [atomtypes] section**

Fixed and replaced [atomtypes] section in atomtypes.itp

=====> Done <=====
\end{verbatim}

FixTop has generated a new file in your folder, atomtypes.itp. This will contain a [atomtypes] section for all the atoms in your topology, and in particular will have a new column with respect to the acetone.itp file that we have looked at before, at.num. This contains the atomic number for all the atomtypes, and is used by CPMD to understand what is the species of the MM atoms.

\begin{verbatim}
; name at.num mass charge ptype sigma epsilon
H0  1 1.0080 0.0000 A 2.471350e-01 6.568880e-02
Br 35 79.9000 0.0000 A 3.955590e-01 1.338880e+00
C 6 12.0100 0.0000 A 3.399670e-01 3.598240e-01
hc 6 0.0000 0.0000 A 3.399670e-01 4.577300e-01
\end{verbatim}

(continues on next page)
In particular, FixTop copied the information from the force field specified in the topology and appended the missing ones for non-standard atom types at the end.

### 9.3.1 Additional options

The easiest way to incorporate these new parameters into an existing GROMACS force-field is to write it directly to the `ffnonbonded.itp` file, containing the `[atomtypes]` definition of the whole system for all default GROMACS force-fields. FixTop is able to replace the `[atomtypes]` section in the file passed as input for the `-out` option if in addition the `-cls` option is set to `True`. It will also clear all other `[atomtypes]` sections from the topology file.

Run the following commands (where we will make a copy of the original `ffnonbonded.itp` file to be able to compare the one generated by FixTop:

```bash
cp ffnonbonded.itp ffnonbonded_original.itp
mimicpy fixtop -top acetone_equilibrated.top -out ffnonbonded.itp -cls True
```

The `ffnonbonded.itp` file will be updated, and in particular will contain four new lines corresponding to the atom types of acetone.

In practice you can create a local copy of the entire force field directory you are interested to use, update the topology including species information for all MM atoms and proceed using PrepQM with a complete topology for MM atoms.

### 9.4 CPMDid

Another very useful tool when working with MiMiC is `cpmdid`: it allows the user to print out the CPMD IDs of both QM and MM atoms, from a selection using the MM topology naming/numbering. This is especially helpful since MM IDs are reshuffled in a non-obvious way (grouping atoms according to species) when transferring data from GROMACS to CPMD. However, CPMD IDs for both QM and MM atoms need to be used in CPMD inputs, for example when applying constraints or adding multiple thermostats.

Create a folder in the `MiMiCPyTutorial/solution` directory where you will test the CPMDid command with different keywords:

```bash
mkdir cpmdid
cd cpmdid
```

Let's suppose for example that we want to know the CPMD IDs for the acetone molecule and one of the surrounding water molecules (residue 32 in this case). The MM IDs can be easily obtained from the topology, or from visual inspection of the structure with, for example, VMD.
CPMDid needs as minimal input a topology file and a CPMD input file from which the `OVERLAPS` section is read. If you skipped the first section of this tutorial, where the PrepQM tool is illustrated, or if you want to start from scratch, you can quickly generate a CPMD input with the following command:

```
cp ../../data/sele.dat .
mimicpy prepqm -top ../acetone_equilibrated.top -coords ../acetone_equilibrated.gro -sele sele.dat
```

where in this case we use a minimal input for PrepQM since we are only interested in correctly generating the `OVERLAPS` section, with the only addition of the `sele` option to avoid having to manually select the QM atoms from the interactive session.

When running MiMiCPy PrepQM command, make sure GROMACS has been sourced, i.e. is available on the command line. In particular, you may need to set the environment variable `GMXLIB` with the following command, where the path to use the one where the GROMACS share/gromacs folder is located:

```
export GMXLIB='/path-to-gromacs-installation/share/gromacs/
```

CPMDid can be now run using the following command:

```
mimicpy cpmdid -top ../acetone_equilibrated.top -inp ../cpmdid/cpmd.inp
```

The CPMDid command will open an interactive session with the following message printed out:

```
***** MiMiCPy *****

For more information type mimicpy [subcommand] --help
```

(continues on next page)
Some atom types had no atom number information. They were guessed as follows:

<table>
<thead>
<tr>
<th>Atom Type</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>c3</td>
<td>C</td>
</tr>
<tr>
<td>o</td>
<td>O</td>
</tr>
<tr>
<td>hc</td>
<td>H</td>
</tr>
</tbody>
</table>

Please enter selection below. Type q or quit to exit.

We can select the acetone molecule and the nearby water molecule (with residue ID 32 in this example) with the following commands (the selection syntax is the same as PrepQM tool. More details on the interactive selection environment can be found in the MiMiCPy documentation):

```
> resname is ACT
> resid is 32
> q
```

This selection will generate the following message, and MiMiCPy will be terminated.

