biobb$_{w}f_{c}ommand - lineDocumentation$

Release 1.0.0

Bioexcel Project

Feb 18, 2021

Contents

CHAPTER 1

Contents

1.1 Command-line workflows with BioExcel Building Blocks

This tutorial aims to illustrate the process of building up a command-line workflow using the BioExcel Building Blocks library (biobb). The tutorial is based on the Protein Gromacs MD Setup Jupyter Notebook tutorial.

1.1.1 Settings

Biobb modules used

- biobb_io: Tools to fetch biomolecular data from public databases.
- biobb_model: Tools to model macromolecular structures.
- biobb_md: Tools to setup and run Molecular Dynamics simulations.
- biobb_analysis: Tools to analyse Molecular Dynamics trajectories.

Software requirements:

- Python3
- Anaconda

1.1.2 Tutorial

Click here to view tutorial in Read the Docs

1.1.3 Version

May 2020 Release

1.1.4 Copyright & Licensing

1.2 Command-line workflows with BioExcel Building Blocks

1.2.1 Based on the Protein MD Setup tutorial using BioExcel Building Blocks (biobb)

This tutorial aims to illustrate the process of **building up a command-line workflow** using the **BioExcel Building Blocks library (biobb)**. The tutorial is based on the **Protein Gromacs MD Setup** Jupyter Notebook tutorial.

Biobb modules used:

- biobb_io: Tools to fetch biomolecular data from public databases.
- biobb_model: Tools to model macromolecular structures.

- biobb_md: Tools to setup and run Molecular Dynamics simulations.
- biobb_analysis: Tools to analyse Molecular Dynamics trajectories.

Software requirements:

- Python3
- Anaconda

1.2.2 Tutorial Sections:

1. *Introduction*
2. *Why Command Line*
3. *Essential points / Before Starting*
    1. *Workflow General Concepts*
    2. *Configuration Input File General Concepts*
4. *First Example*
5. *Protein MD-Setup workflow*
6. *Questions & Comments*

1.2.3 Introduction

Biomolecular workflows built using **BioExcel building blocks** (biobb) can be launched in **command line** (without Jupyter or iPython interactivity) with a combination of a **Python script** and a separated **yaml formatted file** containing the **input parameters**. This approximation combines the power of the **scripting in Python** with the **interoperability** of **biomolecular simulation workflows** provided by the **BioExcel building blocks**, and is the one used in **production** runs.

The example used to illustrate the process of building a **command line biomolecular simulation workflow** with the **BioExcel building blocks** will be the **Protein MD Setup** presented in the Jupyter Notebook tutorial. In the **first steps**

section, the **main points** to take into account when building a **command line workflow** will be introduced, using as example the first 2 steps of the **Protein MD Setup**. In the **Protein MD Setup command-line workflow** section, the complete workflow presented in the Jupyter Notebook tutorial will be translated to a **Python script + yaml-formatted file** to launch it in a **command-line interface**.

1.2.4 IMPORTANT NOTE

Please note that this Jupyter Notebook is **NOT executable**, it is just used to **illustrate the process**. Cells containing source code are used to **show information** in a graphical way, but they are **NOT designed to be executed**.

1.2.5 Why command line

Jupyter Notebooks are fantastic tools to explore and play with software, and in particular, with the **BioExcel building blocks** library. But when a workflow is ready to start with **production**, or it needs to be expanded with more complex **loop/conditional** structures, the best option is to move from a **GUI** to a **command line** execution. In **command line** is where the **real power** of the **BioExcel building blocks** unleashes.

Going from a **Jupyter Notebook** to a **command line python script** is as easy as **exporting** the notebook to a **Python script** (Menu –> File –> Download as –> Python (.py)). The **downloaded Python script** will be executable just typing *python name_of_the_notebook.py* (in our case, *python biobb_MDsetup_tutorial.py*). Remember that for the script to properly work, the library modules should have been **previously installed** in your **conda environment**, following the **Protein MD tutorial** installation steps (*Conda Installation and Launch*).

Moving from the **Jupyter environment** to a **command line**, we are going to miss the **interactivity** and the possibility to use **graphical support** such as 3D structure viewers or plot representation. But it will offer us a jump to the **High Throughput** (HT) regime. Many different instances of the workflow can be launched at the same time with different inputs. Command line executions are offering us **automation and repetition**. However, when a particular **input file** or **step parameter** needs to be changed, the main **Python script** should be changed. That's why the **BioExcel building blocks workflows** are divided in **two files**:

- **Workflow Python code**: Main code of the workflow, with the different steps and flowchart.
- **Configuration file** (YAML): Input files (paths) and parameters for all the different steps of the workflow.

With this, there's no need for the **main Python code** to be modified for every different execution. Just changing the **input configuration file** is enough. The next steps of the tutorial are introducing the **basic concepts** of both **workflow** and **configuration input** files.

1.2.6 Essential points / Before Starting

Before starting with the **tutorial**, a set of **important terms** that will appear during the building of **BioExcel building blocks workflows** need to be introduced. These **terms** are divided in the **workflow** ones, applied in the **Python script** describing the **workflow**, and the **configuration input file** ones, applied in the separated **YAML configuration file**.

Workflow general concepts (Python):

BioExcel building blocks workflows Python scripts always start with the **initialization of 3 main variables**:

- **Configuration info** (conf), loaded from the input configuration file.
- **Properties and parameters** (global_prop, global_paths) for all the different steps of the workflow, parsed from the configuration info.
- **Configuration Info**: Loading the configuration information from the input configuration YAML file. It is done using the *ConfReader* tool from the biobb_common configuration library:

conf = settings.ConfReader(config), where **config** is:

```
- config (str): Path to the configuration YAML file.
```

Example:

> **conf = settings.ConfReader("input_config_file")**

Configuration information read from the file "input_config_file".

- **Global properties**: Collection of properties (tool parameters) for every step of the workflow extracted from the **Configuration Info**. It is generated using the *get_prop_dic* tool from the biobb_common configuration library:

global_prop = conf.get_prop_dic(params), where **params** could be:

```
- prefix (str): Prefix if provided. Prefix is used to add levels of hierarchy in the␣
↪workflow structure.
- global_log (Logger object): Log from the main workflow.
```

Example:

> **global_prop = conf.get_prop_dic(global_log=global_log)** *global_prop variable contains all the properties for all the workflow steps, parsed from the input configuration file. As the global_log is passed as input, local logs from all the steps will be appended to the global log.

