Gromorg Documentation

Abel Carreras

Dec 15, 2022

CONTENTS

1 Introduction 3

2 Installation 5

3 Get started 7

4 Solvent 11

5 Parameters 13

A python interface to automate the calculation of molecular dynamics simulations of organic molecules using GROMACS.

CHAPTER

ONE

INTRODUCTION

Gromorg in a python interface for Gromacs, a popular software to simulate molecular dynamics using molecular mechanics originally developed within the Biophysical Chemistry department at University of Groningen.

Gromorg aims to provide a simple interface to automate the process of simulating molecular dynamics of small organic molecules from simple structural information. This includes the determination of the molecular connectivity, the obtainment the force field parameters, the build of the unitcell/supercell, the solvation of the system and the parsing of the outputs.

The philosophy of Gromorg is to provide a simple interface to link MD simulations performed with Gromacs with other tools using python. An example is its close connection with PyQchem (https://github.com/abelcarreras/PyQchem), a python interface for Q-Chem.

1.1 Main features

- Link first principles & molecular mechanics calculations using PyQchem.
- Get parameters from SwissParam (https://www.swissparam.ch) automatically from molecular structure
- Clean run without intermediate files
- Add solvent molecules to the system
- Extract structures from the trajectory (including surrounding solvent molecules)

CHAPTER

TWO

INSTALLATION

Gromorg can be installed directly from the source code (python package) or via PyPI repository.

2.1 Requirements

- PyQchem (https://github.com/abelcarreras/PyQchem)
- Gromacs (gmxapi) (http://www.gromacs.org)
- Openbabel [Tested on v2.4.1] (python API) (http://openbabel.org)
- MDtraj (https://www.mdtraj.org)

2.2 Install

1) From source code

```
python setup.py install --user
```

2) From PyPI repository

```
pip install gromorg --user
```

CHAPTER

THREE

GET STARTED

3.1 Defining molecule

The definition of the molecule is done creating an instance of PyQchem Structure class. See PyQchem manual for more details. Charge and multiplicity are ignored.

Example of ethylene:

```
from pyqchem.structure import Structure

structure = Structure(coordinates=[[ 0.6695, 0.0000, 0.0000],
                                   [-0.6695, 0.0000, 0.0000],
                                   [ 1.2321, 0.9289, 0.0000],
                                   [ 1.2321,-0.9289, 0.0000],
                                   [-1.2321, 0.9289, 0.0000],
                                   [-1.2321,-0.9289, 0.0000]],
                      symbols=['C', 'C', 'H', 'H', 'H', 'H'])
```

3.2 Defining GROMACS input parameters

GROMACS input parameters are defined in a dictionary. For example:

```
gmx_params = {
              'integrator': 'md-vv',      # Verlet integrator
              'nsteps': 10000,            # 0.001 * 10000 = 100 ps
              'dt': 0.001,                # time step, in ps
              # Temperature coupling is on
              'tcoupl': 'nose-hoover',     # Nose-Hoover thermostat
              'tau_t': 0.3,                # time constant, in ps
              'ref_t': 300,                # reference temperature, one for each group, in␣
↪K
              # Bond parameters
              'gen_vel': 'yes',            # assign velocities from Maxwell distributio
              'gen_temp': 300,             # temperature for Maxwell distribution
              'gen_seed': -1,              # generate a random seed
              }
```

The defined parameters override/add to the default values set by gromorg. This way for most purposes it is not necessary to define most parameters. These default values are:

```
'integrator': 'md-vv',      # Verlet integrator
'nsteps': 5000,             # 0.001 * 5000 = 50 ps
'dt': 0.001,                # ps
# Output control
'nstxout': 1,               # save coordinates every 0.001 ps
'nstvout': 1,               # save velocities every 0.001 ps
'nstenergy': 1,             # save energies every 0.001 ps
'nstlog': 100,              # update log file every 0.1 ps
# Bond parameters
'continuation': 'no',        # first dynamics run
'cutoff-scheme': 'Verlet',   # Buffered neighbor searching
'verlet-buffer-tolerance': 3.3e-03,
# 'ns_type': 'grid',          # search neighboring grid cells
'nstlist': 10,               # 20 fs, largely irrelevant with Verlet
'rcoulomb': 1.0,             # short-range electrostatic cutoff (in nm)
'rvdw': 1.0,                 # short-range van der Waals cutoff (in nm)
'DispCorr': 'EnerPres',      # account for cut-off vdW scheme
# Electrostatics
'coulombtype': 'PME',        # Particle Mesh Ewald for long-range electrostatics
'pme_order': 4,              # cubic interpolation
'fourierspacing': 0.16,      # grid spacing for FFT
# Temperature coupling is on
'tcoupl': 'nose-hoover',     # Nose-Hoover thermostat
'tc-grps': 'system',         # one coupling group
'tau_t': 0.3,                # time constant, in ps
'ref_t': 100,                # reference temperature, one for each group, in K
# Pressure coupling is off
'pcoupl': 'no',              # no pressure coupling in NVT
# Periodic boundary conditions
'pbc': 'xyz',                # 3-D PBC
# Velocity generation
'gen_vel': 'yes',            # assign velocities from Maxwell distributio
'gen_temp': 10,              # temperature for Maxwell distribution
'gen_seed': -1,              # generate a random seed
```