<table>
<thead>
<tr>
<th>Atom</th>
<th>Residue</th>
<th>CPMD ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 C1</td>
<td>1 ACT</td>
<td>1</td>
</tr>
<tr>
<td>2 C2</td>
<td>1 ACT</td>
<td>2</td>
</tr>
<tr>
<td>3 C3</td>
<td>1 ACT</td>
<td>3</td>
</tr>
<tr>
<td>4 O1</td>
<td>1 ACT</td>
<td>10</td>
</tr>
<tr>
<td>5 H1</td>
<td>1 ACT</td>
<td>4</td>
</tr>
<tr>
<td>6 H2</td>
<td>1 ACT</td>
<td>5</td>
</tr>
<tr>
<td>7 H3</td>
<td>1 ACT</td>
<td>6</td>
</tr>
</tbody>
</table>

(continues on next page)
As it is possible to see from the table in the output, the CPMD IDs for the acetone molecule are the first 10 CPMD IDs (the QM atoms are always placed at the beginning from CPMD) and the IDs for the water molecule change from 101, 102 and 103 in the MM topology to non-subsequent indices: 41, 577 and 578, respectively.

### 9.4.1 Output formats

In the previous example the CPMD IDs were only shown on the terminal and not saved. Moreover, the table format may not be always straightforward to convert into a CPMD input files, especially when you are dealing with a large number of atoms.

The CPMD IDs can be saved into an output file by adding the optional keyword -out, followed by the output file name.

Different output formats (for the result shown on the terminal or saved in the output file) can be specified by the additional keyword -print, each of them useful in different cases:

- **table**: a table with Atom, Residue and CPMD ID columns, reporting respectively MM atom name and number, MM residue name and number, and CPMD ID. This is the default option and is the same format as the one printed to the terminal before.

- **list**: a list of the CPMD IDs preceded by a line reporting ‘No. of atoms:’ and their number. This format is for example useful when constraints want to be specified using the **FIX ATOMS** keyword, which needs to be followed by the number of atoms to be fixed and a list of their IDs.

- **range**: CPMD IDs grouped in ranges of subsequent atoms, with one group of consecutive atoms per line. The IDs of atoms with no other closeby IDs simply correspond to a line with a single number (the atom ID). This format is for example useful when multiple thermostats are used: the keyword **NOSE IONS LOCAL** collects groups of atoms to separate thermostats and ranges of IDs need to be specified for each of them.

To test these options, try to run the following commands (where the -out keyword is used in combination with different -print options), always selecting **resname is ACT** and **resid is 32** as done previously.

```bash
mimicpy cpmdid -top ../acetone_equilibrated.top -inp ../prepqm/cpmd.inp -print table -out cpmdids_table.txt
```

In this case you should see in the output file the same table printed on the terminal in the previous example.

```bash
mimicpy cpmdid -top ../acetone_equilibrated.top -inp ../prepqm/cpmd.inp -print list -out cpmdids_list.txt
```

In this case, the output should be the following:
No. of atoms: 13

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>10</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>577</td>
<td>578</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
mimicpy cpmdid -top ../acetone_equilibrated.top -inp ../prepqm/cpmd.inp -print range -out cpmdids_range.txt
```

In this case, the output should be the following:

```
1 to 10
41
577 to 578
```

Compare the three files generated to verify they provide the same information. Based on a different context, one format or the other may be more useful.

To conclude, one additional useful keyword for MiMiCPy CPMDid, with the same usage as the PrepQM tool, is `--select`: it allows to store in a file the selection command in text format and provides it as input, to avoid manually typing it in the interactive session.

### 9.5 Debugging with CPMD2Coords and Geom2Coords

There are two additional commands that are especially useful while debugging a CPMD input file for MiMiC: CPMD2Coords and Geom2Coords. CPMD2Coords allows to convert the atom coordinates in a MiMiC CPMD input file to a GRO or a PDB file. This can assist in debugging the QM region in a MiMiC run, and confirm the correct atoms were placed in the QM region. Geom2Coords allows to convert a CPMD `GEOMETRY` or `GEOMETRY.xyz` file to a GRO or PDB file for easy visualization.

Create a folder in the `MiMiCPyTutorial/solution` directory where you will test the CPMD2Coords and Geom2Coords commands:

```
mkdir debug
cd debug
```

#### 9.5.1 CPMD2Coords

After selecting the QM region using PrepQM, it is often good practice to check by visual inspection that all the correct QM atoms have been included. Any mistakes can lead to large issues when running MiMiC. But, it is difficult to extract the coordinates from the given CPMD input and visualize them. Here we can use the CPMD2Coords tool.

A CPMD input file is needed to test. If you skipped the first section of this tutorial, where the PrepQM tool is illustrated, or if you want to start from scratch, you can quickly generate a CPMD input with the following command:

