PyPol Manual

Nicholas Francia

April 18, 2021

Chapter 1

Installation

This library works only on Linux OS.

1.1 Requirements: Python programs

All required dependencies should be installed before running PyPol:

- Python3
- numpy & scipy
- matplotlib
- openbabel
- progressbar2
- networkx

1.2 Requirements: Non-Python programs

All the following programs should be installed before running PyPol:

- GROMACS(>=2020)
- AmberTools
- PLUMED2 with the crystallization module

- PLUMED2 - Hack The Tree
- InterMol

When you first use PyPol it will ask for the paths of all these programs binaries and save them for future use. You can insure Python can find pypol.py by adding The PyPol directory to PYTHONPATH with the command:

```
export PYTHONPATH=''$PYTHONPATH:<path_to_pypol>/PyPol/''
```

If you set the path to this file as environment variables, you only have to do it once. You can add the same string to your *.bashrc* file.

Chapter 2

Quick Overview

This chapter is a quick introduction to PyPol and its features.

2.1 Create and manage a new project

The project object contains all the relevant information about the simulations and the different methods used. Initially, you have to create a new Project object by specifying the path of new directory. The function pypol.new_project will make that directory plus a series of other directories that will be used by the program. These are:

- Input: stores the initial structures and input files for the different methods
- Output: stores the output files from pypol modules
- data: contain all the structure folders in which simulations are performed

Most of the objects in pypol have help() functions that shows the accessible attributes and modules together with some examples. Everytime the Project object is modified you must write <Project>.save() at the end of your script to save every variation to a pickle file in the project folder.
- Create a new project and print the help() function:

```
from PyPol import pypol as pp
# Creates a new project in folder /home/Work/Project/
project = pp.new_project(r'/home/Work/Project/', name=project1)
# Print available attributes, methods and examples
project.help()
# Save project to be used later
project.save()
```

Once saved, the Project object can be retrieved by loading it with the function pypol.load_project specifying only the project path. If the project folder is moved to another position or computer, you can change the project directory by changing the attribute Project.working_directory.

- Load an existing project and print information

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Name of the project
print(project.name)
# Path to the project folder
print(project.working_directory)
# Save project to be used later
project.save()
```

- Change the working directory. Copy/Move the project in the new path and then run:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/New_Project_Path/')
# Change the working directory. This might take a while
project.working_directory = r'/home/Work/New_Project_Path/'
# Save project to be used later
project.save()
```

Structures can be added to the project with the Project.add_structures module. This takes the path of a single structure or a folder containing different structures and copy them to the Input/Initial_Structures folder. Then, they are converted to Crystal objects to be used later. During this process structures are converted to the PDB file format with openbabel and the atomtype of each atom is detected with the AmberTools program "atomtype". Files that do not have a format extension compatible with openbabel are ignored.

- Add new crystal structures to the project:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Add all crystal structures in the given folder
project.add_structures(r'/home/Work/Structures/')
# Add the specified crystal structure
project.add_structures(r'/home/Work/Structures/str1.pdb')
# List of structures imported in the project
print(project.initial_crystals)
# Save project to be used later
```

```python
project.save()
```

After importing the initial set of structures, we need to create a new Method object to run the simulations. Every method is characterised by the MD package used (only gromacs for now), the molecular forcefield and the list of simulations to be used. Method objects can be created with the Project.new_method function. In the beginning, you just need to specify a label for that method. This can be used to get back that method later on or delete it. Methods are stored in the Project.methods list.

- Create a new method and print its manual:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Creates a new method
gaff = project.new_method('GAFF')
# Print new method manual
gaff.help()
# Save project to be used later
project.save()
```

- Retrieve an existing method:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Get an existing method
gaff = project.get_method('GAFF')
# Print method MD package
print(gaff.package)
# Save project to be used later
project.save()
```

- Delete an existing method:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Delete an existing method
project.del_method('GAFF')
# Save project to be used later
project.save()
```

Once created, the Method object is empty and must be integrated with the topology that will be used in simulations. PyPol requires the molecular .itp file format and the coordinate file of the isolated molecule from which the forcefield is created. The topology file can be added but the only the [defaults] section will be imported as the [system] and [molecules] section depends on the molecular forcefield and the number of molecules in each crystal. The coordinate file of the isolated molecule is used to reindex every molecule and force them to have the same order of the one specified in the itp file. Molecular forcefield are imported in the Project with the module Project.Method.new_molecule. In this module, also the potential energy of the isolated molecule can be added and it will be used to calculate the lattice energy of each crystal. Finally, the Project.Method.generate_input module will replicate the unit cells to have supercells of the closest dimensions to the ones specified in the box parameter. If the parameter *orthogonalize* is set to True, the most orthogonal non-primitive cell is used instead of the unit cell. After this the data folder will contain all the supercells in the GRO and PDB file formats and the topology files with the correct parameters.