- **Global paths**: Collection of paths (tool inputs & outputs) for every step of the workflow extracted from the **Configuration Info**. It is generated using the *get_paths_dic* tool from the biobb_common configuration library:

global_paths = conf.get_paths_dic(params), where **params** could be:

```
- prefix (str): Prefix if provided. Prefix is used to add levels of hierarchy in the␣
↪workflow structure.
```

Example:

> **global_paths = conf.get_paths_dic()**

Optionally, a **Global Log** could be initialized, capturing **log information** for the **whole workflow**.

- **Global log files** (output log and error log).
- **Global Log**: Name of the file that will contain the **global log of the workflow** execution. Usually **created only once** at the beginning of the workflow. It is generated using the *get_logs* tool from the biobb_common file_utils library:

global_log_out, global_log_err = file_utils.get_logs(params), where **params** could be:

```
- path (str): Path to the log file directory. By default assigned to the current␣
↪working directory.
- prefix (str): Prefix added to the name of the log file. By default log.out and log.
↪err.
```

(continues on next page)

(continued from previous page)

```
- step (str): String added between the prefix and the name of the log file.
- can_write_console (bool): If True, show log in the execution terminal. False by␣
↪default.
- level (str): Set Logging level. ['CRITICAL','ERROR','WARNING','INFO','DEBUG',
↪'NOTSET']. INFO by default.
- light_format (bool): Minimalist log format. False by default.
```

Example:

log_out,log_err = file_utils.get_logs(path='/home/biobb/wf_example/', prefix='tutorial', level='WARNING')

Log file directory written to the '/home/biobb/wf_example/' working dir path (path), with level of logging set to 'WARNING' (level). The names of the log files will be tutorial_log.out and tutorial_log.err (prefix).

The next cell shows an example of the typically first lines of a **BioExcel building blocks** workflow, initializing the **Configuration Info** (conf), the **tools parameters** (global_prop), the **tools inputs & outputs** (global_paths) and the **global log** (global_log):

```
# Workflow concepts: Configuration Info, global paths & global properties
from biobb_common.configuration import settings

conf = settings.ConfReader(config_yaml_file)
global_prop = conf.get_prop_dic()
global_paths = conf.get_paths_dic()

# Workflow concepts: Global Log
from biobb_common.tools import file_utils

global_log, _ = file_utils.get_logs(path="/home/biobb/wf_example", level='WARNING')
```

Configuration input file concepts (YAML):

The **YAML configuration file** that contains all the **workflow input parameters** has a well-defined structure, divided in **2 main sections**: the **Global Workflow Properties**, which are properties applied to the whole workflow, and the different **Workflow Steps**, with properties for all the different steps of the workflow, one by one.

1. **Global Workflow Properties**: Define global workflow properties and are typically stated at the beginning of the YAML file. Available global properties are:
   - *working_dir_path*: Workflow output directory, where all results are going to be written.
   - *can_write_console_log*: Enable the automatic output of the information log to the console.
   - *remove_tmp*: Remove temporary files after execution.
   - *restart*: Skip already computed steps. Automatically detects already computed steps checking existence of output files. Useful in long executions that didn't reach the end for whatever reason.

   Example of **Global Workflow Properties**:

```
# YAML configuration input file concepts: Global Workflow Properties

working_dir_path: md_tutorial      # Folder to write i/o files of the workflow␣
↪steps
can_write_console_log: False       # Verbose writing of log information
remove_tmp: True                   # Remove temporary files after execution
restart: True                      # Skip steps already performed
```

2. **Workflow Steps**: Independent **properties** applied to **each single step** of the workflow, defined **one by one**. Usually, there should be at least as many steps definitions as steps defined in the **Python workflow**. The name of the step is used in the Python workflow script to **identify input/output files and properties** for the corresponding workflow step. **Workflow Steps** definitions are divided in **two fields**:

   - **Paths**: **Inputs** and **Outputs** of the **building blocks**. Defined as **file names** with relative or absolute paths, or as **dependencies** from output files coming from **previous steps of the workflow**. If a step needs (as input) an output from a previous step, a dependency should be specified. Dependencies syntax is defined as: *dependency/previous_step_name/previous_step_output_name* (see example below).
   - **Properties**: **Configuration parameter** (properties) of the **building blocks**. Specific input parameters for the particular building block, always defined inside a *properties* field.

   Example of **Workflow Steps**:

```
# Configuration input file concepts: Workflow Steps

# Step 1: Downloading PDB 1AKI and saving it to the "structure.pdb" file
step1_pdb:
  paths:
    output_pdb_path: structure.pdb
  properties:
    pdb_code: 1aki

# Step 2: Fixing side chains of the structure and saving it to the "fixsidechain.
↪pdb" file
# Input structure (input_pdb_path) is defined as a dependency from the
# output_pdb_path of the previous step1_pdb step (dependency/step1_pdb/output_pdb_
↪path)
step2_fixsidechain:
  paths:
    input_pdb_path: dependency/step1_pdb/output_pdb_path
    output_pdb_path: fixsidechain.pdb
```

Paths and properties for each of the **building blocks** available in the **biobb library** can be found in the corresponding **module documentation pages**. Please visit https://mmb.irbbarcelona.org/biobb/availability/source to quickly find links to the desired **module documentation**.

Each **building block** has its own set of specific properties (parameters). However, there is a list of properties that are **common** to most of the **building blocks**. Those are the **Common Step Properties** that can be added to any of the **BioExcel building blocks**:

- *can_write_console_log*: Overwrite can_write_console_log workflow property on this step.
- *remove_tmp*: Overwrite remove_tmp workflow property on this step.
- *restart*: Overwrite restart workflow property on this step.

There is also a list of **common properties** for all the **container-compatible building blocks** (those able to wrap tools embedded in **Docker** or **Singularity** containers). The list of these **common properties** are:

- *container_path*: Path to the binary executable of your container.
- *container_image*: Container Image identifier.
- *container_volume_path*: Path to an internal directory in the container.
- *container_working_dir*: Path to the internal CWD in the container.
- *container_user_id*: User number id to be mapped inside the container.
- *container_shell_path*: Path to the binary executable of the container shell.

Example of **Workflow Steps** with **container Common Properties**, taken from the Mutation free energy calculations tutorial:

```
# Configuration input file concepts: Common Step Properties

# Step 1: Modelling mutated structure with the desired new residue (Isoleucine 10 to
↪Alanine) using pmx package.
step1_pmx_mutate:
  paths:
    output_structure_path: mut.gro
  properties:
    mutation_list : Ile10Ala
    force_field: amber99sb-star-ildn-mut
    container_path: singularity
    container_image: /home/group/user/singularities/pmx_standalone.sif
```

1.2.7 First Example

The first two steps of the **Protein MD Setup** workflow are responsible for downloading a **protein structure** from the **PDB database**, and **fixing the structure**, adding any **missing side chain atoms**. The building blocks used for this are the Pdb building block, from the biobb_io package, including tools to **fetch biomolecular data from public databases**, and the FixSideChain building block, from the biobb_model package, including **tools to check and model 3d structures**, **create mutations** or **reconstruct missing atoms**.