```
cp ../../data/sele.dat .
mimicpy prepqm -top ../acetone_equilibrated.top -coords ../acetone_equilibrated.gro -sele sele.dat
```

Now we can test the use of CPMD2Coords, by running:
This will result in the atoms and coordinates in the QM region of `cpmd.inp` being written to `qm.gro` in GROMACS format.

<table>
<thead>
<tr>
<th>Coordinates from <code>cpmd.inp</code>, Generated by MiMiCPy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>1ACT</td>
</tr>
<tr>
<td>0.28300</td>
</tr>
</tbody>
</table>

This can be easily visualized using for example VMD, or any other visualization software, and helps to confirm that the QM region selection is correct.

### 9.5.2 Geom2Coords

Another such tool is Geom2Coords. MiMiC runs output the trajectory and a GEOMETRY file from CPMD and as a TRR file from GROMACS. Usually these files will contain the same information, so it is prudent to only deal with the easier to handle TRR file. However, in times of unexpected errors, the information may not match or the TRR file may not be written correctly. In these cases it is imperative to be able to visualize the GEOMETRY file. We can do this with the Geom2Coords tool.

If you have already successfully run a MiMiC QM/MM run of the acetone system, you will have a GEOMETRY file which you can move to the `debug` directory we created. If you want to start from scratch, we provide the required file to test in the `data` folder:

```
cp ../../data/GEOMETRY_example ./GEOMETRY
```

Then run:

```
mimicpy geom2coords -geom GEOMETRY -top ../acetone_equilibrated.top -inp cpmd.inp -out...
  _GEOMETRY.gro -guess True
```

This will result in `GEOMETRY.gro`, which can be visualized easily for example using VMD, or any other visualization software. Information such as atom number, residue name, and residue number are associated to each atom (as in a standard GROMACS coordinate file), and this helps the visualization, since the initial GEOMETRY file only had coordinates and velocities.
9.6 As a Python Library

The features of MiMiCPy can be accessed directly from Python code. This approach is particularly powerful when developing automated workflows. This final section of the handbook is intended as an introduction to the use of MiMiCPy as a Python library, and especially during the QM region selection not all the commands will end up in a reasonable choice of the atoms. However, they should help to illustrate some basic commands to work with MiMiCPy as a Python library and show its capabilities.

In order to follow this section, some familiarity with MiMiCPy is expected and in particular all the details of the selection syntax or CPMD input are not explained in detail again. For this reason, we suggest to at least go through the PrepQM section of this handbook before proceeding.

Create a folder in the MiMiCPTutorial/solution directory where you will test the usage of MiMiCPy as a python library:

```bash
mkdir python-library
cd python-library
```

First import the library:

```python
import mimicpy
```

The Preparation class is central to the code architecture. It can be initialized as follows:

```python
prep = mimicpy.Preparation(selector)
```

The selector variable is an instance of a Selector class. This collates the input topology and coordinates, and handles the selection keywords from the user. Currently, there are three types of selector classes implemented:

```python
selector = mimicpy.DefaultSelector(mpt, coords)
sselector = mimicpy.VMDSelector(mpt)
sselector = mimicpy.PyMOLSelector(mpt)
```

We use DefaultSelector to take advantage of the selection language provided by MiMiCPy. Notice that the selector accepts two arguments, an Mpt instance and a CoordsIO instance (the CoordsIO instance can be optionally passed to VMDSelector and PyMOLSelector. If not, it is assumed to be already loaded in the VMD/PyMOL environment). The Mpt or MiMiCPy Topology class is the class that handles parsing of all MM topologies supported by MiMiC. For example, to read a GROMACS topology file of acetone:

```python
mpt = mimicpy.Mpt.from_file('..acetone_equilibrated.top')
```

Make sure GROMACS has been sourced, i.e. is available on the command line.

The Mpt class exposes the internal information of the topology. For example, to access the residue ID of the atoms with index 5:

```python
mpt[5]['resid']
```

To access the residue IDs of all atoms as a list, we write:

```python
mpt.resid
```

A similar syntax exists for properties like name, type, resname and mol.

Similarly, the CoordsIO class handles parsing of all MM coordinate files supported by MiMiC. This includes .gro and .pdb files.
```python
coordsio = mimicpy.CoordsIO('./acetone_equilibrated.gro')
```

The coordinates and box size can be accessed by:

```python
coordsio.coords
coordsio.box
```

When setting the `selector` variable, the topology information can be passed giving the `Mpt` class as input or directly passing the path to the topology file. The `CoordsIO` class is loaded internally and the path to the coordinates file needs to be passed.

In summation, to create a `Preparation` instance, you can use one of the two following commands:

```python
mpt = mimicpy.Mpt.from_file('./acetone_equilibrated.top')
prep = mimicpy.Preparation(mimicpy.DefaultSelector(mpt, './acetone_equilibrated.gro'))
```

or, equivalently:

```python
prep = mimicpy.Preparation(mimicpy.DefaultSelector('./acetone_equilibrated.top', './acetone_equilibrated.gro'))
```

We can now add QM atoms, and prepare our input files using the `prep` object.