- Import Forcefield and generate structures:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Creates a new method
gaff = project.new_method('GAFF')
# Topology file to be used in simulations
path_top = '/home/Work/Forcefield/topol.top'
# Molecular forcefield file
path_itp = '/home/Work/Forcefield/MOL.itp'
# Isolated molecule file.
path_crd = '/home/Work/Forcefield/molecule.mol2'
# Copy relevant part of the topology to the project folder
gaff.new_topology(path_top)
# Copy molecular forcefield of molecule "MOL"
gaff.new_molecule(path_itp, path_crd, -100.0000)
# Generates the input files for the simulation
gaff.generate_input(box=(50., 50., 50.), orthogonalize=True)
# Save project to be used later
project.save()
```

2.2 Run Simulations and import results

PyPol has 4 different simulation objects that differ in the way inputs are created and results are read by the program. These can be created with the Method.new_simulation function by specifying the required Simulation Object:

- "em": Energy minimization using Gromacs. If no mdp file is specified, the default one is used.
- "cr": Cell relaxation using LAMMPS. If no input or forcefiled file are specified, a new topology is obtained converting the Gromacs one with InterMol.
- "md": Molecular Dynamics using Gromacs. If no mdp file is specified the default ones are used. Check the PyPol/data/Defaults/Gromacs folder to see or modify them.

- "wtmd": Well-Tempered Metadynamics simulations.

In general, all the simulation objects have:

- help() module with specifics and examples
- generate_input() module to copy the mdp file to all the folders in the data directory and to generate a bash script to run all the simulations.
- get_results() module to verify the normal termination of the simulation and calculate the lattice or potential energy

The first one is the EnergyMinimization object can be used with all the Gromacs energy minimization algorithms. To use default mdp files use the names "em" and "relax". Alternatively, specify the path to the Gromacs mdp file to be used. Default mdp files are stored in the PyPol installation directory in PyPol/data/Defaults/Gromacs and can be modified if needed. If bash_script is set to True, ascript to run all simulations is created in:
<project_folder>/data/<method_name>/run_<simulation_name>.sh
After performing all simulations, the get_results module calculate the lattice energy of the crystal and change the crystal status from "incomplete" to "complete".

- Create a new EnergyMinimization object using default mdp files and print its manual:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Create a new simulation using defaults em.mdp file
em = gaff.new_simulation("em", simtype="em")
# Print new simulation manual
em.help()
# Save project to be used later
project.save()
```

- Alternatively, create a new EnergyMinimization object using a specified mdp file:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Gromacs mdp file
path_mdp = '/home/Work/InputFiles/em.mdp'
# Create a new simulation
em = gaff.new_simulation("em", "em", path_mdp)
```

```python
# Save project to be used later
project.save()
```

- Generate inputs for all the crystals, including a bash script to run all simulations:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve an existing simulation
em = gaff.get_simulation("em")
# Copy the MDP file in each crystal directory
em.generate_input(bash_script=True)
# Save project to be used later
project.save()
```

- Check the normal termination of each simulation and get the energy landscape:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve an existing simulation
em = gaff.get_simulation("em")
# Check normal termination and import potential energy
em.get_results()
# Print the lattice energy of each crystal
for crystal in em.crystals:
    print(crystal.name, crystal.energy)
    # Create personalised labels for some structures
    if crystal.name == "E1":
        crystal.label = "Form I"
    if crystal.name == "E2":
        crystal.label = "Form II"
# Plot the crystal energy landscape. The legend will include forms I and II
em.plot_landscape("LatticeEnergyLandscape.png")
# Save project to be used later
project.save()
```

Since Gromacs does not allow for the simulation box to relax during the energy minimization, the optimised structure can be converted to the LAMMPS format for which this feature is available. The topology conversion for a single molecule is done using InterMol and then replicated for each molecule inside the crystal. As before, the get_results module is used to check the simulations and upload the results. A good approach is to optmize the initial atomic positions with Gromacs according to the imported topology, then relax the cell parameters using LAMMPS and finally rerun an energy minimization simulation with gromacs. The last step is done to avoid possible differences between the two MD packages.

- Create a new CellRelaxation object, using InterMol to generate LAMMPS topology:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Create a new simulation
cr = gaff.new_simulation("cr", "cr")
# Save project to be used later
project.save()
```

