FORCE Gromacs

Release 0.2.0.dev0

unknown

Nov 11, 2020

CONTENTS

CHAPTER

ONE

FORCE BDSS GROMACS PLUGIN

This repository contains the implementation of a plugin for the Business Decision Support System (BDSS), contributing the GROMACS Molecular Dynamics package. It is implemented under the Formulations and Computational Engineering (FORCE) project within Horizon 2020 (NMBP-23-2016/721027).

The `GromacsPlugin` class contributes several BDSS objects, including `DataSource` and `NotificationListener` subclasses, as well as a stand-alone wrapper around Gromacs version 2019.4.

CHAPTER

TWO

INSTALLATION

Installation requirements include an up-to-date version of `force-bdss`. Additional modules that can contribute to the `force-wfmanager` UI are also included, but a local version of `force-wfmanager` is not required in order to complete the installation.

To install `force-bdss` and the `force-wfmanager`, please see the following instructions.

After completing at least the `force-bdss` installation steps, clone the git repository:

```
git clone https://github.com/force-h2020/force-bdss-plugin-gromacs
```

the enter the source directory and run:

```
python -m ci install
```

This will allow install the plugin in the `force-py36` edm environment, allowing the contributed BDSS objects to be visible by both `force-bdss` and `force-wfmanager` applications.

CHAPTER

THREE

DOCUMENTATION

Full documentation is being hosted at the FORCE GROMACS ReadTheDocs page.

Alternatively, to build the documentation locally in the `doc/build` directory, run:

```
python -m ci docs
```

CHAPTER

FOUR

USER MANUAL

4.1 Introduction

GROMACS is a Molecular Dynamics (MD) package, primarily designed to simulate biomolecules, such as proteins, lipids and nucleic acids.

A cross-platform distribution is available as an egg from EDM, currently supporting version 2019.4

4.1.1 Chemicals Module Design

The Force Gromacs wrapper is built on top of a small library defining a hierarchical collection of generic chemical species, `force_gromacs.chemicals`. It is build for extensibility, providing base traits classes as well as the interfaces for these objects that could be fulfilled by objects in an external package.

The lowest object in this hierarchy is the `IParticle`, which defines a very simple interface for a class that possesses both `mass` and `charge` attributes. A particle is not therefore fixed to a specific length scale since a proton, atom, molecule or large molecular complex species would all be able to fulfil this interface. An example of an object that fulfils this interface is the `GromacsParticle` class.

```
>>> my_particle = GromacsParticle(mass=12, charge=0)
>>> my_particle.mass
12.0
>>> my_particle.charge
0.0
```

Next up from this is the `IParticleGroup` class, representing a collection of particles. The interface inherits from `IParticle`, so that a group of particles can also be represented by a reduced representation given by a single particle.

In the atomistic length scale, we also introduce the `IFragment` class to describe molecular fragments. A fragment represents a part of a molecule containing a single or collection of covalently bonded particles. Therefore it inherits from `IParticleGroup`, but also contains the attribute `stoichiometry`, the stoichiometric number of each fragment in the molecule. An example of an object that fulfils this interface is the `GromacsFragment` class, which also contains (optional) information regarding the geometry of molecular fragments.

```
>>> my_fragment = GromacsFragment(particles=[my_particle])
>>> my_fragment.mass
12.0
>>> my_fragment.charge
0.0
>>> my_fragment.stoichiometry
1
```

A molecule of sodium cabonate Na_2CO_3 consists of 3 fragments: two Na^+ atomic ions and the CO_3^{2-} molecular ion:

$$Na_2CO_3 \leftrightarrow 2Na^+ + CO_3^{2-}$$

All ionic species are free to dissociate, and therefore do not possess any constraints in a MD simulation regarding their equations of motion. We can also freely add and take away integer numbers of these objects in a simulation cell and calculate their molecular concentrations in a mixture. However, in reality we cannot 'add' fragments from a jar on the laboratory shelf, and instead describe formulations by their constituent molecules (typically in concentration % by mass).

Therefore, the `Molecule` class is designed to represent a full computational model for a chemical found in the laboratory. It contains a list of `IFragment` classes, and must be overall electronically neutral. We can describe the calcium carbonate molecule by the following `force_gromacs` objects:

Firstly the constituent atomic particles:

```
>>> sodium = GromacsParticle(element='Na', mass=11, charge=1)
>>> carbon = GromacsParticle(element='C', mass=12, charge=4)
>>> oxygen = GromacsParticle(element='O', mass=16, charge=-2)
```

Next the fragment ions:

```
>>> sodium_ions = GromacsFragment(particles=[sodium], stoichiometry=2)
>>> sodium_ions.mass
22.0
>>> sodium_ions.charge
2.0

>>> carbonate_ion = GromacsFragment(particles=[carbon] + 3 * [oxygen])
>>> carbonate_ion.mass
60.0
>>> carbonate_ion.charge
-2.0
```

And finally the full molecule:

```
>>> sodium_carbonate = Molecule(fragments=[sodium_ions, carbonate_ion])
>>> sodium_carbonate.mass
82.0
>>> sodium_carbonate.charge
0.0
>>> sodium_carbonate.neutral
True
```

4.2 BDSS Plugin Objects

The FORCE Gromacs plugin also contributes serveral BDSS objects that can be used either out of the box or provide a base class for further customization.