The first 2 steps of the **Protein MD Setup** workflow Jupyter Notebook tutorial look like this:

```
# Extract from the Protein MD Setup workflow Jupyter Notebook tutorial
# https://github.com/bioexcel/biobb_wf_md_setup

#### Input Vars ####

pdbCode = "1AKI"

#### Step 1 ####

# Downloading desired PDB file
# Import module
from biobb_io.api.pdb import Pdb

# Create properties dict and inputs/outputs
downloaded_pdb = pdbCode+'.pdb'
prop = {
    'pdb_code': pdbCode
}

# Create and launch bb
Pdb(output_pdb_path=downloaded_pdb,
    properties=prop).launch()

#### Step 2 ####

# Check & Fix PDB
# Import module
from biobb_model.model.fix_side_chain import FixSideChain

```

(continues on next page)

(continued from previous page)

```
# Create prop dict and inputs/outputs
fixed_pdb = pdbCode + '_fixed.pdb'

# Create and launch bb
FixSideChain(input_pdb_path=downloaded_pdb,
             output_pdb_path=fixed_pdb).launch()
```

Converting these 2 steps into a **command-line workflow** requires to split the code in two:

- The **workflow** (**Python script**)
- The **input parameters** (**YAML file**)

Python Script

The **Python script** is build taking just the calls to the **BioExcel building blocks**.

Step by step, first we need to import all the required modules:

- **sys**, to be able to retrieve the **input parameters** of the **Python script** (argv).
- **settings**, from **biobb_common** module, to be able to retrieve the **workflow input parameters** from the separated **YAML file**.
- **Pdb**, from **biobb_io** module, to retrieve a **PDB structure** from the RCSB PDB database.
- **FixSideChain**, from **biobb_model**, to **add any missing atom** in the side chains of the protein.

```
#!/usr/bin/env python3

import sys
from biobb_common.configuration import settings
from biobb_io.api.pdb import Pdb
from biobb_model.model.fix_side_chain import FixSideChain
```

Then, retrieve the **YAML configuration** file from the **command-line arguments** passed to the **Python script** (argv), and load them using the *settings* **ConfReader** function. This function is able to split the input data included in the **configuration file** in two: **paths** (inputs/outputs) and **properties** (parameters) for each of the defined **steps of the workflow**. This is done using the corresponding class functions **get_paths_dic** and **get_prop_dic**.

```
#### Input Vars ####

# Loading the biobb configuration reader
conf = settings.ConfReader(sys.argv[1])

# Reading the inputs and properties from the separated YAML file
conf_properties = conf.get_prop_dic()
conf_inputs = conf.get_paths_dic()
```

Finally, we need to call the **building blocks** using its common class function **launch**. As input parameters all the **building blocks** are prepared to receive the **paths** (inputs and outputs) and the **properties** (input parameters). As we already have previously read all this data and stored it in the **conf_inputs** and **conf_properties** variables, we can now use them to extract the corresponding **step inputs**, using the **name (id)** of each **step** as a **key**. To be compatible with the format accepted by the **building blocks**, the **input paths** should be **expanded** (using the Python **prefix operator to unpack dictionaries**). Thus, the **paths** for the **step1** are passed as an argument using its expanded version: **conf_inputs["step1_pdb"]**. Oppositely, the **properties** of each step are passed directly: **properties=conf_properties["step1_pdb"]**. This syntax is repeated for **all of the steps** in the workflow.

```
#### Step 1 ####
Pdb(**conf_inputs["step1_pdb"], properties=conf_properties["step1_pdb"]).launch()

#### Step 2 ####
FixSideChain(**conf_inputs["step2_fixsidechain"], properties=conf_properties["step2_
↪fixsidechain"]).launch()
```

The information placed in the **paths** and **properties** terms of the **YAML separated files** is explained in the next section.

The **complete example** of the **Python script** for the first two steps of the **Protein MD Setup** workflow looks like this:

```
#!/usr/bin/env python3

import sys
from biobb_common.configuration import settings
from biobb_io.api.pdb import Pdb
from biobb_model.model.fix_side_chain import FixSideChain

#### Input Vars ####

# Loading the biobb configuration reader
conf = settings.ConfReader(sys.argv[1])

# Reading the inputs and properties from the separated YAML file
conf_properties = conf.get_prop_dic()
conf_inputs = conf.get_paths_dic()

#### Step 1 ####
Pdb(**conf_inputs["step1_pdb"], properties=conf_properties["step1_pdb"]).launch()

#### Step 2 ####
FixSideChain(**conf_inputs["step2_fixsidechain"], properties=conf_properties["step2_
↪fixsidechain"]).launch()
```

YAML configuration file

The **YAML configuration file** containing the **input parameters** for this example includes the **paths** and **properties** of the two first steps of the **Protein MD Setup** workflow:

```
# Step 1: Downloading PDB 1AKI and saving it to the "structure.pdb" file
step1_pdb:
  paths:
    output_pdb_path: structure.pdb
  properties:
    pdb_code: 1aki

# Step 2: Fixing side chains of the structure and saving it to the "fixsidechain.pdb"␣
↪file
# Input structure (input_pdb_path) is defined as a dependency from the
# output_pdb_path of the previous step1_pdb step (dependency/step1_pdb/output_pdb_
↪path)
step2_fixsidechain:
  paths:
    input_pdb_path: dependency/step1_pdb/output_pdb_path
    output_pdb_path: fixsidechain.pdb
```

Run example workflow

The **final step** of the process is running the **command-line workflow**. For that, the **Python script** and the **YAML configuration file** presented in the previous cells should be written to disk (e.g. biobb_MDsetup_tutorial-lite.py and biobb_MDsetup_tutorial-lite.yaml), and finally both files should be used to run the workflow.

It is important to note that in order to properly run the workflow, all the **BioExcel building blocks modules** used should be **previously installed** following the **Protein MD tutorial** installation steps (*Conda Installation and Launch*).