MD simulations inputs are created by the MolecularDynamics object. Default mdp files can be used by typing the following names without specifying any mdp path:

- "nvt": 3ns simulation in the NVT ensemble
- "berendsen": 1ns simulation in the NPT ensemble using the Berendsen barostat
- "parrinello": 3ns simulation in the NPT ensemble using the Parrinello-Rahman barostat
- "md": 5ns simulation in the NPT ensemble using the Parrinello-Rahman barostat

Equivalent "mdvvnvt", "mdvvberendsen", "mdvvparrinello", "mdvvmd" are also available that uses the velocity Verlet algorithm for integrating Newton's equations of motion.
- Create a new MolecularDynamics objects:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Gromacs mdp file
path_mdp = '/home/Work/InputFiles/nvt.mdp'
# Create a new simulation
nvt = gaff.new_simulation("nvt", "md", path_mdp)
# Save project to be used later
project.save()
```

WTmetaD simulations inputs and the analysis of biased simulations can be done with the Metadynamics object. The default mdp file can be used using the name "wtmd". This is equivalent to the "md" one but no step termination is specified. Once created, the PyPol will ask you if you want to use the default sets of collective variables used for WTmetaD. This includes:

- Potential Energy and Density. This uses the CELL and ENERGY Plumed CVs to calculate the density and the potential energy rescaled by the number of molecules.

- Simulation Box Angles. This uses the CELL Plumed CV and add bias on the non-diagonal elements of the simulation box.

The plumed metadynamics parameters can also be changed using the class attributes: biasfactor, pace, height, temp, stride. The grid min and max are automatically taken considering the crystals involved in the analysis, but can be modified updating the CVs objects. An energy cutoff to the added bias is by default set to 2.5 kJ mol $^{-1}$ but it can be modified trough the class property "energy_cutoff". In addition, a distance RMSD analysis can be added that stops the simulation when the a certain distance cutoff is reached. The value of the cutoff can be identified by looking at its variations on stable MD trajectories. - Create a new Metadynamics objects:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Gromacs mdp file
path_mdp = '/home/Work/InputFiles/meta.mdp'
# Create a new simulation using only the cluster centers and select CVs to be used.
meta = mdvv.new_simulation("meta", "wtmd", crystals="centers")
# Modify WTMD parameters
meta.drmsd = True
meta.drmsd_toll = 0.25
meta.height = 3.0
meta.temp = 300
meta.pace = 1000
meta.energy_cutoff = 2.0
# Generate inputs
meta.generate_input(bash_script=True)
# Save project to be used later
project.save()
```

A clustering analysis can be performed at regular intervals of work using the "generate_analysis_input" and "get_analysis_results" modules. - Generate analysis inputs based on specific clustering method (set of fingerprints, see next section)

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing simulation and clustering parameters objects
gaff = project.get_method('GAFF')
meta = gaff.get_simulation('meta')
cp = gaff.get_clustering_parameters("CM1")
```

```
cp.sigma_cutoff = 0.15
# Generate analysis inputs every 0.5 kJ mol-1 of added bias
meta.generate_analysis_input(cp, start=0.5, end=2.0, interval=0.5)
# Save project to be used later
project.save()
```

- Get results of the analysis:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method and simulation
gaff = project.get_method('GAFF')
meta = gaff.get_simulation('meta')
# Get results:
meta.get_analysis_results(plot=False) # Avoid plotting the distributions of each crystal
# Save project to be used later
project.save()
```

2.3 Trajectory analysis

2.3.1 Distributions: Molecular orientation and identification of melted structures

Between an MD step and another, it is useful to check if some of the structures are melted. A good way to do it is to check the orientational disorder of the molecules inside the crystal. We can identify one vector that connects two atoms for each molecule and calculate the intermolecular angle between each of the pairs. The distribution of these angles should exhibit well defined peaks if the system is in the crystal form. Melted structures have molecules oriented randomly and a distribution trend given by 0.5*sin(x). By comparing this reference distribution with the structure one, melted structures are easily identified. Molecular orientation distributions can also be used as part of the crystal fingerprints in the clustering analysis.