3.3 Setting up the calculation

Example of simple parallel(openMP) calculation using 4 threads:

```
calc = GromOrg(structure,
               params=gmx_params,        # MDP parms
               box=[10, 10, 10],         # unitcell a, b, c in angstrom
               angles=[90, 90, 90],      # unitcell alpha, beta, gamma in degree
               supercell=[3, 3, 3],      # size of supercell
               delete_scratch=True,      # if true delete temp files when finished␣
↪(default: True)
               silent=False,             # if true print MD log info in screen (default:␣
↪False)
               omp_num_threads=False,    # number of parallel threads used (default: 1)
               maxwarn=0,                # max number of GROMACS warnings (default: 0)
               )
```

The defined structure correspond to the unit cell and it is replicated according to the `supercell` argument. This means that while usually will contain a single molecule, it is possible to include several units particularly to define crystals. The number of MPI processes is automatically set according to the cores available and the `omp_num_threads` parameter. The calculation is run in a temporary folder in the current working directory. This directory is deleted by default when the calculation is finished. `delete_scratch` keyword can be set to change this behavior.

`maxwarn` keyword can be used to set the maximum number of warnings GROMACS can ignore before aborting the calculation. It is recommended to keep this value to 0 (default) and only change it once you have revised all warnings and made sure you can ignore them.

3.4 Run the calculation

Once the calculation is set up, it can be run with the following command:

```
trajectory, energy = calc.run_md(whole=True)  # unwraps the trajectory
```

The return of this function is the trajectory as a MDtraj object and the energy as a dictionary. MDtraj is a flexible format to store trajectory data. Check the documentation of MDtraj for more information. (https://www.mdtraj.org/1.9.5/load_functions.html). The only argument of this function is `whole`, this optional arguments controls the unwrapping of the trajectory. That is, if `whole` is True, the trajectory is unwrapped such that the molecules are shown as whole for each step of the trajectory.

A simple way to visualize the trajectory is to store it in the disk as a common format. This can be done using `save` method:

```
trajectory.save('trajectory.gro')
```

MDtraj supports different formats, such as GROMACS (gro), Protein Data Bank (pdb) and xyz.

Energy dictionary contains the total energy, the kinetic energy and the potential energy as lists. This can be plotted, for example, as:

```
import matplotlib.pyplot as plt
plt.plot(energy['potential'], label='potential')
plt.plot(energy['kinetic'], label='kinetic')
plt.plot(energy['total'], label='total')
plt.legend()
plt.show()
```

3.5 Extracting molecules from trajectory

One of the main applications of Gromorg is use the geometries of the molecules within the trajectory. For this purpose gromorg package provides some functions to do this. `mdtraj_to_pyqchem` function is used to extract a particular molecule from a trajectory.

```
from gromorg.tools import mdtraj_to_pyqchem

molecule = mdtraj_to_pyqchem(trajectory, ifame, ires, center=True)
```

where `trajectory` is the trajectory as a MDtraj object, `ifame` is the index of the frame and `ires` is the index of the residue. `center` is an optional argument that centers the molecule in the origin. Each residue correspond to a unit cell (*structure* in the first example), this means that in the case that the unit cell contains multiple molecules, the separation of them will need to be done manually.

The return of this function is a PyQChem Structure containing the molecule. This object can be used directly with PyQchem to perform quantum chemistry calculations. For example:

```
from gromorg.tools import mdtraj_to_pyqchem
from pyqchem import QchemInput, get_output_from_qchem

qc_input = QchemInput(molecule,
                      jobtype='sp',
                      exchange='hf',
                      basis='6-31G')

output = get_output_from_qchem(qc_input)

print(output)
```

CHAPTER

FOUR

SOLVENT

4.1 Defining solvent molecule

Solvent molecules can be easily added in analogous way to the main molecule. First the solvent molecule is defined as a PyQchem Structure object.