The **command line** is shown in the cell below:

```
python biobb_MDsetup_tutorial-lite.py biobb_MDsetup_tutorial-lite.yaml
```

Workflow output

The **execution of the workflow** will write information to the **standard output** such as the **tools** being executed, the **command lines**, **inputs and outputs** used, and **state** of each step (exit codes). The next cell contains a **real output** for the execution of our first example:

```
2020-05-20 18:32:14,229  [INFO ]  Downloading: 1aki from: https://files.rcsb.org/
↪download/1aki.pdb
2020-05-20 18:32:14,229  [INFO ]      Downloading: 1aki from: https://files.rcsb.org/
↪download/1aki.pdb
2020-05-20 18:32:15,192  [INFO ]  Writting pdb to: /Users/biobb_tutorials/VT/cli/Yaml/
↪md_tutorial-lite/step1_pdb/structure.pdb
2020-05-20 18:32:15,192  [INFO ]      Writting pdb to: /Users/biobb_tutorials/VT/cli/
↪Yaml/md_tutorial-lite/step1_pdb/structure.pdb
2020-05-20 18:32:15,192  [INFO ]  Filtering lines NOT starting with one of these␣
↪words: ['ATOM', 'MODEL', 'ENDMDL']
2020-05-20 18:32:15,192  [INFO ]      Filtering lines NOT starting with one of these␣
↪words: ['ATOM', 'MODEL', 'ENDMDL']

2020-05-20 18:32:15,750  [INFO ]  check_structure -i /Users/biobb_tutorials/VT/cli/
↪Yaml/md_tutorial-lite/step1_pdb/structure.pdb
        -o /Users/biobb_tutorials/VT/cli/Yaml/md_tutorial-lite/step2_fixsidechain/
↪fixsidechain.pdb --force_save fixside --fix ALL

2020-05-20 18:32:15,750  [INFO ]  Exit code 0

2020-05-20 18:32:15,750  [INFO ]
================================================================================
=                     MDWeb structure checking utility                         =
=                   A. Hospital, P. Andrio, J.L. Gelpi 2018                    =
================================================================================


Structure /Users/biobb_tutorials/VT/cli/Yaml/md_tutorial-lite/step1_pdb/structure.pdb␣
↪loaded
 Title:
 Experimental method: unknown
 Resolution: 0.0 A

 Num. models: 1
 Num. chains: 1 (A: Protein)
 Num. residues:  129
 Num. residues with ins. codes:  0
 Num. HETATM residues:  0
```

(continues on next page)

(continued from previous page)

```
 Num. ligands or modified residues:  0
 Num. water mol.:  0
 Num. atoms:  1001

Running fixside. Options: --fix ALL
No residues with missing side chain atoms found
Structure not modified, saving due to --force_save option
Final Num. models: 1
Final Num. chains: 1 (A: Protein)
Final Num. residues:  129
Final Num. residues with ins. codes:  0
Final Num. HETATM residues:  0
Final Num. ligands or modified residues:  0
Final Num. water mol.:  0
Final Num. atoms:  1001
Structure saved on /Users/biobb_tutorials/VT/cli/Yaml/md_tutorial-lite/step2_
↪fixsidechain/fixsidechain.pdb

2020-05-20 18:32:15,751  [INFO ]      Executing: check_structure -i /Users/biobb_
↪tutorials/VT/cli/Yaml/md_tutorial-lite/...
2020-05-20 18:32:15,751  [INFO ]      Exit code 0
```

1.2.8 Protein MD-Setup workflow

The **last step** of this tutorial illustrates the building of a **complex workflow** using the **BioExcel building blocks** library in **command line**. The example used is taken from the **Protein MD Setup** Jupyter Notebook tutorial. It is **strongly recommended** to take a look at this notebook before moving on to the next sections of this tutorial, as it contains information for all the **building blocks** used. The aim of this tutorial is to illustrate how to build a **command line workflow** using the **BioExcel building blocks**. For information about the science behind every step of the workflow, please refer to the **Protein MD Setup** Jupyter Notebook tutorial. The workflow presented in the next cells is a translation of the very same workflow to **Python + YAML files**, including the same number of steps (23) and **building blocks**.

Steps:

First of all, let's define the **steps of the workflow**.

- **Fetching PDB Structure**: step 1
- **Fix Protein Structure**: step 2
- **Create Protein System Topology**: step 3
- **Create Solvent Box**: step 4
- **Fill the Box with Water Molecules**: step 5
- **Adding Ions**: steps 6 and 7
- **Energetically Minimize the System**: steps 8, 9 and 10
- **Equilibrate the System (NVT)**: steps 11, 12 and 13
- **Equilibrate the System (NPT)**: steps 14, 15 and 16
- **Free Molecular Dynamics Simulation**: steps 17 and 18

- **Post-processing Resulting 3D Trajectory**: steps 19 to 23

Mandatory and optional **inputs** and **outputs** of every **building block** can be consulted in the appropriate **documentation** pages from the corresponding **BioExcel building block category** (see updated table here).

As explained in the previous sections, the workflow should be **split in two different files**: a **Python script** including the **workflow pipeline**, and a **YAML file** containing all the **required inputs**.

Python Script:

The steps to convert the **Protein MD Setup Jupyter Notebook tutorial** to a **command line workflow** are simple:

1. **Importing** all the needed **libraries**: **system libraries** (sys, os, argparse) and **BioExcel building block libraries** (biobb_*).
2. Parsing the **input configuration file (YAML)** (see previous steps of this tutorial), and dividing the information in **global paths** and **global properties**. Optionally **initializing a global log** file.
3. Declaring the **steps of the workflow**, one by one, using as inputs the **global paths** and **global properties**, identified by the corresponding step name.

NOTE: Remember that going from a **Jupyter Notebook** to a **command line python script** is as easy as **exporting** the notebook to a **Python script** (Menu –> File –> Download as –> Python (.py)). However, splitting the workflow in two files allows changing **input parameters** without the need of any modification to the main **Python script**, really convenient if **many executions** are planned.

The final **Protein MD Setup Jupyter Notebook tutorial** converted to an independent **Python script** follows:

```
#!/usr/bin/env python3

# Conversion of the BioExcel building blocks Protein MD Setup Jupyter Notebook␣
↪tutorial
# to a command line workflow with two files: Python Script and YAML input␣
↪configuration file
# Example of Python Script (should be accompanied by a YAML input configuration file)

# Importing all the needed libraries
import sys
import os
import time
import argparse
from biobb_common.configuration import settings
from biobb_common.tools import file_utils as fu
from biobb_io.api.pdb import Pdb
from biobb_model.model.fix_side_chain import FixSideChain
from biobb_model.model.mutate import Mutate
from biobb_md.gromacs.pdb2gmx import Pdb2gmx
from biobb_md.gromacs.editconf import Editconf
from biobb_md.gromacs.solvate import Solvate
from biobb_md.gromacs.grompp import Grompp
from biobb_md.gromacs.genion import Genion
from biobb_md.gromacs.mdrun import Mdrun
from biobb_analysis.gromacs.gmx_rms import GMXRms
from biobb_analysis.gromacs.gmx_rgyr import GMXRgyr
from biobb_analysis.gromacs.gmx_energy import GMXEnergy
from biobb_analysis.gromacs.gmx_image import GMXImage
from biobb_analysis.gromacs.gmx_trjconv_str import GMXTrjConvStr

# Receiving the input configuration file (YAML)
```