- Select atoms of the orientational vectors and create plumed inputs:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule("MOL")
# Retrieve the CV Object
mo = gaff.get_cv("mo")
# Use the first two atoms to define the orientational vect
```

```
mo.set_atoms((0, 1), molecule)
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generate plumed driver input for the selected simulation
mo.generate_input(npt)
# Save project
project.save()
```

- Import distributions once the plumed driver analysis is finished:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule("MOL")
# Retrieve the CV Object
mo = gaff.get_cv("mo")
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generate plumed driver input for the selected simulation
mo.get_results(npt, plot=False)
# Save project
project.save()
```

- Check orientational disorder:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve the CV Object
mo = gaff.get_cv("mo")
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Identify melted structures
mo.identify_orientational_disorder(npt)
# Save project
project.save()
```

2.3.2 Distributions: Radial Distribution Function

The RDF object generates plumed inputs to calculate the Radial Distribution Function of molecules in each crystal. The coordinates to use can be the geometrical center of the molecule or the center of mass. Hydrogen atoms can be ignored in this calculation. The resulting distribution can be used as part of the crystal fingerprint in the clustering analysis.

- Create a new Collective Variable object, modify some of his attributes and generate the plumed inputs:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule("MOL")
# Create the RDF object for the radial distribution func
rdf = gaff.new_cv("rdf-center", "rdf")
# Use the geometrical center instead of center of mass
rdf.center = "geometrical"
# Use all atoms to calculate the molecule geometric center
rdf.set_atoms("all", molecule)
# Retrieve a completed simulation
nvt = gaff.get_simulation("npt")
# Generate plumed driver input for the selected simulation
rdf.generate_input(npt)
# Save project to be used later
project.save()
```

- Retrieve an existing CV and check if plumed drive analysis is completed for all crystals:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve the RDF object
rdf = gaff.get_cv("rdf-center")
# Retrieve a completed simulation
nvt = gaff.get_simulation("nvt")
# Check and import resulting distributions
rdf.get_results(nvt)
# Save project to be used later
project.save()
```

- Generate inputs for the RDF of molecules center of mass excluding hydrogen atoms:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve the CV Object
rdf = gaff.get_cv("rdf")
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule("MOL")
# Select the atoms in the molecule
rdf.set_atoms("non-hydrogen", molecule)
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generate plumed driver input for the selected simulation
rdf.generate_input(npt)
# Save project
project.save()
```

- Import distributions once the plumed driver analysis is finished:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule("MOL")
# Retrieve the CV Object
rdf = gaff.get_cv("rdf")
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generate plumed driver input for the selected simulation
rdf.get_results(npt, plot=False)
# Save project
project.save()
```

2.3.3 Distributions: Torsions

With Torsions object, plumed inputs to calculate the distribution of torsional angles can be created. The resulting distribution can be used as part of the crystal fingerprint in the clustering analysis.

- Select atoms of the torsional angles and create plumed inputs:

```python
from PyPol import pypol as pp
```

```python
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule("MOL")
# Retrieve the CV Object
tor = gaff.get_cv("tor")
# Use the first four atoms to define the torsional angle
tor.set_atoms((0, 1, 2, 3), molecule)
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generate plumed driver input for the selected simulation
tor.generate_input(npt)
# Save project
project.save()
```

- Import distributions once the plumed driver analysis is finished:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule("MOL")
# Retrieve the CV Object
tor = gaff.get_cv("tor")
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generate plumed driver input for the selected simulation
tor.get_results(npt, plot=False)
# Save project
project.save()
```

2.3.4 Combine 1D distributions in ND distribution

Two or more 1D torsional or intermolecular angles can be combined together to create multidimentsional distributions.

- Create two Torsions CV and combine them in 2D distributions:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
```

```python
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Use molecular forcefield info for the CV
molecule = gaff.get_molecule(0)
# Create a new Torsions object
tor1 = gaff.new_cv("tor1", "tor")
# Define the atoms for the torsional angle
tor1.set_atoms((7, 12, 3, 6), molecule)
# Create a new Torsions object
tor2 = gaff.new_cv("tor2", "tor")
# Define the atoms for the torsional angle
tor2.set_atoms((4, 8, 16, 5), molecule)
# Combine two CV
tor = gaff.combine_cv("2d-tor", (tor1, tor2))
# Retrieve a completed simulation
nvt = gaff.get_simulation("nvt")
# Generate plumed driver input for the selected simulation
tor.generate_input(nvt)
# Save project to be used later
project.save()
```

2.3.5 Groups

Sometimes it is useful to divide the set of structures in subset based on distribution similarities or specific attributes of the crystal. One good example is to calculate the distribution of relevant torsional angles in a molecule and then divide the set of structures based on the different conformation of the molecule. This can be done by comparing all pairs of distributions and cluster together the similar ones. Alternatively, you can select a set of boundaries in the CV space and classify the structures based on which areas of the space are non-zero. If a group object is included in the clustering analysis, structures are firstly divided into the calculated groups. Then, fingerprint comparison is done only between members of the same group.

- Retrieve an existing CV and create groups of crystals from it:

