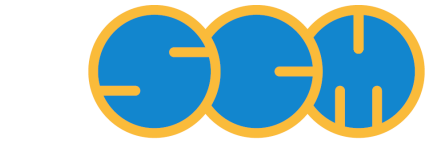

FlexMD Manual

ADF Program System
Release 2013

Scientific Computing & Modelling NV
Vrije Universiteit, Theoretical Chemistry
De Boelelaan 1083; 1081 HV Amsterdam; The Netherlands
WWW: www.scm.com
E-mail: support@scm.com

FlexMD: A python library for flexible multi-scale molecular dynamics simulation

Table of Contents

Basic philosophy and intended usage

We present a flexible python library for molecular dynamics, specialized in multi-scale simulations in a broad sense. At its core, the library interfaces the Atomistic Simulation Environment (ASE) [1] molecular dynamics modules with a wide range of molecular mechanics and electronic structure codes. As such, it allows simple dynamics using forces computed with any energy/gradient evaluator provided by the ADF package.

Additionally, FlexMD allows the partitioning of a system into regions described at different resolution, with the aim of running multi-scale (hybrid) force calculations. Besides the traditional, rigid, multi-scale partitioning, FlexMD includes different schemes for Adaptive Multi-scale Molecular Dynamics. Such simulations allow the resolution of a particle to change according to its distance from a predefined active site, which is a necessity for successful multi-scale description of diffusive systems such as chemical reactions in solution.

Finally, the library couples the dynamics to rare events techniques, either implemented in FlexMD itself, or accessible through an interface with the PLUMED library for free energy calculations [7], opening the possibility for evaluation on time-scales beyond the reach of standard molecular dynamics simulations.

The FlexMD package is designed to make simulation options possible that are not available natively in the ADF package. Its flexible nature makes it very versatile, but comes at a cost. This cost might be completely neglectable in most simulations, but it can be very high in some cases (usually when combining only cheap methods such as forcefields).

The intended users for the FlexMD package are those with some Unix/Linux experience and a basic understanding of the Python Programming Language. The user is also supposed to have a basic understanding of the various methods he wishes to combine. For example, if metadynamics is supposed to be combined with ADF, FlexMD expects the user to have knowledge about DFT calculations and the usage of Collective Variables. Finally, as with every computational method, the user should monitor the FlexMD performance, both in accuracy and speed.

FlexMD functionality summary

Molecule

Input/output

- Reads and writes PDB and XYZ files
- Reads and writes topology data (in CHARMM format)
- Reads and writes force field data (on CHARMM format)

Analysis

- Extracts geometry data

Drawing functionality

- Adds atoms and bonds
- Changes bond-lengths, angles and torsions
- Cuts fragments
- Cuts solvent boxes and droplets
- Performs rotations and translations, to fit bonds to axes and planes

Periodic functionality

- Adds periodic images
- Wraps molecules into periodic box

Water specific

- Finds hydrogen bonds
- Finds shortest water bridge connecting H-donor and acceptor

Energy and force calculations

Standard

- ADF
- DFTB
- REAXFF
- UFF
- MOPAC
- NAMD
- Lennard-Jones force fields

Multi-scale

- QM/MM, mechanical embedding: Combines all the codes above
- Hybrid : More flexible than QM/MM. Combines different force calculations by summing or subtracting the energies and forces. The standard calculations (above) can therefore be combined with:
    - Metadynamics
    - Plumed (external code that computes free energy data)
    - Constraints
- Adaptive QM/MM (for chemistry in solution)
    - Difference-based Adaptive Solvation (DAS)
    - Sorted Adaptive Partitioning (SAP)
    - Buffered-Core (BC)
    - Flexible Inner Region Ensemble Separator (FIRES)

Molecular Dynamics

- Uses ASE as the molecular dynamics driver for all above methods
- Analyses trajectories

1. Introduction

FlexMD is a python package providing molecular dynamics (MD) simulations using the energy evaluation methods made available by the ADF suite. A set of example scripts can be found in the examples/scmlib directory of a standard ADF installation.