(continues on next page)

(continued from previous page)

```
conf = settings.ConfReader(sys.argv[1])

# Initializing a global log file
global_log, _ = fu.get_logs(path=conf.get_working_dir_path(), light_format=True)

# Parsing the input configuration file (YAML);
# Dividing it in global paths and global properties
global_prop = conf.get_prop_dic(global_log=global_log)
global_paths = conf.get_paths_dic()

# Declaring the steps of the workflow, one by one
# Using as inputs the global paths and global properties
# identified by the corresponding step name
# Writing information about each step to the global log
global_log.info("step1_pdb: Dowload the initial Structure")
Pdb(**global_paths["step1_pdb"], properties=global_prop["step1_pdb"]).launch()

global_log.info("step2_fixsidechain: Modeling the missing heavy atoms in the␣
↪structure side chains")
FixSideChain(**global_paths["step2_fixsidechain"], properties=global_prop["step2_
↪fixsidechain"]).launch()

global_log.info("step3_pdb2gmx: Generate the topology")
Pdb2gmx(**global_paths["step3_pdb2gmx"], properties=global_prop["step3_pdb2gmx"]).
↪launch()

global_log.info("step4_editconf: Create the solvent box")
Editconf(**global_paths["step4_editconf"], properties=global_prop["step4_editconf"]).
↪launch()

global_log.info("step5_solvate: Fill the solvent box with water molecules")
Solvate(**global_paths["step5_solvate"], properties=global_prop["step5_solvate"]).
↪launch()

global_log.info("step6_grompp_genion: Preprocess ion generation")
Grompp(**global_paths["step6_grompp_genion"], properties=global_prop["step6_grompp_
↪genion"]).launch()

global_log.info("step7_genion: Ion generation")
Genion(**global_paths["step7_genion"], properties=global_prop["step7_genion"]).
↪launch()

global_log.info("step8_grompp_min: Preprocess energy minimization")
Grompp(**global_paths["step8_grompp_min"], properties=global_prop["step8_grompp_min
↪"]).launch()

global_log.info("step9_mdrun_min: Execute energy minimization")
Mdrun(**global_paths["step9_mdrun_min"], properties=global_prop["step9_mdrun_min"]).
↪launch()

global_log.info("step10_energy_min: Compute potential energy during minimization")
GMXEnergy(**global_paths["step10_energy_min"], properties=global_prop["step10_energy_
↪min"]).launch()

global_log.info("step11_grompp_nvt: Preprocess system temperature equilibration")
Grompp(**global_paths["step11_grompp_nvt"], properties=global_prop["step11_grompp_nvt
↪"]).launch()
```

(continues on next page)

(continued from previous page)

```
global_log.info("step12_mdrun_nvt: Execute system temperature equilibration")
Mdrun(**global_paths["step12_mdrun_nvt"], properties=global_prop["step12_mdrun_nvt"]).
↪launch()

global_log.info("step13_energy_nvt: Compute temperature during NVT equilibration")
GMXEnergy(**global_paths["step13_energy_nvt"], properties=global_prop["step13_energy_
↪nvt"]).launch()

global_log.info("step14_grompp_npt: Preprocess system pressure equilibration")
Grompp(**global_paths["step14_grompp_npt"], properties=global_prop["step14_grompp_npt
↪"]).launch()

global_log.info("step15_mdrun_npt: Execute system pressure equilibration")
Mdrun(**global_paths["step15_mdrun_npt"], properties=global_prop["step15_mdrun_npt"]).
↪launch()

global_log.info("step16_energy_npt: Compute Density & Pressure during NPT␣
↪equilibration")
GMXEnergy(**global_paths["step16_energy_npt"], properties=global_prop["step16_energy_
↪npt"]).launch()

global_log.info("step17_grompp_md: Preprocess free dynamics")
Grompp(**global_paths["step17_grompp_md"], properties=global_prop["step17_grompp_md
↪"]).launch()

global_log.info("step18_mdrun_md: Execute free molecular dynamics simulation")
Mdrun(**global_paths["step18_mdrun_md"], properties=global_prop["step18_mdrun_md"]).
↪launch()

global_log.info("step19_rmsfirst: Compute Root Mean Square deviation against␣
↪equilibrated structure (first)")
GMXRms(**global_paths["step19_rmsfirst"], properties=global_prop["step19_rmsfirst"]).
↪launch()

global_log.info("step20_rmsexp: Compute Root Mean Square deviation against minimized␣
↪structure (exp)")
GMXRms(**global_paths["step20_rmsexp"], properties=global_prop["step20_rmsexp"]).
↪launch()

global_log.info("step21_rgyr: Compute Radius of Gyration to measure the protein␣
↪compactness during the free MD simulation")
GMXRgyr(**global_paths["step21_rgyr"], properties=global_prop["step21_rgyr"]).launch()

global_log.info("step22_image: Imaging the resulting trajectory")
GMXImage(**global_paths["step22_image"], properties=global_prop["step22_image"]).
↪launch()

global_log.info("step23_dry: Removing water molecules and ions from the resulting␣
↪structure")
GMXTrjConvStr(**global_paths["step23_dry"], properties=global_prop["step23_dry"]).
↪launch()
```

Input YAML Configuration File:

The **YAML configuration file** containing the **input parameters** for the **Protein MD Setup** workflow includes the **workflow global properties** and the specific **paths** and **properties** of the previously introduced **23 steps**.

The **workflow global properties** are typically stated at the beginning of the YAML file (e.g. *working_dir_path: md_tutorial*).

The **name of the steps** should **match** the one written in the **Python Script**: step name is the **link** of the two files (e.g. *step2_fixsidechain*).

For **each step**, the **paths** section contains the list of **input and/or output files**, either with **file names** (with relative or absolute system paths) or with **dependencies** from previous steps of the workflow (see previous sections of the tutorial).

For **each step**, the **properties** section contains the list of **building block-specific input parameters**, if needed (properties are optional). The list of specific properties for each of the **BioExcel building blocks** can be found in the appropriate documentation sites (see updated table here).

