GROMACS CYPWare 1.0 Script and HTML Tool Manual

Introduction:

GROMACS CYPWare is a web interface that calculates Marcus ET parameters for Cytochrome P450 BM3 reactions. This tool is developed by Pulkit Asati under the supervision of Dr. Vaibhav A. Dixit, Assistant Professor in the Department of Medicinal Chemistry at NIPER Guwahati. The software, GUI, websites, tools, layouts, and logos are copyrighted and the exclusive property of the Department of Medicinal Chemistry, NIPER Guwahati.

Dependencies:

Before running GROMACS CYPWare 1.0, you need to ensure that the following open-source tools are installed:

1. Openbabel 3.1 or higher
2. AmberTools
3. GROMACS

Installation:

Openbabel 3.1 or higher Openbabel can be installed from GitHub, Sourceforge, or as a conda package. If you have conda, you can install Openbabel with the following command: conda install -c conda-forge openbabel Alternatively, you can install it from the main website https://www.anaconda.com/ and follow the installation instructions.

AmberTools AmberTools is available for free from the http://ambermd.org/ website. Alternatively, you can install it as a conda package with the following command: conda install -c conda-forge ambertools

GROMACS GROMACS can be downloaded from https://gitlab.com/gromacs/gromacs. It is free, open-source software released under the GNU General Public License (GPL), and starting with version 4.6, the GNU Lesser General Public License (LGPL).

Download the CYPWare code from the Github page of the PI.

https://github.com/Dixit-s-lab/CYPWare-1.0

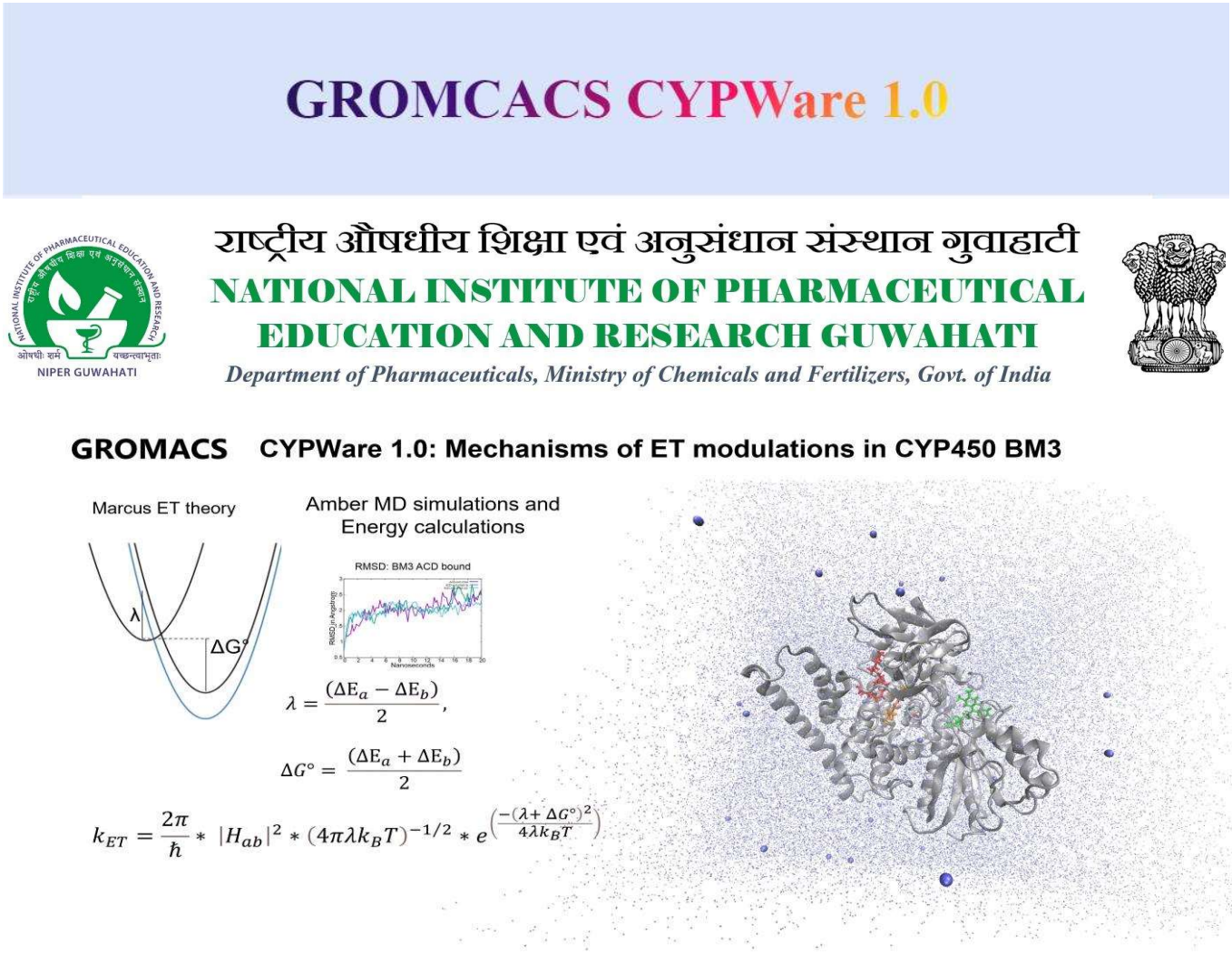
Unzip all the files and folders into a directory.