Example of water:

```
solvent = Structure(coordinates=[[0.000000,  0.000000,  0.000000],
                                 [0.758602,  0.000000,  0.504284],
                                 [0.758602,  0.000000,  -0.504284]
                                 ],
                     symbols=['O', 'H', 'H'])
```

then the solvent molecule is added during the calculation setup:

```
calc = GromOrg(structure,
               params=gmx_params,        # MDP parms
               box=[30, 30, 30],         # unitcell a, b, c in angstrom
               supercell=[1, 1, 1],      # size of supercell
               solvent=solvent,          # solvent molecule
               solvent_scale=0.57,       # solvent scale parameter
               )
```

To adjust the density of the solvent molecule, the solvent_scale parameter can be used. This correspond to the `-scale` option in the `gmx solvate` command. (https://manual.gromacs.org/current/onlinehelp/gmx-solvate.html?highlight=solvate)

4.2 Extracting molecular cluster from trajectory

To extract the molecules from the trajectory `mdtraj_to_pyqchem` can be used as previously. However, when including the solvent it becomes convenient to extract not only the main molecule but also some of closest solvent molecules. For this purpose `get_cluster` function can be used.

```
cluster = get_cluster(trajectory, iframe, ires, cutoff=5.0, center=True)
```

where `iframe` is the frame number, `ires` is the residue number and `cutoff` is the distance cutoff. Also, `center` is a boolean parameter that can be used to center the molecule on the origin. The return value is a PyQchem Structure object that contains the molecule corresponding to ires and the molecules surrounding it at less than `cutoff` distance.

Note: Notice that in the trajectory the first residue indices always correspond to the main molecule while the others are solvent molecules. Using the value defined in `supercell` the user can figure out how many main molecules are present in the trajectory.

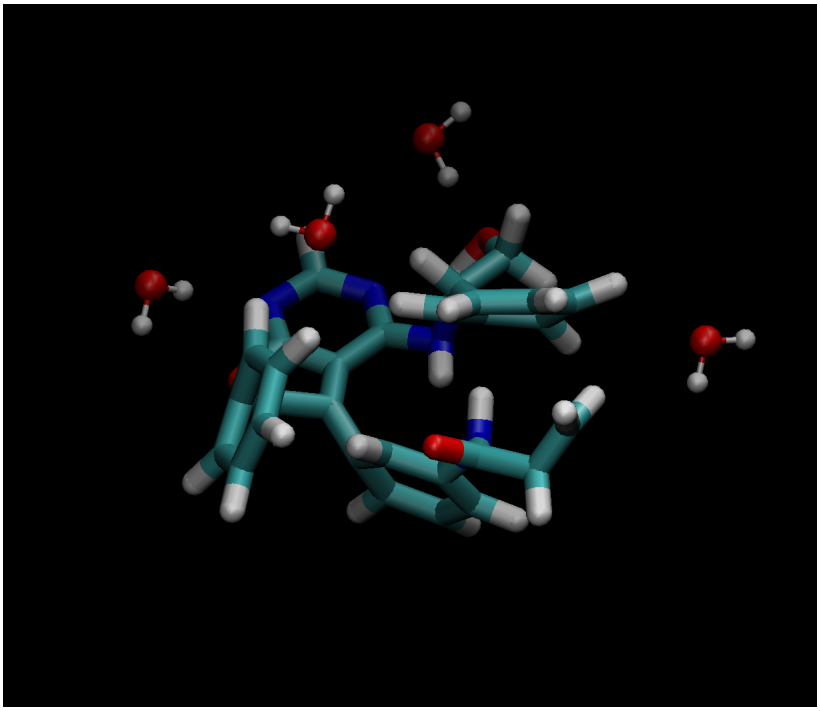

Fig. 1: Simple example of the Structure (main mol + solvent) obtained using `get_cluster` function.

CHAPTER

FIVE

PARAMETERS

The parameters are obtained automatically using SwissParam service (https://www.swissparam.ch). This is done in a seamlessly without user intervention. The molecular structure is transformed to *.mol2* format and the connectivity is generated using OpenBabel library. Then the molecule is updated to SwissParam servers and the parameters are downloaded. This process requires internet connection and is usually very fast for small molecules.

Once the parameters are obtained, they are stored in a cache file (.parameters.pkl) in the working directory. This file can contain the parameters for multiple molecules and it is used for all calculations performed in the same directory. This file can be used to run calculations offline.

5.1 Custom parameters

In some cases, the parameters obtained from SwissParam servers are not suitable for the calculation. In this case, the user can provide custom parameters computed elsewhere. The custom parameters should be written in an gromacs *.itp* formatted file. Also it is necessary to include a *.pdb* formatted file of the structure with the same atom labels than the *.itp* file. Then the following function can be used to generate a *.parameters.pkl* file containing the parameters:

```
from gromorg.setparam import SetParams
from pyqchem.structure import Structure

# Initialize the SetParams with database file (append data if file exists)
data = SetParams(filename='.parameters.pkl')

# Add parameters to database
data.add_data('ethylene.itp', 'ethylene.pdb')  # ethylene
```

An example is provided in the *examples* directory (https://github.com/abelcarreras/gromorg/blob/master/examples/set_params.py).