The **input paths** and **parameters** for the **Protein MD Setup Jupyter Notebook tutorial** converted to an independent **Input YAML Configuration file** follows:

```yaml
# Example of a YAML configuration file for a BioExcel building blocks workflow

working_dir_path: md_tutorial      # Folder to write i/o files of the workflow steps
can_write_console_log: False       # Verbose writing of log information
restart: True                      # Skip steps already performed

step1_pdb:
  paths:
    output_pdb_path: structure.pdb
  properties:
    pdb_code: 1aki

step2_fixsidechain:
  paths:
    input_pdb_path: dependency/step1_pdb/output_pdb_path
    output_pdb_path: fixsidechain.pdb

step3_pdb2gmx:
  paths:
    input_pdb_path: dependency/step2_fixsidechain/output_pdb_path
    output_gro_path: pdb2gmx.gro
    output_top_zip_path: pdb2gmx_top.zip

step4_editconf:
  paths:
    input_gro_path: dependency/step3_pdb2gmx/output_gro_path
    output_gro_path: editconf.gro

step5_solvate:
  paths:
    input_solute_gro_path: dependency/step4_editconf/output_gro_path
    output_gro_path: solvate.gro
    input_top_zip_path: dependency/step3_pdb2gmx/output_top_zip_path
    output_top_zip_path: solvate_top.zip

step6_grompp_genion:
  paths:
```

(continues on next page)

(continued from previous page)

```
    input_gro_path: dependency/step5_solvate/output_gro_path
    input_top_zip_path: dependency/step5_solvate/output_top_zip_path
    output_tpr_path: gppion.tpr
  properties:
    mdp:
      nsteps: 5000
    simulation_type: minimization

step7_genion:
  paths:
    input_tpr_path: dependency/step6_grompp_genion/output_tpr_path
    output_gro_path: genion.gro
    input_top_zip_path: dependency/step5_solvate/output_top_zip_path
    output_top_zip_path: genion_top.zip
  properties:
    neutral: True
    concentration: 0.05

step8_grompp_min:
  paths:
    input_gro_path: dependency/step7_genion/output_gro_path
    input_top_zip_path: dependency/step7_genion/output_top_zip_path
    output_tpr_path: gppmin.tpr
  properties:
    mdp:
      nsteps: 5000
      emtol: 500
    simulation_type: minimization

step9_mdrun_min:
  paths:
    input_tpr_path: dependency/step8_grompp_min/output_tpr_path
    output_trr_path: min.trr
    output_gro_path: min.gro
    output_edr_path: min.edr
    output_log_path: min.log

step10_energy_min:
  paths:
      input_energy_path: dependency/step9_mdrun_min/output_edr_path
      output_xvg_path: min_ene.xvg
  properties:
      terms:  ["Potential"]

step11_grompp_nvt:
  paths:
    input_gro_path: dependency/step9_mdrun_min/output_gro_path
    input_top_zip_path: dependency/step7_genion/output_top_zip_path
    output_tpr_path: gppnvt.tpr
  properties:
    mdp:
      nsteps: 5000
    simulation_type: nvt

step12_mdrun_nvt:
  paths:
    input_tpr_path: dependency/step11_grompp_nvt/output_tpr_path
```

(continues on next page)

(continued from previous page)

```
    output_trr_path: nvt.trr
    output_gro_path: nvt.gro
    output_edr_path: nvt.edr
    output_log_path: nvt.log
    output_cpt_path: nvt.cpt

step13_energy_nvt:
  paths:
      input_energy_path: dependency/step12_mdrun_nvt/output_edr_path
      output_xvg_path: nvt_temp.xvg
  properties:
      terms:  ["Temperature"]

step14_grompp_npt:
  paths:
    input_gro_path: dependency/step12_mdrun_nvt/output_gro_path
    input_top_zip_path: dependency/step7_genion/output_top_zip_path
    output_tpr_path: gppnpt.tpr
    input_cpt_path: dependency/step12_mdrun_nvt/output_cpt_path
  properties:
    mdp:
      nsteps: 5000
    simulation_type: npt

step15_mdrun_npt:
  paths:
    input_tpr_path: dependency/step14_grompp_npt/output_tpr_path
    output_trr_path: npt.trr
    output_gro_path: npt.gro
    output_edr_path: npt.edr
    output_log_path: npt.log
    output_cpt_path: npt.cpt

step16_energy_npt:
  paths:
      input_energy_path: dependency/step15_mdrun_npt/output_edr_path
      output_xvg_path: npt_den_press.xvg
  properties:
      terms:  ["Pressure","Density"]

step17_grompp_md:
  paths:
    input_gro_path: dependency/step15_mdrun_npt/output_gro_path
    input_top_zip_path: dependency/step7_genion/output_top_zip_path
    output_tpr_path: gppmd.tpr
    input_cpt_path: dependency/step15_mdrun_npt/output_cpt_path
  properties:
    mdp:
      nsteps: 50000
    simulation_type: free

step18_mdrun_md:
  paths:
    input_tpr_path: dependency/step17_grompp_md/output_tpr_path
    output_trr_path: md.trr
    output_gro_path: md.gro
    output_edr_path: md.edr
```

(continues on next page)

(continued from previous page)

```
    output_log_path: md.log
    output_cpt_path: md.cpt

step19_rmsfirst:
  paths:
      input_structure_path: dependency/step17_grompp_md/output_tpr_path
      input_traj_path: dependency/step18_mdrun_md/output_trr_path
      output_xvg_path: md_rmsdfirst.xvg
  properties:
      selection: Backbone

step20_rmsexp:
  paths:
      input_structure_path: dependency/step8_grompp_min/output_tpr_path
      input_traj_path: dependency/step18_mdrun_md/output_trr_path
      output_xvg_path: md_rmsdexp.xvg
  properties:
      selection: Backbone

step21_rgyr:
  paths:
      input_structure_path: dependency/step17_grompp_md/output_tpr_path
      input_traj_path: dependency/step18_mdrun_md/output_trr_path
      output_xvg_path: md_rgyr.xvg
  properties:
      selection: Backbone

step22_image:
  paths:
      input_traj_path: dependency/step18_mdrun_md/output_trr_path
      input_top_path: dependency/step17_grompp_md/output_tpr_path
      output_traj_path: imaged_traj.trr
  properties:
      center_selection: Protein
      output_selection: Protein
      pbc: mol
      center : True

step23_dry:
  paths:
      input_structure_path: dependency/step18_mdrun_md/output_gro_path
      input_top_path: dependency/step17_grompp_md/output_tpr_path
      output_str_path: imaged_structure.gro
  properties:
      selection: Protein
```

Running the workflow:

The **final step of the process** is **running the workflow**. For that, the complete workflow **Python Script** should be written to a file (e.g. biobb_MDsetup_tutorial.py), the **YAML configuration input file** should be written to a separate file (e.g. biobb_MDsetup_tutorial.yaml) and finally both files should be used for the **command line execution**.