FlexMD can be accessed interactively by running `startpython`, followed by a standard python import command for the package `scm.flexmd`. The python help function can be used to obtain detailed documentation about all FlexMD classes. In the following example, an inquiry of one class (the MDMolecule class) can be performed.

```
$ startpython
>>> from scm import flexmd
>>> help(flexmd.MDMolecule)
```

To leave the interactive help, press q. The help function can also be used to list the contents of the FlexMD package:

```
$ startpython
>>> from scm import flexmd
>>> help(flexmd)
```

Python can also give the help documentation as plain text:

```
$ startpython
>>> from scm import flexmd
>>> import pydoc
>>> print pydoc.render_doc(flexmd.ForceJob, "Help on %s")
```

2. Molecular Dynamics

FlexMD defines the molecular system under study through the **MDMolecule** class: an instantiation of this class holds all information about the molecular system to be simulated, such as coordinates, topology, and force field parameters (if needed). An **MDMolecule** object can be initialized from a PDB or XYZ file, by specifying its path at object creation.

An interface to energy evaluators is provided by specialized **ForceJob** classes, acting as wrappers around the ADF suite of programs. A **ForceJob** requires an **MDMolecule** object to be specified at creation. The resulting **ForceJob** object can either be used directly by the Atomistic Simulation Environment (ASE) [1] library as a calculator object (see examples/scmlib/ASE_emt_h2o) or with the **ASEMDPropagator** class, which provides methods for running an MD time step using ASE classes. Internally, the propagator sets up the required ASE objects, passes the **ForceJob** object to them, and retrieves the new positions and velocities. An additional protagonist, an **MDManager** class instance, coordinates the MD simulation by running the MD steps with the **ASEMDPropagator** object and writing trajectory information to file.

During creation of an **MDManager** object, a directory ‘QMMD’ is created, which contains a file TRAJEC00.DCD holding the geometries along the trajectory, a file FTRAJEC00.DCD holding the forces along the trajectory, and finally a file ENERGY00.dat holding the potential and kinetic energy, as well as the temperature throughout the evaluation. To extract the geometries from the trajectory file, the **DCDFile** class is available, providing methods to read and write geometries to and from a trajectory file in DCD format. The **MDManager** is also responsible for handling restart of a previous MD evaluation: if a 'QMMD' directory is already present at script invocation, the new output files will be assigned the number subsequent to the highest numbered files in that directory. In addition, provided the previous run terminated normally, the restart will continue from the final geometry and velocity of the previous run.

The ADF package contains different electronic structure methods of varying degrees of accuracy and speed. The best-known methods are the ADF Density Functional Theory (DFT) code itself, and the BAND DFT code for periodic systems. FlexMD provides an interface toward both programs. For the interface with ADF, FlexMD makes use of classes from PyADF [2], a scripting framework for efficient quantum chemistry calculations. In addition to ADF and BAND, several semi-empirical methods are included in the ADF suite, such as DFTB and the NDDO type schemes available in the MOPAC package [3]. The ADF suite also provides classical mechanics methods, such as the reactive force field ReaxFF and the simple force field UFF. Interfaces to all of these methods are available in FlexMD. A simple example of a python script for MD using the **UFFForceJob** class for UFF calculations can be found in the examples directory, under `examples/scmlib/flexmd_uff_h2o`.

To increase the flexibility of FlexMD, an interface towards force calculations using the NAMD2.8 classical molecular dynamics package is provided (`examples/scmlib/namd_h2o`). NAMD2.8 is not distributed with the ADF suite, but it is available from a third party to be downloaded and installed (http://www.ks.uiuc.edu/Development/Download/download.cgi).

3. Multi-scale Molecular Dynamics

The design of the **ForceJob** class allows for flexible extension of its behavior, while at the same time keeping the client code unaware of its nature: it can either act as a simple wrapper for ADF programs, or it can be a more complex orchestrating class, combining simpler **ForceJob** classes to implement multi-scale strategies. One application of this extensible design can be found in the **QMMMForceJob** object, which combines a QM and an MM method in an IMOMM-type scheme (mechanical embedding). The **QMMMForceJob** object is assigned two other **ForceJob** objects, the first representing the high-resolution calculation (QM), while the other represents the low resolution (MM). Both **ForceJob** objects contain an **MDMolecule** object for the full molecular system. The selection of the QM-region is handled by the **QMMMForceJob**, which contains the information about the part of the molecule that constitutes the QM region. When forces are requested from the **QMMMForceJob**, the following behavior is orchestrated: first, a MM force calculation is performed on the full system; then, the QM-region is selected, a QM calculation is executed solely for that region, and energy and forces are added to those from the full system MM calculation. Finally, an MM calculation is computed forthe small QM-region, and the energy and forces are subtracted, yielding the final result, returned to the invoker. In symbols:

EQM/MM(Full) = EMM(Full) + EQM(QMRegion) – EMM(QMRegion)

The **QMMMForceJob** handles periodic boundary conditions if the low-level (MM) method supports this feature (i.e. NAMD). Whether the periodic interaction of the QM region with itself is handled at high or low resolution depends on the method used for the QM calculation. An example of QM/MM MD calculations can be found in the examples directory `examples/scmlib/qmmm_dftbUFF_2h2o`. The **QMMMForceJob** allows the use of link atoms when the QM boundary cuts through covalent bonds. However, this feature comes at the price of an increased script complexity. An example of a QMMM link-atom MD simulation is provided in the examples directory, under `examples/scmlib/qmmm_linkatom_dftbNAMD_glutamate`.