```python
prep.add('name is O1 or name is C1 or name is C2')
prep.add('name is H1 or name is H2 or name is H3')
```

We can also delete atoms, to get the same result:

```python
prep.add('resname is ACT')
prep.delete('name is C1 or name is C3')
prep.delete('name is H4 or name is H5 or name is H6')
```

We can also add boundary atoms, either manually:

```python
prep.add('name is C1', is_bound=True)
```

or by automatic detection:

```python
prep.find_bound_atoms()
```

Both methods should result in the same CPMD input file. To generate the CPMD input and the GROMACS index file:

```python
ndx, cpmd = prep.get_mimic_input()
```

A number of optional arguments can be passed to `get_mimic_input()`, allowing for many of the same functionalities of `PrepQM`:

```python
get_mimic_input(inp_tmp=None, ndx_out=None, inp_out=None, box_padding=0.0, cell_as_absolute=False, ndx_group_name='QMatoms', path=None, charge=None, pp_info=None)
```

Where

- `inp_tmp`: template cpmd input file
- `ndx_out`: name of gromacs index file
- `inp_out`: name of the mimic cpmd input file
MiMiC

- **box_padding**: extra distance between qm atoms and wall in nm
- **cell_as_absolute**: return qm cell info as absolute instead of relative
- **ndx_group_name**: name of qmatsos group in the index file
- **path**: path in the mimic section, overrides template
- **charge**: charge of the qm region, overrides default algorithm
- **pp_info**: pseudopotential information as a pandas DataFrame or whitespace separated file

This function returns an instance of the Ndx and CPMD class. These handle the parsing and data access to resulting CPMD and index files. We can view the contents by printing them to console:

```python
print(cpmd)
print(ndx)
```

These instances make it easier to edit the input scripts. Suppose we want to add a new &DFT section to cpmd:

```python
cpmd.DFT = mimicpy.scripts.cpmd.Section()
```

This will create a new empty section called &DFT. We can now add parameters to this:

```python
cpmd.DFT.FUNCTIONAL___BLYP = True
```

This will write the parameter `FUNCTIONAL BLYP` (** designates a space) to the DFT section. If you want to remove this parameter:

```python
cpmd.DFT.FUNCTIONAL___BLYP = False
```

Parameters with values can also be set. For example, to set the trajectory sample rate in the &CPMD section:

```python
cpmd.CPMD.TRAJECTORY___SAMPLE = 500
```

If ndx_out and inp_out are passed to `get_mimic_input()`, the CPMD input script and the index file are written to files. We can also manually write them by running:

```python
with open('cpmd.inp', 'w') as f:
    f.write(str(cpmd))
```
FREQUENTLY ASKED QUESTIONS (FAQ)

For now, there are no questions.
CHAPTER
ELEVEN

HOW TO GET HELP

MiMiC is distributed for free and without any guarantee of reliability, accuracy, or suitability for any particular purpose. No obligation to provide technical support is expressed or implied.

We have a discussion group for our user community. We encourage you to be comprehensive in providing explanations and to share all relevant background information that could facilitate better assistance. Please be aware that although we strive to assist you to the best of our ability, we are not equipped to offer support for external programs, such as CPMD and GROMACS. For bug reports, please visit our GitLab page.

11.1 Where should you post?

Before you decide to seek help, please make sure that your issue is not a common one by consulting the FAQ.

- **How do I ...?** Ask in the discussion group or file an issue on GitLab
- **I get this error, why?** Ask in the discussion group or file an issue on GitLab
- **I get this error and I am sure it is a bug.** Please file an issue on GitLab
- **I have an idea/request.** Ask in the discussion group or file an issue on GitLab
- **I have an idea/request and a plan for its implementation.** Please file an issue on GitLab
- **Why do you ...?** Ask in the discussion group or file an issue on GitLab
- **When will you ...?** Ask in the discussion group or file an issue on GitLab
- **Can I open a discussion on this part of the code?** Please file an issue on GitLab
MiMiCPy is a freely distributed component of the MiMiC package, consisting of tools to easily prepare and debug MiMiC input files. The instructions for installation are described in the MiMiCPy.

MiMiCPy primarily consists of a set of command line tools to easily prepare and debug complex CPMD and GROMACS input files for a MiMiC run. The list of tools are:

- **PrepQM**
  Prepare the QM and MM input files from MM data

- **CPMD2Coords**
  Convert the atom coordinates in a CPMD/MiMiC input file to a GRO or PDB file

- **FixTop**
  Fix the [ atomtypes ] section of the GROMACS MM topology for a MiMiC run

- **CPMDid**
  Find the CPMD indices of QM and MM atoms in the system

- **Geom2Coords**
  Convert the atom coordinates in a CPMD GEOMETRY file to a GRO or PDB file

Each tool is described in following sections. In addition to the command line tools, plugins of the PrepQM tool for VMD and PyMOL have also been provided. MiMiCPy can also be used as a Python library for more advanced functionalities.

For more details about MiMiCPy, we refer to the following article:

As the name suggests, **mimicpy prepqm** is the main tool to prepare MiMiC QM/MM input files. In short, it accepts MM topology and coordinate files, and after inputting the QM selection, outputs the CPMD input file and GROMACS index file necessary to create the TPR run file (or it can output the TPR run file directly).

An example of running a simple **mimicpy prepqm** command in the console:

```
$ mimicpy prepqm -top acetone.top -coords acetone.gro

***** MiMiCPy *****

For more information type mimicpy [subcommand] --help

====> Running prepqm <=====

**Reading topology**

Cannot find path to Gromacs installation.
Read atoms from acetone.itp.
No atoms found in acetone.top.