```python
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Retrieve the CV Object
tor = gaff.get_cv("tor")
# Import plumed output and check normal termination
```

```
tor.get_results(npt)
# Create the GGFD object
conf = gaff.ggfd("conf", tor)
# Use the similarity grouping method
conf.grouping_method = "similarity"
# Use the simps method to calculate the Hellinger distance
conf.integration_type = "simps"
# Generates groups
conf.run(tor)
project.save()
```

- Group structures by similarity:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve the CV Object
tor = gaff.get_cv("tor")
# Create the GGFD object
conf = gaff.ggfd("conf", tor)
# Use the similarity grouping method
conf.grouping_method = "similarity"
# Use the simps method to calculate the hellinger distance
conf.integration_type = "simps"
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generates groups
conf.run(tor)
project.save()
```

- Group structures using boundaries:

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrieve the CV Object (2D torsional angle)
tor = gaff.get_cv("tor")
# Create the GGFD object
conf = gaff.ggfd("conf", tor)
# Use the "groups" grouping method
conf.grouping_method = "groups"
# Cutoff for determining which areas are occupied
```

```python
conf.group_threshold = 0.1
# Define the group boundaries in the (2D) CV space
conf.set_group_bins((-2., 0, 2.),(-1., 2.), periodic=True)
# Retrieve a completed simulation
npt = gaff.get_simulation("npt")
# Generates groups
conf.run(tor)
project.save()
```

2.3.6 Molecular and crystal attributes

User can add custom made features to Crystal and Molecule objects using the Crystal.set_attribute and Molecule.set_attribute. In the following example, the chirality of ibuprofen molecules is identified and saved as molecular attribute, using the RDKit and openbabel python packages. Then, enantiopure and racemic structures are distinguished using this information.

```python

from PyPol import pypol as pp
from rdkit import Chem
from openbabel import openbabel
import os

project = pp.load_project(r'/home/nicolas/Work/Ibuprofen/project_tmp/')

for crystal in project.crystals:
    ChiralDict[crystal._name] = []
    molecules = crystal.molecules
    # Identify the R or S configuration of each molecule and assign the molecule attribute
    for molecule in molecules:
        molecule._save_gro(crystal.path + molecule.residue + ".gro")

        ob_conversion = openbabel.OBConversion()
        ob_conversion.SetInAndOutFormats("gro", "mol")
        mol = openbabel.OBMol()
        ob_conversion.ReadFile(mol, crystal.path + molecule.residue + ".gro")
        ob_conversion.WriteFile(mol, crystal.path + molecule.residue + ".mol")

        mol1 = Chem.MolFromMolFile(crystal.path + molecule.residue + ".mol")
        Chem.AssignAtomChiralTagsFromStructure(mol1)
        c = Chem.FindMolChiralCenters(mol1)[0][1]
        ChiralDict[crystal._name].append(c)
        molecule.set_attribute("Chirality", c)
        os.remove(crystal.path + molecule.residue + ".gro")
    crystal.update_molecules(molecules)
    # Assign a crystal attribute to each crystal depending if it is enantioure or racemic
    if all(k == ChiralDict[crystal._name][0] for k in ChiralDict[crystal._name]):
        crystal.set_attribute("Chirality", "Eniantiopure")
    else:
        crystal.set_attribute("Chirality", "Racemic")

project.save()
```

The crystal attribute can be then used by the GGFA (Generate Groups From Attributes) object to classify the crystals.

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Create a group object from the Chirality attribute previously determined
chi = gaff.ggfa(name="Chirality", attribute="Chirality")
```

2.3.7 Clustering analysis

After having calculated the different distributions and groups, you can perform a clustering analysis using the crystal fingerprints to identify those structures that convert to the same geometry. The resulting cluster can be seen in the Output directory together with the Distance matrix and the FSFDP outputs.

```
from PyPol import pypol as pp
# Load project from the specified folder
project = pp.load_project(r'/home/Work/Project/')
# Retrieve an existing method
gaff = project.get_method('GAFF')
# Retrive the different distributions and groups objects
rdf = mdvv.get_cv("RDF")
mo = mdvv.get_cv("mo")
chi = mdvv.get_cv("Chirality")
tor = mdvv.get_cv("tor")
conf = mdvv.get_cv("Conformation")
# Create a clustring object using the distributions and group objects needed
cm = mdvv.new_clustering_parameters("Cluster1", (chi, conf, rdf, mo, tor))
# Modify the clustering algorithm parameters
cm.d_c_neighbors_fraction = 0.01
cm.sigma_cutoff = 0.15
# Run the clustering analysis
cm.run(meta)
```