For more complex multi-scale calculations the **HybridForceJob** class can be used. This class allows the combination of a large set of different **ForceJob**s, each of them describing either the same, or different molecular systems. Each **ForceJob** can either involve a calculation on the full **MDMolecule** object it contains, or restricted to a specified region of the corresponding molecule. The forces from each contributing **ForceJob** can either be added or subtracted from the total force according to user preference, as specified at construction of the **HybridForceJob** object.

In order to perform QM/MM simulations on chemical reactivity in solution, it is important that the description of the solvent molecules can change on the fly, as the molecules move towards or away from the reactive region. To facilitate this, an **AdaptiveQMMMForceJob** class is available to provide adaptive QM/MM simulations using several available schemes, as described by Bulo et al.[4] In these schemes, the description of the diffusing molecules changes gradually from QM to MM and vice versa, based on the distance of those molecules to a predefined reactive site. Various schemes are available for assigning the QM and MM character of the molecules. The class contains a **QMMMForceJob** object, as well as a partitioning object that assigns the partial QM and MM character to the molecules. An examples python script for such an adaptive QM/MM simulation, using the DAS [4] method, is provided in the examples directory, `examples/scmlib/adqmmm_mopacscmUFF_h2o`.

4. Biased Molecular Dynamics

Constraints can be added to a simulation using the derived **ForceJob** class **WallJob**. The constraint is in the form of a large one-dimensional Gaussian on the potential energy surface, along a predefined Collective Variable (CV). Examples of CV's are the distance between two atoms, the coordination number of two atoms, but also more complex quantities such as the minimum distance between two sets of atoms, or the distance of an atom to a hydroxide ion. The Collective Variables can be specified through the **CollectiveVariable** class. Derived **CollectiveVariable** classes are available to specify sums or multiples of other **CollectiveVariable** objects.

Regular MD calculations are limited in the time-scales achievable with current hardware. The order of such time-scales is much smaller than what is required for chemical reactions. To overcome this problem, two rare-events methods have been implemented directly into the library: metadynamics [5] and umbrella sampling [6]. Both these methods involve biasing the simulations along a CV. An example of a metadynamics input can be found in the examples directory in `examples/scmlib/metadynamics_emt_h2o`.

For a wider range of rare-events methods, FlexMD also offers an interface with the PLUMED library for free energy calculations [7].To use this, a PLUMED input file is required, and for this we refer to the PLUMED manual. An example of a FlexMD input script using PLUMED can be found in the examples directory in `examples/scmlib/plumed_emt_h2o`.

5. Working with FlexMD

It is recomended to read the sections 1. Introduction and 2. Molecular Dynamics before working with FlexMD. Basic understanding of the Python Programming Language is also required. The Python website hosts documentation and a tutorial that can be used to learn Python.

The performance of the FlexMD package is difficult to predict because it depends on system size, the type of ForceJobs used and how these ForceJobs are combined. It is advised to first test the overhead of the FlexMD package for your system before running large simulations. When ab initio forces are involved, the overhead should not give a significant performance penalty. However, it may become a bottleneck when your system only uses cheap forcefields.

5.1 Creating a molecule object

FlexMD can be run trough the interactive python interpeter in the ADF package. To start it, run: `$ADFBIN/startpython` in a terminal, followed by:

```
>>> from scm import flexmd
```

Note that it is also possible, and usually more convenient, to write your FlexMD code in a file and then to execute this file. To do this, type all the commands you would use in the interactive interpeter in a file, and then enter `$ADFBIN/startpython myFlexMDjob.py` in a terminal (after changing to the directory where the file was stored of course).