Some atom types had no atom numbers information.
They were guessed as follows:

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Atom Type</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>c</td>
</tr>
<tr>
<td>c3</td>
</tr>
<tr>
<td>o</td>
</tr>
<tr>
<td>hc</td>
</tr>
</tbody>
</table>

**Reading coordinates** |Done

Please enter selection below. For more information type 'help'
> add resname is ACT

(continues on next page)
> q
Using default values for maxstep and timestep
Wrote Gromacs index file to index.ndx
Wrote new CPMD input script to cpmd.inp

=====> Done <=====  

13.1 Options

mimicy prepqm has the following options:

13.1.1 Input Options

- **-top** [.top]
The file name of the MM topology file. Currently, only GROMACS topology files are accepted.

- **-coords** [.gro/.pdb]
The file name of the initial coordinates for the QM/MM run. Please make sure that the order of atoms matches exactly with that in the topology file.

- **-mdp** [.mdp]
An optional GROMACS MD parameters file can be passed. If this is passed, mimicy prepqm will check the file for errors and consistency with the MiMiC run. It will then automatically generate the GROMACS run TPR file using the gmx grompp tool.

13.1.2 Output Options

- **-out** [.inp] (cpmd.inp)
The file name of the output CPMD script. By default, it is cpmd.inp.

- **-ndx** [.ndx] (index.ndx)
The file name of the output GROMACS index file containing the GROMACS IDs of the QM atoms. This is necessary to generate a run TPR file. By default, the file name is index.ndx.

- **-tpr** [.tpr] (mimic.tpr)
If the GROMACS MD parameters file is passed, the TPR file will be generated with this file name. By default, it is generated as mimic.tpr.

13.1.3 Other Options

- **-guess** (True)
CPMD requires the atomic species to be specified for each atom in the input script. This is sometimes not contained in the GROMACS topology, especially for non-standard atoms not present in the force field data. In these cases, MiMiCPy can guess the elements of each atom from other information present in the topology. This flag toggles this guessing of atomic elements. The elements are guessed using the following algorithm:

1. If the atomic mass of the atom is 1 a.m.u, it is assigned as a hydrogen.
2. If the atomic mass is between 1 and 36 a.m.u, the atomic number is approximated as half of the atomic mass.
3. If the atomic mass is greater than or equal to 36, the atomic name is checked to see if it corresponds with a valid atomic symbol.

4. If this is not true, the same is done with the atom type.

5. If this is also not true, the first letter of the atomic name is checked to see if it corresponds with a valid atomic symbol.

6. If this is also not true, the same is done for the atomic type.

8. Finally if this is also not true, the atom is assigned as hydrogen and a warning is printed.

**mimicpy prepqm** will print out the elements guessed in an easy to read format. This is an example for a non-standard acetone molecule:

<table>
<thead>
<tr>
<th>Atom Type</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>c3</td>
<td>C</td>
</tr>
<tr>
<td>o</td>
<td>O</td>
</tr>
<tr>
<td>hc</td>
<td>H</td>
</tr>
</tbody>
</table>

Always cross-check that the elements guessed make physical sense. If this guessing is turned off, the program will exit with an error if it encounters an atom with no element information in the topology.

**-sele [.txt/.dat]**

The file name of the list of selection commands in text format. The selection commands to select the QM region atoms can be entered in the interactive session, or optionally passed as a file. The format of the selection keywords is given in the following section.

**-pp [.dat]**

The file name of the text file containing the pseudopotential information for each atom element. The pseudopotential filename, the non-locality (LMAX, LOC) and other labels given as the header of each pseudopotential in the &ATOMS section of the CPMD input script can be specified in this file. MiMiCPy will use this information to automatically fill up the pseudopotential information. The data for each element is given in separate lines, and each line has the following format:

<element> <pp filename> <link pp filename> <labels> <lmax> <loc>

A - can be specified to skip an of this information. Below and example is given, detailing the pseudopotential information for zinc and carbon.

Zn Zn_MT_BLYP_SEMI.psp - GAUSS-HERMITE=20 D S # Zinc atoms
C C_MT_BLYP.psp C_GIA_DUM_AN_BLYP.oecp KLEINMAN-BYLANDER P - # Carbon atoms

Characters specified after # are treated as comments and not read.