5.2 Example data

5.2.1 ethylene.itp

```
; ----
; Built itp for test.mol2
;    by user vzoete     Wed Jul 14 16:04:50 UTC 2021
; ----
;
```

(continues on next page)

(continued from previous page)

```
[ atomtypes ]
; name at.num  mass   charge  ptype    sigma            epsilon
C=C     6   12.0110  0.0  A         0.372396    0.284512
HCMM    1    1.0079  0.0  A         0.235197    0.092048


[ pairtypes ]
;  i     j    func     sigma1-4       epsilon1-4 ; THESE ARE 1-4 INTERACTIONS

[ moleculetype ]
; Name nrexcl
test 3

[ atoms ]
; nr type resnr resid atom cgnr charge mass
   1 C=C  1  LIG C       1 -0.3000  12.0110
   2 C=C  1  LIG C1      2 -0.3000  12.0110
   3 HCMM 1  LIG H       3  0.1500   1.0079
   4 HCMM 1  LIG H1      4  0.1500   1.0079
   5 HCMM 1  LIG H2      5  0.1500   1.0079
   6 HCMM 1  LIG H3      6  0.1500   1.0079

[ bonds ]
; ai aj fu b0 kb, b0 kb
  1   2 1 0.13330  572403.8  0.13330  572403.8
  1   3 1 0.10830  311344.8  0.10830  311344.8
  1   4 1 0.10830  311344.8  0.10830  311344.8
  2   5 1 0.10830  311344.8  0.10830  311344.8
  2   6 1 0.10830  311344.8  0.10830  311344.8

[ pairs ]
; ai aj fu
  3   5 1
  3   6 1
  4   5 1
  4   6 1

[ angles ]
; ai aj ak fu th0 kth ub0 kub th0 kth ub0 kub
  2   1   3 1  121.0040  322.18    121.0040  322.18
  2   1   4 1  121.0040  322.18    121.0040  322.18
  3   1   4 1  119.5230  219.80    119.5230  219.80
  1   2   5 1  121.0040  322.18    121.0040  322.18
  1   2   6 1  121.0040  322.18    121.0040  322.18
  5   2   6 1  119.5230  219.80    119.5230  219.80

[ dihedrals ]
; ai aj ak al fu phi0 kphi mult phi0 kphi mult
  3   1   2   5 9 180.00  25.1040 2   180.00  25.1040 2
  3   1   2   6 9 180.00  25.1040 2   180.00  25.1040 2
  4   1   2   5 9 180.00  25.1040 2   180.00  25.1040 2
```

(continues on next page)

(continued from previous page)

```
  4   1   2   6 9 180.00  25.1040 2   180.00  25.1040 2

[ dihedrals ]
; ai aj ak al fu xi0 kxi xi0 kxi
  1   3   2   4 2   0.00   3.6150     0.00   3.6150
  2   5   1   6 2   0.00   3.6150     0.00   3.6150


#ifdef POSRES_LIGAND
[ position_restraints ]
; atom  type      fx      fy      fz
   1 1 1000 1000 1000
   2 1 1000 1000 1000
#endif
```

Note: Notice that `moleculetype` name must be **test**.

5.2.2 ethylene.pdb

```
REMARK  FOR INFORMATIONS, PLEASE CONTACT:
REMARK  ZOETE VINCENT
REMARK  VINCENT.ZOETE_AT_ISB-SIB.CH
REMARK  SWISS INSTITUTE OF BIOINFORMATICS
REMARK  MOLECULAR MODELING GROUP
REMARK  QUARTIER SORGE - BATIMENT GENOPODE
REMARK  CH-1015 LAUSANNE
REMARK  SWITZERLAND
REMARK  T: +41 21 692 4082
REMARK ***************************************************************
REMARK   DATE:     7/14/21     16: 4:50      CREATED BY USER: root
ATOM      1  C   LIG     1       0.669   0.000   0.000  1.00  0.00      LIG
ATOM      2  C1  LIG     1      -0.669   0.000   0.000  1.00  0.00      LIG
ATOM      3  H   LIG     1       1.232   0.929   0.000  1.00  0.00      LIG
ATOM      4  H1  LIG     1       1.232  -0.929   0.000  1.00  0.00      LIG
ATOM      5  H2  LIG     1      -1.232   0.929   0.000  1.00  0.00      LIG
ATOM      6  H3  LIG     1      -1.232  -0.929   0.000  1.00  0.00      LIG
TER       7      LIG      1
END
```

Gromorg is being developed by Abel Carreras within the Molecular Electronic Structure Group at Donostia International Physics Center (DIPC), Euskadi, Spain.