As in the previous example, it is important to note that in order to **properly run the workflow**, all the **software dependencies** should have been previously **installed in the system**, following the **Protein MD tutorial** installation steps (*Conda Installation and Launch*).

The final **command line** is shown in the cell below:

```
python biobb_MDsetup_tutorial.py biobb_MDsetup_tutorial.yaml
```

Workflow output

The **execution of the workflow** will write information to the **standard output** such as the **tools** being executed, the **command lines**, **inputs and outputs** used, and **state** of each step (exit codes). Thanks to the additional information added in the **Python Script**, the **log file** contains a line for **each of the steps** being executed. The next cell contains a **real output** for the execution of the **Protein MD Setup command line workflow**:

```
2020-05-21 08:51:15,517 [MainThread  ] [INFO ]  step1_pdb: Download the initial␣
↪Structure
2020-05-21 08:51:15,518 [MainThread  ] [INFO ]      Downloading: 1aki from: https://
↪files.rcsb.org/download/1aki.pdb
2020-05-21 08:51:16,545 [MainThread  ] [INFO ]      Writting pdb to: /Users/biobb_
↪tutorials/VT/cli/Yaml/md_tutorial/step1_pdb/structure.pdb
2020-05-21 08:51:16,545 [MainThread  ] [INFO ]      Filtering lines NOT starting with␣
↪one of these words: ['ATOM', 'MODEL', 'ENDMDL']
2020-05-21 08:51:16,547 [MainThread  ] [INFO ]  step2_fixsidechain: Modeling the␣
↪missing heavy atoms in the structure side chains
2020-05-21 08:51:17,087 [MainThread  ] [INFO ]      Executing: check_structure -i /
↪Users/biobb_tutorials/VT/cli/Yaml/md_tutorial/step1...
2020-05-21 08:51:17,088 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:51:17,088 [MainThread  ] [INFO ]  step3_pdb2gmx: Generate the topology
2020-05-21 08:51:17,605 [MainThread  ] [INFO ]      Executing: gmx pdb2gmx -f /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step2_fix...
2020-05-21 08:51:17,606 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:51:17,606 [MainThread  ] [INFO ]      Compressing topology to: /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step3_pdb2gmx/pdb2gmx_top.zip
2020-05-21 08:51:17,621 [MainThread  ] [INFO ]      Removed: ['step3_pdb2gmx_p2g.top',
↪ 'step3_pdb2gmx_p2g.itp']
2020-05-21 08:51:17,621 [MainThread  ] [INFO ]  step4_editconf: Create the solvent box
2020-05-21 08:51:17,645 [MainThread  ] [INFO ]      Centering molecule in the box.
2020-05-21 08:51:17,645 [MainThread  ] [INFO ]      Distance of the box to molecule: ␣
↪ 1.00
2020-05-21 08:51:17,645 [MainThread  ] [INFO ]      Box type: cubic
2020-05-21 08:51:17,680 [MainThread  ] [INFO ]      Executing: gmx editconf -f /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step3_pd...
2020-05-21 08:51:17,681 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:51:17,681 [MainThread  ] [INFO ]  step5_solvate: Fill the solvent box␣
↪with water molecules
2020-05-21 08:51:18,031 [MainThread  ] [INFO ]      Executing: gmx solvate -cp /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step4_ed...
2020-05-21 08:51:18,031 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:51:18,046 [MainThread  ] [INFO ]      Removed: ['0fd87786-2e54-429b-
↪9c0f-4861286178d9']
2020-05-21 08:51:18,046 [MainThread  ] [INFO ]  step6_grompp_genion: Preprocess ion␣
↪generation
2020-05-21 08:51:18,073 [MainThread  ] [INFO ]      Will run a minimization md of␣
↪5000 steps
2020-05-21 08:51:18,478 [MainThread  ] [INFO ]      Executing: gmx grompp -f step6_
↪grompp_genion_grompp.mdp -c /Users/biobb_tutorials/...
2020-05-21 08:51:18,478 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:51:18,479 [MainThread  ] [INFO ]      Removed: ['3e68b0e0-75f8-4696-
↪9991-d56cc6eb18ee', 'mdout.mdp', 'step6_grompp_genion_grompp.mdp']
```

(continues on next page)

(continued from previous page)

```
2020-05-21 08:51:18,479 [MainThread  ] [INFO ]  step7_genion: Ion generation
2020-05-21 08:51:18,498 [MainThread  ] [INFO ]      To reach up 0.05 mol/litre␣
↪concentration
2020-05-21 08:51:18,638 [MainThread  ] [INFO ]      Executing: echo "SOL" | gmx␣
↪genion -s /Users/biobb_tutorials/VT/cli/Yaml/md_tutori...
2020-05-21 08:51:18,638 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:51:18,653 [MainThread  ] [INFO ]      Removed: ['5f216174-ac78-4592-
↪bd88-742c4d8fcf67']
2020-05-21 08:51:18,653 [MainThread  ] [INFO ]  step8_grompp_min: Preprocess energy␣
↪minimization
2020-05-21 08:51:18,678 [MainThread  ] [INFO ]      Will run a minimization md of␣
↪5000 steps
2020-05-21 08:51:19,083 [MainThread  ] [INFO ]      Executing: gmx grompp -f step8_
↪grompp_min_grompp.mdp -c /Users/biobb_tutorials/VT/...
2020-05-21 08:51:19,084 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:51:19,085 [MainThread  ] [INFO ]      Removed: ['8bc12f7e-5df6-4b86-
↪b19e-93363d9ed2b4', 'mdout.mdp', 'step8_grompp_min_grompp.mdp']
2020-05-21 08:51:19,085 [MainThread  ] [INFO ]  step9_mdrun_min: Execute energy␣
↪minimization
2020-05-21 08:53:13,890 [MainThread  ] [INFO ]      Executing: gmx mdrun -s /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step8_gromp...
2020-05-21 08:53:13,891 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:53:13,892 [MainThread  ] [INFO ]      Removed: []
2020-05-21 08:53:13,892 [MainThread  ] [INFO ]  step10_energy_min: Compute potential␣
↪energy during minimization
2020-05-21 08:53:13,932 [MainThread  ] [INFO ]      Executing: gmx energy -f /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step9_mdru...
2020-05-21 08:53:13,932 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:53:13,934 [MainThread  ] [INFO ]  step11_grompp_nvt: Preprocess system␣
↪temperature equilibration
2020-05-21 08:53:13,956 [MainThread  ] [INFO ]      Will run a nvt md of 5000 steps
2020-05-21 08:53:14,490 [MainThread  ] [INFO ]      Executing: gmx grompp -f step11_
↪grompp_nvt_grompp.mdp -c /Users/biobb_tutorials/VT...
2020-05-21 08:53:14,490 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:53:14,491 [MainThread  ] [INFO ]      Removed: ['fcc69355-9536-44b5-
↪9b26-bda7f6ec9bf6', 'mdout.mdp', 'step11_grompp_nvt_grompp.mdp']
2020-05-21 08:53:14,491 [MainThread  ] [INFO ]  step12_mdrun_nvt: Execute system␣
↪temperature equilibration
2020-05-21 08:55:26,785 [MainThread  ] [INFO ]      Executing: gmx mdrun -s /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step11_grom...
2020-05-21 08:55:26,785 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:55:26,787 [MainThread  ] [INFO ]      Removed: ['traj_comp.xtc']
2020-05-21 08:55:26,787 [MainThread  ] [INFO ]  step13_energy_nvt: Compute␣
↪temperature during NVT equilibration
2020-05-21 08:55:26,825 [MainThread  ] [INFO ]      Executing: gmx energy -f /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step12_mdr...
2020-05-21 08:55:26,825 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:55:26,826 [MainThread  ] [INFO ]  step14_grompp_npt: Preprocess system␣
↪pressure equilibration
2020-05-21 08:55:26,846 [MainThread  ] [INFO ]      Will run a npt md of 5000 steps
2020-05-21 08:55:27,422 [MainThread  ] [INFO ]      Executing: gmx grompp -f step14_
↪grompp_npt_grompp.mdp -c /Users/biobb_tutorials/VT...
2020-05-21 08:55:27,422 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:55:27,424 [MainThread  ] [INFO ]      Removed: ['b8b9a946-8614-4930-
↪9dcf-2de71cbd1d58', 'mdout.mdp', 'step14_grompp_npt_grompp.mdp']
2020-05-21 08:55:27,424 [MainThread  ] [INFO ]  step15_mdrun_npt: Execute system␣
↪pressure equilibration
```