**-bound (False)**

Toggle automatic guessing of boundary atoms. By default, this is False. It is important to specify boundary atoms at the QM-MM interface. Atoms can be marked as boundary atoms manually in the selection interface (see next section). Or MiMiCPy can automatically mark the QM atoms at the
QM/MM interface as boundary atoms. If this is performed, prepqm will output the atoms marked as link in the format below:

```
Link atoms were automatically set. The below atoms were marked as link:

+------------------+
| Atom | Residue |
+------------------+
| 82 CA | 5 HE2 |
+------------------+
| 286 CA | 18 HD5 |
+------------------+
| 255 CA | 16 HD4 |
+------------------+
```

-nsa [.txt/.dat]

File name of file containing list of non-standard atom types in a two columns format. CPMD requires the element information for all QM atoms. If this is not present for some atom types in the topology, they can be specified in this file:

```
<atom type> <element>
```

For example:

```
c c
c3 c
o1 O
```

-ff

MiMiCPy requires access to the force field data directory to read the GROMACS topology. This is usually present in the GMXDATA or GMXLIB environment labels (in a standard GROMACS installation). If this is not found, MiMiCPy checks the current directory for the force field folder. To override this behavior, pass the directory containing the force field folder with this flag.

-inp [.inp]

File name of the template CPMD input script. By default, mimicpy prepqm outputs a barebones CPMD script, with only the &MIMIC and &ATOMS sections filled. To have a complete input script, place the other CPMD sections and commands in this template script.

-gmx (gmx)

If the GROMACS MD parameters file is passed, mimicpy prepqm will attempt to run the GROMACS preprocessor to generate a TPR file. Specify the name of the executable for running GROMACS (for example, gmx, gmx_mpi or gmx_mpi_d) with this flag.

-pad (0)

In MiMiC, the QM region is treated as an isolated system. This often requires the use of Poisson solver like the Tuckerman solver, where a padding should be given between the outermost QM atoms and the QM box wall. This value can be specified as a float in nanometers with this flag. The value is added in each direction of the box size, in both direction (i.e., multiplied by 2).

-abs (False)

In a MiMiC run, the CELL size in the CPMD script must be specified in the relative format. But you can force the the cell size to be written in the absolute format with this flag.

-qma (QMatoms)

The name of the QM atoms group in the GROMACS index file. This name must be passed correctly in the .mdp file. If the GROMACS MD parameters file is passed, this is checked automatically.
-path
The path to the GROMACS TPR file must be specified in the &MIMIC section of the CPMD script. By default, mimicpy prepqm uses the current directory. This can be overridden by passing a custom directory with this flag.

-q
Specify the total charge of the QM region. By default, MiMiCPy calculates this by using the MM charges of the QM atoms from the force field data. Passing this flag overrides this behavior.

-buffer (1000)
Buffer size for reading input topology. Should only be used if unexpected errors occur in reading of topology.

-buffer coordinates (1000)
Buffer size for reading input coordinates. Should only be used if unexpected errors occur in reading of coordinates.

13.2 The Interactive Selection Environment

After mimicpy prepqm successfully reads the given input options, the user will be taken into the interactive selection environment. Here, the atoms to be added to the QM region can be specified using the MiMiCPy selection language. Details on the selection language can be found in the following section. The following are valid commands:

**add <selection>**
The `add` command, followed by a selection command, can be used to add atoms to the QM region. Here is an example: `add resname is ACT` Any errors with the selection command are displayed below.

**add-link <selection>**
The `add-link` command, followed by a selection command, can be used to link atoms to the QM region. Here is an example: `delete resname is ACT` Any errors with the selection command are displayed below. Link atoms will be separated into their own pseudopotential block in the &ATOMS section of the CPMD input file.

**delete <selection>**
The `delete` command, followed by a selection command, can be used to delete/remove atoms from the QM region. Here is an example: `delete resname is ACT` Any errors with the selection command are displayed below.

**clear**
The `clear` command completely clears the QM region, and deletes all selected QM atoms.

**view <filename>**
The `view` command, optionally followed by a text filename, can be used to display the details of the QM atoms selected. If a filename is specified, the details are written to file. If not, they are written to the console window.

**q or quit**
The `q` or the `quit` command causes mimicpy prepqm to exit from the interactive environment. All QM selected will be used to generate the QM and MM input scripts.
13.2.1 The Selection Language

The syntax for the selection query involves the following general structure:

```
keyword logical_operator value
```

where `keyword` can include the following:

- `resname` The residue name of the selection according to the MM topology
- `resid` The residue ID of the selection according to the MM topology
- `name` The atom name of the selection according to the MM topology
- `type` The atom type of the selection according to the MM topology
- `id` The atom ID of the selection according to the order they appear in the MM coordinates
- `mol` The molecule of the selection according to the MM topology

and `logical_operator` can be `is`, `not`, `>`, `<`, `>=`, or `<=`.