Open the "cypwarecode.html" file in a web browser.

Usage:

Once you have installed all the dependencies, you can run GROMACS CYPWare 1.0. To use the tool, follow the steps below:

Launch the GROMACS CYPWare web interface.



Fill the input entries for the system you want to simulate.

Set the necessary parameters.

Click on the "Submit" button to get the output file for the calculation.

Wait for the calculation to complete.

Download the output files.

Properly cite the software and publication associated with the same for any work/report/thesis/research-article/review-article resulting from the use of CYPWare 1.0.

Commercial Usage:

For commercial usage of CYPWare 1.0, please contact the PI at vaibhavadixit@gmail.com or vaibhav@niperguwahati.in.

Conclusion:

GROMACS CYPWare 1.0 is a powerful tool for calculating Marcus ET parameters for Cytochrome P450 BM3 reactions. By following the steps outlined above, you can use the tool to simulate and analyze your system. Please ensure that all dependencies are installed before running the tool to avoid errors.

Once the user has completed these steps, they should be able to access the web interface and begin using the tool to calculate Marcus ET parameters for Cytochrome P450 BM3 reactions.

It is important to note that the tool has several dependencies, including Openbabel 3.1 or higher, AmberTools, and GROMACS. The user must ensure that all of these dependencies are installed before attempting to use the tool. The tool's documentation provides instructions for installing these dependencies. Additionally, the tool is subject to copyright protection, and any work resulting from the use of the tool should properly cite the software and publication associated with it.

This **HTML code** is a form that collects input from a user. The input fields are:

- LIGAND NAME (without extension)
- MULTIPLICITY of LIGAND (OPTIONAL)
- CHARGE OF LIGAND
- PROTEIN NAME (without extension)
- NAME OF FOLDER FOR ALL OUTPUT
- NAME OF FOLDER FOR FERRIC OR OXIDIZED STATE
- NAME OF FOLDER FOR FERROUS OR REDUCED STATE
- POSE NUMBER FROM VINA OUTPUT FOR USING AS LIGAND
- ET CALCULATION FOR(options—bound : apo : both)

script **run.sh**

This script is designed to automate the calculation of molecular dynamics simulations using GROMACS and AMBER software for different calculation types (apo, bound, or both). It takes an input file in the format of **GROMACS_CYPWareData.txt** to define the simulation parameters and then runs the appropriate set of calculations based on the input.

The following is a step-by-step guide on how to use this script:

Step 1: copy an input file

Copy an input file named **GROMACS_CYPWareData.txt** in the same directory where the **run.sh** script is located. This input file should contain the necessary simulation parameters and values, in the following format:

ligand=NPG

protein=1bvy-af

dirpath=NPG-cypware

oxddir=NPG-oxd

reddir=NPG-red

lignad_multiplicity=1

posenumber=1

lignad_charge=0

calculation_type=apo

You can change these parameter values as per your requirements.

Step 2: Run the script

Open your terminal and navigate to the directory containing the **run.sh** script and the input file. Run the following command:

bash ./run.sh GROMACS_CYPWareData.txt

This will run the script using the input file as an argument.

The command **bash ./run.sh GROMACS_CYPWareData.txt** is executing a shell script called **run.sh** using the Bash shell. The script takes a single argument, which is the file name **GROMACS_CYPWareData.txt**.

The contents of **GROMACS_CYPWareData.txt** are a set of key-value pairs that represent parameters for the script. These include the ligand, protein, directory paths, charge and multiplicity of the ligand, and the type of calculation to be performed (either "apo", "bound", or "both").

The steps in the script are:

- The script reads in an input file ($1) and exports the variables specified in the input file using xargs.
- The current working directory (cwd) is saved.
- Several scripts are sourced, including amber.sh and files.sh, which may contain functions or variables needed for the simulations.
- A welcome message is printed by calling the function "welcome_part1".
- The contents of the input file are printed using "cat".
- A new directory is created using the variable "dirpath".
- Depending on the value of the variable "calculation_type" (which should be defined in the input file), either the "apo_type" or "complex_type" function is called, or an error message is printed.
- If the value of "calculation_type" is "apo", the "apo_type" function is called, which performs the following steps: a. The protein is prepared in the apo state by calling the "apo_state" function. b. The protein is indexed using the "indexing_apo" function. c. Multiple molecular dynamics (MD) simulations are submitted using the "apo_all_md_submit" function. d. A slurm script is created for the calculation of the energy time correlation (ETC) using the "slurm_sript" function. e. The last step is

appended to the slurm script using "append_commands". f. The slurm script is submitted using sbatch and dependencies on previous jobs are defined.