4.2.1 Data Sources

FragmentDataSource

The `FragmentDataSource` class provides an interface for a BDSS user to create a `GromacsFragment` that can be propagated through a Workflow as a `DataValue`. The data source takes no input parameters, and produces a single output parameter, which is the `GromacsFragment` instance being constructed. Therefore it can be considered as a factory of `GromacsFragment` objects.

The following information must be provided in the model interface:

- **Name**: A human readable name of the fragment.
- **Symbol**: The symbol that is used to idenfity the fragment in GROMACS input files
- **Coordinate**: A path to the GROMACS coordinate `.gro` file that described the fragment's geometry
- **Topology**: A path to the GROMACS topology `.itp` file that described the fragment's force field parameters

MoleculeDataSource

The `MoleculeDataSource` class provides an interface for a BDSS user to create a `GromacsMolecule` that can be propagated through a Workflow as a `DataValue`. The data source takes one or more `GromacsFragment` input parameters, and produces a single output parameter, which is the `GromacsMolecule` instance being constructed. Therefore it can be considered as a factory of `GromacsMolecule` objects.

The following information must be provided in the model interface:

- **No. Fragment Types**: The number of different fragment types in the molecule
- **Fragment Numbers**: The stoichiometric number of each fragment type in the molecule

SimulationDataSource

The `SimulationDataSource` base class provides an interface for a BDSS user to create and run a `BaseGromacsSimulationBuilder` instance that can construct and perform a `GromacsPipeline` object. The data source takes one or more `GromacsMolecule` input parameters, and produces a single output parameter, which is the file path to the GROMACS trajectory file that is expected to be post-processed by further data sources.

The data source requires developers to implement the `create_simulation_builder` method, which produces a `BaseGromacsSimulationBuilder` instance.

```
def create_simulation_builder(self, model, parameters):
    """Method that returns a `GromacsSimulationBuilder` object capable
    of generating a `GromacsPipeline`

    Parameters
    ----------
    model: SimulationDataSourceModel
        The BaseDataSourceModel associated with this class
    parameters: List(DataValue)
        a list of DataValue objects containing the information needed
        for the execution of the DataSource.

    Returns
    -------
    simulation_builder: BaseGromacsSimulationBuilder
```

(continues on next page)

(continued from previous page)

```
        An object capable of generating a GromacsPipeline that calls
        a Gromacs simulation
    """
```

It also emits a `SimulationProgressEvent` object containing the bash script for each GROMACS command that is called during the simulation.

The following information must be provided in the model interface:

- **Name**: A human readable name of the simulation.
- **Output Directory**: The local directory path that will be used to contain simulation output and input files
- **No. Molecule Types**: The number of different molecule types in the simulation cell
- **Size**: The total number of fragments in the simulation
- **No. Steps**: The length of the simulation in time steps
- **MARTINI Parameter**: A path to the GROMACS topology `.itp` file that contains the MARTINI forcefield
- **Minimization Parameter File**: File path to the GROMACS parameter `.top` file that contains the instructions for an energy minimization run
- **Production Parameter File**: File path to the GROMACS parameter `.top` file that contains the instructions for a production run
- **Overwrite Simulation Data?**: Whether or not to overwrite any existing simulation data files.
- **Dry Run?**: Whether or not to perform a dry run of the simulation.
- **MPI Run?**: Whether or not to perform an MPI run of the simulation using parallel processing
- **No. Processes**: Number of processes to use in an MPI run
- **Overwrite Data?**: Whether or not to overwrite any existing simulation data files.

4.2.2 Notification Listeners

HPCWriter

The `HPCWriter` class reacts to a `SimulationProgressEvent` in order to output bash file that can be used as a submission script to a HPC. The file contains the series of GROMACS commands that is communicated by the `SimulationProgressEvent` as well as a customizable header and prefix. The output file name is determined by the name of the simulation contained in the event's bash script.

The following information must be provided in the model interface:

- **Header**: The header of the output bash file containing HPC submission script details
- **Prefix**: An extended multi-line section of the output bash file containing any additioanl submission script details
- **Dry Run?**: Whether or not to perform a dry run of writing the file.

CHAPTER

FIVE

API REFERENCE