Different selection queries can be joined by using and or or operators, and grouped with brackets, such as:

```
add (resname is RES) and (name is x)
```

which will add the atom named X of residue RES.

13.3 VMD and PyMOL Plugins

The console version of the PrepQM tool allows for selecting the QM region with only the MiMiCPy selection language. This is sufficient for simple selection commands. But, if more complex QM regions are required, PrepQM plugins for the popular molecular visualization software VMD and PyMOL are provided. Many of the same options available in the console version, are available in the plugins as well. Refer to the previous section for a detailed explanation on these options. Before running PrepQM in the command line window of the visualization software, the coordinate file containing the input atomic coordinates for the MiMiC run must be loaded.

13.3.1 PyMOL Plugin

The usage and options for the PrepQM PyMOL plugin is given below:

```
prefqm top, selection [, link_selection [, ndx [, out [[, inp [, pad [, abs [, qma [,...
          path [, q [, pp [, link]]]]]]]]]]]]]]]
```

- `top` The file name of the MM topology file. Currently, only GROMACS topology files are accepted.
- `selection` The PyMOL selection expression or name pattern of the QM region.
**link**
Optional PyMOL selection expression or name pattern of the link atoms in the QM region.

**ndx**
The file name of the output GROMACS index file containing the GROMACS IDs of the QM atoms.
{default: index.ndx}

**out**
The file name of the output CPMD script. {default: cpmd.inp}

**inp**
File name of the template CPMD input script. Performs a similar function to the console version.

**pad**
The padding to be added to the QM region in nanometers. The value is added in each direction of the box size, in both direction (i.e., multiplied by 2). {default: 0}

**abs**
Force the cell size to be written in the absolute format with this flag. {default: False}

**qma**
The name of the QM atoms group in the GROMACS index file. {default: qmatoms}

**path**
The path to the GROMACS TPR file must be specified in the &MIMIC section of the CPMD script.

**q**
Specify the total charge of the QM region.

**pp**
The file name of the text file containing the pseudopotential information for each atom element.
Performs a similar function to the console version.

**link**
Toggle automatically guessing of link atoms. {default: True}

An example of running a simple prepqm command in the PyMOL command line:

```
prepqm acetone.top, sele0, pad=3.5, link=True
```

### 13.3.2 VMD Plugin

The usage and options for the PrepQM VMD plugin is given below:

```
prepqm [top topol.top] [sele atomselect0] [sele_link None] [molid 0] [indx index.ndx] ...
[out cpmd.inp] [inp None] [pad 0] [abs False] [qma qmatoms] [path None] [q 0] [pp ...
None] [find_link False]
```

**top**
The file name of the MM topology file. Currently, only GROMACS topology files are accepted.
{default: topol.top}

**sele**
The PyMOL selection expression or name pattern of the QM region. {default: atomselect0}

**sele_link**
Optional PyMOL selection expression or name pattern of the link atoms in the QM region.

**molid**
The molecular ID of the coordinate file loaded into VMD.
ndx
The file name of the output GROMACS index file containing the GROMACS IDs of the QM atoms.
{default: index.ndx}

out
The file name of the output CPMD script. {default: cpmd.inp}

inp
File name of the template CPMD input script. Performs a similar function to the console version.

pad
The padding to be added to the QM region in nanometers. The value is added in each direction of
the box size, in both direction (i.e., multiplied by 2). {default: 0}

abs
Force the the cell size to be written in the absolute format with this flag. {default: False}

qma
The name of the QM atoms group in the GROMACS index file. {default: qmatoms}

path
The path to the GROMACS TPR file must be specified in the &MIMIC section of the CPMD script.

q
Specify the total charge of the QM region.

pp
The file name of the text file containing the pseudopotential information for each atom element.
Performs a similar function to the console version.

link
Toggle automatically guessing of link atoms. {default: True}

An example of running a simple prepqm command in the VMD command line:

```
prepqm -top acetone.top -sele sele0 -pad 3.5 -link True
```
14.1 CPMD2Coords

`mimicpy cpmd2coords` is a tool to convert the atom coordinates in a MiMiC CPMD input file to a GRO or a PDB file. The coordinates in the CPMD input script can be hard to read and debug. This tool uses the information for the `&MIMIC` and `&ATOMS` sections, and combines it with the MM topology to create a coordinate file that can be easily read and visualized by a molecular visualization software. This can assist in debugging the QM region in a MiMiC run, and confirm the correct atoms were placed in the QM region.

14.1.1 Options

`mimicpy cpmd2coords` has the following options:

**Input Options**

- `-top [.top]`
  The file name of the MM topology file. Currently, only GROMACS topology files are accepted.

- `-inp [.inp]`
  The file name of the CPMD input script with the atoms coordinates formatted for a MiMiC run.

**Output Options**

- `-coords [.gro/.pdb] (mimic.gro)`
  The file name of the output coordinate file. Currently, GRO and PDB formats are supported. By default, it is `mimic.gro`.