(continues on next page)

(continued from previous page)

```
2020-05-21 08:57:41,425 [MainThread  ] [INFO ]      Executing: gmx mdrun -s /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step14_grom...
2020-05-21 08:57:41,425 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:57:41,427 [MainThread  ] [INFO ]      Removed: ['traj_comp.xtc']
2020-05-21 08:57:41,427 [MainThread  ] [INFO ]  step16_energy_npt: Compute Density &␣
↪Pressure during NPT equilibration
2020-05-21 08:57:41,466 [MainThread  ] [INFO ]      Executing: gmx energy -f /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step15_mdr...
2020-05-21 08:57:41,466 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:57:41,466 [MainThread  ] [INFO ]  step17_grompp_md: Preprocess free␣
↪dynamics
2020-05-21 08:57:41,487 [MainThread  ] [INFO ]      Will run a free md of 100.0 pico␣
↪seconds
2020-05-21 08:57:41,895 [MainThread  ] [INFO ]      Executing: gmx grompp -f step17_
↪grompp_md_grompp.mdp -c /Users/biobb_tutorials/VT/...
2020-05-21 08:57:41,895 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 08:57:41,896 [MainThread  ] [INFO ]      Removed: ['1e93ee86-7a67-47d7-
↪a877-aa7e3ce5e0fb', 'mdout.mdp', 'step17_grompp_md_grompp.mdp']
2020-05-21 08:57:41,896 [MainThread  ] [INFO ]  step18_mdrun_md: Execute free␣
↪molecular dynamics simulation
2020-05-21 09:19:08,588 [MainThread  ] [INFO ]      Executing: gmx mdrun -s /Users/
↪biobb_tutorials/VT/cli/Yaml/md_tutorial/step17_grom...
2020-05-21 09:19:08,589 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 09:19:08,591 [MainThread  ] [INFO ]      Removed: ['traj_comp.xtc']
2020-05-21 09:19:08,591 [MainThread  ] [INFO ]  step19_rmsfirst: Compute Root Mean␣
↪Square deviation against equilibrated structure (first)
2020-05-21 09:19:08,903 [MainThread  ] [INFO ]      Executing: echo "Backbone Backbone
↪" | gmx rms -s /Users/biobb_tutorials/VT/cli/Yam...
2020-05-21 09:19:08,903 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 09:19:08,903 [MainThread  ] [INFO ]  step20_rmsexp: Compute Root Mean␣
↪Square deviation against minimized structure (exp)
2020-05-21 09:19:09,182 [MainThread  ] [INFO ]      Executing: echo "Backbone Backbone
↪" | gmx rms -s /Users/biobb_tutorials/VT/cli/Yam...
2020-05-21 09:19:09,182 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 09:19:09,182 [MainThread  ] [INFO ]  step21_rgyr: Compute Radius of␣
↪Gyration to measure the protein compactness during the free MD simulation
2020-05-21 09:19:09,425 [MainThread  ] [INFO ]      Executing: echo "Backbone" | gmx␣
↪gyrate -s /Users/biobb_tutorials/VT/cli/Yaml/md_t...
2020-05-21 09:19:09,425 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 09:19:09,425 [MainThread  ] [INFO ]  step22_image: Imaging the resulting␣
↪trajectory
2020-05-21 09:19:09,740 [MainThread  ] [INFO ]      Executing: echo "Protein" "Protein
↪" | gmx trjconv -f /Users/biobb_tutorials/VT/cli...
2020-05-21 09:19:09,740 [MainThread  ] [INFO ]      Exit code 0
2020-05-21 09:19:09,741 [MainThread  ] [INFO ]  step23_dry: Removing water molecules␣
↪and ions from the resulting structure
2020-05-21 09:19:09,913 [MainThread  ] [INFO ]      Executing: echo "Protein" | gmx␣
↪trjconv -f /Users/biobb_tutorials/VT/cli/Yaml/md_t...
2020-05-21 09:19:09,913 [MainThread  ] [INFO ]      Exit code 0
```

Next steps

Now that you have the **Protein MD Setup command line workflow** ready, try to play with it doing **simple exercises**:

- Change input PDB code.
- Make equilibration phases long.

- Remove / add new steps into the workflow.
- Introduce a loop of 2 different PDB codes.
- Introduce a mutation.
- Program an Alanine-Scanning.

Some of these exercises (and more) can be found in the Virtual Training section (examples link) of the BioExcel building blocks website.

If you have found the tutorial interesting, please take a look at the **BioExcel events**, where new **Webinars** and **Virtual Trainings** about the **BioExcel building blocks** are being planned.

1.2.9 Questions & Comments

Questions, issues, suggestions and comments are really welcome!

- GitHub issues:
  - https://github.com/bioexcel/biobb
- BioExcel forum:
  - https://ask.bioexcel.eu/c/BioExcel-Building-Blocks-library

CHAPTER 2

Github repository.