Most FlexMD jobs will start with importing FlexMD and creating an MDMolecule object. This can be done by starting from a geometry in xyz or pdb format, or by manually adding the atoms in the FlexMDjob.py file. Geometries can be generated in the ADF GUI, and then be exported to xyz file. For more details on the MDMolecule object, run `$ADFBIN/startpython`, import flexmd and call `help(flexmd.mdmolecule)`.

```
>>> from scm import flexmd
>>> myMol = flexmd.MDMolecule('myGeometryFile.xyz')
```

Some ForceJobs require the system to be periodic. If we create an MDMolecule object from a pdb file that includes periodic information, the periodic boundary conditions are automaticly imported. If the information is not there, we can add it to the MDMolecule object:

```
>>> myMol = flexmd.pdb.set_box([50.0,25.0,100.0])
```

Info on set_box (and other functions, such as set_cellvectors, and write_pdb) can be found using `help(flexmd.pdbmolecule)`.

It is also possible to write the info in the MDMolecule object to a pdb file. to do so, call pdb.write_pdb('mypdbfile.pdb') on the myMol object:

```
>>> myMol.pdb.write_pdb('mypdbfile.pdb', box=True)
```

5.2 Creating a ForceJob

To specify what type of forces we want to use in the MD simulation, a ForceJob must be created. FlexMD has a number of ForceJobs (see PACKAGE CONTENTS in `help(flexmd)`), most of them with examples in `$ADFHOME/examples/scmlib`. The ForceJobs can be combined into a single ForceJob using `flexmd.hybrid_ForceJob`. As an example, we combine a reaxff_ForceJob with a metadynamicsjob and a walljob:

```
>>> from scm import flexmd
>>> myMol = flexmd.MDMolecule('myGeometryFile.xyz')
>>> myMol = flexmd.pdb.set_box([50.0,25.0,100.0])
    # setup our reaxff ForceJob and attach the forcefield file
    # (place the ff file in the same dir as the script and the xyz!)
>>> myReaxffForceJob = flexmd.ReaxffForceJob(molecule=myMol)
>>> myReaxffForceJob.settings.set_ff_filename('reax_forcefield_file.ff')

    # next we define the collective variable: the distance between atom 1 and 2
>>> myCvs = [flexmd.DistCV([1,2])]
    # create a set of metadynamics properties, using the CV
>>> mtdSettings = flexmd.MetadynamicsSettings(cvs=myCvs, widths=[0.30],
height=0.25 )
    # create the metadynamics job by combining the molecule, settings (with
CV)
    # and the number of md steps between depositing metadynamics hills.
>>> myMetadynamicsjob = flexmd.MetadynamicsJob( myMol, settings=mtdSettings,
nstep=150 )

    # add a wall to prevent the two atoms from drifting more than 10 Angstrom
away.
>>> myWalljob = flexmd.WallJob(molecule=myMol, cvs=myCvs, cntrs=[10.0],
widths=[1.0], heights=[500.0])

    # combine the forces into a hybrid job that will be used for the MD
>>> myForceJob = flexmd.HybridForceJob( [[myReaxffForceJob,'+'],
[theMetadynamicsjob,'+'], [theWalljob,'+']], myMol )
```

Note that all the ForceJobs require some special input and settings, and that these settings can be applied both before and after defining the ForceJob. For the reaxffForceJob, we first define the ForceJob, and add the forcefield parameters file afterwards. For the metadynamics job we reverse this, and first create a metadynamicsJobSettings object, which is then used in the creation of the metadynamics job. For more detailed info on the different ForceJobs and their inputs, see the help function by calling help on a ForceJob, for example: `help(flexmd.ReaxffForceJob)` or `help(flexmd.WallJob)`. Also remember that other examples of ForceJobs can be found in `$ADFHOME/examples/scmlib`.

5.3 Creating and running the MD job

Before the simulation can be set in motion, a propagator is needed. The propagatorJob controls simulation settings such as temperature and timestep size. FlexMD uses the Atomistic Simulation Environment (ASE) [1] for this. The MDPropagatorJob object is created just like the other objects in FlexMD:

```
    # do this after importing flexmd and creating a ForceJob.
    # it creates the MDPropagator job, with some settings
>>> myMDJob = flexmd.ASEMDPropagatorJob( ForceJob=myForceJob )
>>> myMDJob.settings.set_tempcontrol( True, nhfreq=2, maxdef=50.0 )
>>> myMDJob.settings.set_temperature(300.0)
>>> myMDJob.settings.set_timestep( 0.02 )
```

For more details on the ASEMDPropagatorJob, view it's help page:
`help(flexmd.ASEMDPropagatorJob)`, or take a look at the MDSettings object:
`help(flexmd.MDSettings)`.