**Other Options**

- `-guess (True)`
  Toggling guessing of atomic species in the MM topology. For more details refer to the `PrepQM section`.

- `-nsa [.txt/.dat]`
  File name of file containing list of non-standard atom types in a two columns format. For more details refer to the `PrepQM section`.

- `-ff`
  Pass the directory containing the force field folder with this flag. For more details refer to the `PrepQM section`. 
-buf (1000)
  Buffer size for reading input topology. Should only be used if unexpected errors occur in reading of topology.

14.2 FixTop

mimicpy fixtop is a tool to write a consolidated [ atomtypes ] section in the GROMACS topology, containing the atomic elements of all (QM and MM) atoms in the system. It is important for CPMD to have access to the atomic element information all atoms. If some of them are missing, it can lead to garbage values and segmentation faults in a MiMiC run. This is especially true if an MM atom, without atomic elements information, participates in a QM-MM bond.

Often times, non-standard atoms in a GROMACS topology may not have atom species information specified. FixTop adds an [ atomtypes ] section in a GROMACS ITP file. These elements are either guessed by MiMiCPy, or can be explicitly specified with the -nsa flag.

14.2.1 Options

mimicpy fixtop has the following options:

Input Options

-top [.top]
  The file name of the MM topology file. Currently, only GROMACS topology files are accepted.

Output Options

-fix [.itp] (atomtypes.itp)
  The file name of the GROMACS ITP file where the [ atomtypes ] section, containing the atomic elements of all atoms, will be written to.

Other Options

-guess (True)
  Toggling guessing of atomic elements in the MM topology. For more details refer to the PrepQM section.

-nsa [.txt/.dat]
  File name of file containing list of non-standard atom types in a two columns format. For more details refer to the PrepQM section.

-ff
  Pass the directory containing the force field folder with this flag. For more details refer to the PrepQM section.

-cls [.itp .itp .itp ..]
  Pass a space separated list of GROMACS ITP files, where the [ atomtypes ] section will be cleared from all of them. GROMACS allows only one [ atomtypes ] section in a topology. The main [ atomtypes ] section will be written to the file specified in the -fix flag. All other [ atomtypes ] sections need to be cleared.
-buf (1000)
  Buffer size for reading input topology. Should only be used if unexpected errors occur in reading of topology.

14.3 CPMDid

mimicpy cpmdid is a tool to print out the CPMD IDs of both QM and MM atoms. These IDs are needed to select atoms in the CPMD script when performing certain commands like applying constraints, adding multiple thermostats, etc. The MM IDs of atoms from GROMACS are reshuffled by CPMD in a non-obvious way. This tool keeps track of the conversion of MM IDs to QM IDs. The atoms can be selected by the MiMiCPy selection language (described here).

14.3.1 Options

mimicpy cpmdid has the following options:

Input Options

- **-top [top]**
  The file name of the MM topology file. Currently, only GROMACS topology files are accepted.

- **-inp [inp]**
  The file name of the CPMD input script with all information for a MiMiC run.

Output Options

- **-out [txt]**
  An optional text file to write the results to.

Other Options

- **-sele [txt/dat]**
  The file name of the list of selection commands in text format. The selection commands to select the QM region atoms can be entered in the interactive session, or optionally passed as a file. The format of the selection keywords is given in the following section.

- **-ff**
  Pass the directory containing the force field folder with this flag. For more details refer to the PrepQM section.

- **-buf (1000)**
  Buffer size for reading input topology. Should only be used if unexpected errors occur in reading of topology.
14.4 Geom2Coords

mimicpy geom2coords is a tool to convert the a CPMD GEOMETRY or GEOMETRY.xyz file to a GRO or PDB file for easy visualization. Although, the QM and MM trajectories contain the same information (especially when the MiMiC simulation outputs physically correct results), there are often times when they do not match. In these events, visualizing the CPMD GEOMETRY output can be helpful in diagnosing errors with the simulation. This tool provides an easy to use interface for this purpose.

14.4.1 Options

mimicpy geom2coords has the following options:

**Input Options**

- **-geom [GEOMETRY/GEOMETRY.xyz]**
  The file name of the CPMD GEOMETRY file. Can be a plain GEOMETRY file, or an CPMD XYZ file.

- **-top [.top]**
  The file name of the MM topology file. Currently, only GROMACS topology files are accepted.

- **-inp [.inp]**
  The file name of the CPMD input script with all information for a MiMiC run.

**Output Options**

- **-coords [.gro/.pdb] (GEOMERY.gro)**
  The file name of the output coordinates. Currently, only GRO and PDB files are supported.

**Other Options**

- **-ff**
  Pass the directory containing the force field folder with this flag. For more details refer to the PrepQM section.

- **-buf (1000)**
  Buffer size for reading input topology. Should only be used if unexpected errors occur in reading of topology.