The propagatorJob can be used to create an MDManager object:

```
    # create an MD manager
>>> myManager = flexmd.MDManager( mdjob=myMDJob)
```

The manager object is now in control of the MD simulation, and we can use it to run the simulation for a number of steps:

```
    # tell the MD manager to run the simulation
>>> myManager.run( ncycles = 2500 )
```

Note that the number of steps here should be increased a lot if metadynamics effects are to be observed, but it is always wise to first run a small number of steps to check if everything works. Some information will be printed during the simulation, depending on the settings of the components used. The manager will also create some folders in the working directory, and store the data produced by the simulation in there.

The full flexmd jobfile should now look something like this:

```
>>> from scm import flexmd
>>> myMol = flexmd.MDMolecule('myGeometryFile.xyz')
>>> myMol = flexmd.pdb.set_box([50.0,25.0,100.0])
    # setup our reaxff ForceJob and attach the forcefield file
    # (place the ff file in the same dir as the script and the xyz!)
>>> myReaxffForceJob = flexmd.ReaxffForceJob(molecule=myMol)
>>> myReaxffForceJob.settings.set_ff_filename('reax_forcefield_file.ff')

    # next we define the collective variable: the distance between atom 1 and 2
>>> myCvs = [flexmd.DistCV([1,2])]
    # create a set of metadynamics properties, using the CV
>>> mtdSettings = flexmd.MetadynamicsSettings(cvs=myCvs, widths=[0.30],
height=0.25 )
    # create the metadynamics job by combining the molecule, settings (with
CV)
    # and the number of md steps between depositing metadynamics hills.
>>> myMetadynamicsjob = flexmd.MetadynamicsJob( myMol, settings=mtdSettings,
nstep=150 )

    # add a wall to prevent the two atoms from drifting more than 10 Angstrom
away.
>>> myWalljob = flexmd.WallJob(molecule=myMol, cvs=myCvs, cntrs=[10.0],
widths=[1.0], heights=[500.0])

    # combine the forces into a hybrid job that will be used for the MD
>>> myForceJob = flexmd.HybridForceJob( [[myReaxffForceJob,'+'],
[theMetadynamicsjob,'+'], [theWalljob,'+']], myMol )

    # do this after importing flexmd and creating a ForceJob.
    # it creates the MDPropagator job, with some settings
>>> myMDJob = flexmd.ASEMDPropagatorJob( ForceJob=myForceJob )
>>> myMDJob.settings.set_tempcontrol( True, nhfreq=2, maxdef=50.0 )
>>> myMDJob.settings.set_temperature(300.0)
>>> myMDJob.settings.set_timestep( 0.02 )

    # create an MD manager
>>> myManager = flexmd.MDManager( mdjob=myMDJob)
    # tell the MD manager to run the simulation
>>> myManager.run( ncycles = 2500 )
```

References

1. S. R. Bahn, K. W. Jacobsen, *An object-oriented scripting interface to a legacy electronic structure code.* Comput. Sci. Engin. **4**, 56-66 (2002). The Atomistic Simulation Enviroment website and documentation

2. C. R. Jacob, S. M. Beyhan, R. E. Bulo, A. Severo Pereira Gomes, A. W. Gotz, K. Kiewisch, J. Sikkema, L. Visscher, *PyADF - A scripting framework for multiscale quantum chemistry.* J. Comput. Chem. **32**, 2328-2338 (2011)

3. J. J. P. Stewart, *Optimization of parameters for semiempirical methods IV: extension of MNDO, AM1, and PM3 to more main group elements.*, J. Mol. Model. **10**, 155-164 (2004)

4. R. E. Bulo, B. Ensing, J. Sikkema, L. Visscher, *Toward a Practical Method for Adaptive QM/MM Simulations.* J. Chem. Theory Comput. **5**, 2212-2221 (2009)

5. A. Laio, M. Parrinello, *Escaping free-energy minima.* Proc. Natl. Acad. Sci. USA., **99**, 12562-12566 (2002)

6. B. Roux, *The calculation of the potential of mean force using computer simulations.* Comput. Phys. Commun. **91**, 275-282 (1995)

7. M. Bonomi, D. Branduardi, G. Bussi, C. Camilloni, D. Provasi, P. Raiteri, D. Donadio, F. Marinelli, F. Pietrucci, R. A. Broglia, M. Parrinello, *PLUMED: A portable plugin for free-energy calculations with molecular dynamics.* Comp. Phys. Comm. **180**, 1961-1972 (2009). PLUMED website and documentation