- If the value of "calculation_type" is "bound", the "complex_type" function is called, which performs similar steps to "apo_type" but for the bound state.
- If the value of "calculation_type" is "both", both the "apo_type" and "complex_type" functions are called in sequence.
- If the value of "calculation_type" is not one of the specified values, an error message is printed and the script exits

That's it! You have successfully used the **run.sh** script to automate molecular dynamics simulations using GROMACS and AMBER software.

welcome.sh

The **welcome.sh** script is a Bash script that defines three functions: **info**, **welcome_part1**, and **welcome_part2**.

Here are the steps that the script will do when executed:

The script will start with the "shebang" line (**#!/bin/bash**) which tells the system to interpret the script using the Bash shell.

The **info** function is defined. This function is not executed in the script.

The **welcome_part1** function is defined. This function defines an array of color codes, loops through the lines of the output of the **info** function (which is not shown in the script), assigns a color code to each line based on the index of the color array, and prints each line in the assigned color to the terminal. This function is not executed in the script.

The **part2_intro** function is defined. This function prints a multi-line string to the terminal. The string contains a stylized text that spells out "###### # ###### ####### ##### #".

The **welcome_part2** function is defined. This function works the same way as **welcome_part1**, but instead of reading from the **info** function, it reads from the **part2_intro** function and assigns color codes to each line of the multi-line string.

The script does not execute any of the defined functions, but instead defines them for use in other scripts or for interactive use in the terminal.

files.sh

The **files.sh** script defines three functions named **oxidized_state_files()**, **reduced_state_files()**, and **MD_files()**. It also defines a function named **slurm_script()** that takes two positional parameters.

The oxidized_state_files() function performs the following actions:

Creates a file named **4ZF6_mcpbpy.frcmod**.

Creates a file named **CM1.mol2** .

Creates a file named **FE1.mol2**.

Creates a file named **FMN-sq-H.mol2**.

Creates a file named **HEM.frcmod**.

Creates a file named **HM1.mol2**.

Creates a file named **tleap.in**.

The **reduced_state_files()** function performs the following actions:

Creates a file named **4ZF6_mcpbpy.frcmod**

Creates a file named **CM1.mol2**

Creates a file named **FE1.mol2**.

Creates a file named **FMN-ox-H.frcmod**.

Creates a file named **FMN-ox-H.mol2**

Creates a file named **HEM.frcmod** with the following conten

Creates a file named **HM1.mol2**.

Creates a file named **tleap.in**

The MD_files() function performs the following actions:

Creates a file named **em.mdp**

Creates a file named **nvt.mdp**.

Creates a file named **npt.mdp**

Creates a file named **md.mdp**

Creates a file named **ie.mdp**

The slurm_script() function

takes two positional parameters: **$1** is the script name, and **$2** is the job name. It creates a shell script with the given script name and writes the following content to it:

```
#!/bin/bash -l
#SBATCH --job-name=$2   ##job name
#SBATCH -A nsm     ##account name
#SBATCH --nodes=1  ## number of nodes
#SBATCH --nodelist=node1
#SBATCH --error=job.%J.err    ## any error during job submission or execution
#SBATCH --output=job.%J.out   ##any output after job execution
#SBATCH --partition=GPU_NODES    ##partition name
#SBATCH --gres=gpu:1     ## number of gpu card requires
```

module load gromacs-gpu/2022.4

source /home/software/ambermpi4py/amber22/amber.sh

commands.sh

The script named **commands.sh** has several functions that are defined to carry out specific tasks. Below is a description of each of the functions in the order they appear in the script.

ligand_pose_extraction(): This function extracts the desired ligand pose from a pdbqt file. The function changes to the current working directory (**$cwd**), runs Open Babel (**obabel**) to convert the input pdbqt file into a pdb file, changes directory to the **$dirpath** directory, removes any pdb files that do not correspond to the desired pose number, adds hydrogens to the pdb file, and then extracts only the ATOM records from the pdb file.

ligand_parameterization(): This function parameterizes the extracted ligand pose using the AmberTools suite. The function sources the AmberTools environment script (**amber.sh**), runs **antechamber** to generate a mol2 file and a set of parameters (**frcmod**) for the ligand, runs **antechamber** again to generate an ac file, and finally runs **antechamber** one more time to convert the ac file to a pdb file.

append_commands(): This function appends the commands of a given function to a specified file. The function takes two positional parameters: the name of the function and the name of the file to which the function commands will be appended.

tleap_ligand(): This function generates a **frcmod** file for the bound protein-ligand complex using tleap. The function calls **ligand_pose_extraction()** to extract the desired ligand pose, generates a slurm script (**slurm.sh**) for running tleap, appends the commands of **ligand_parameterization()** to **slurm.sh**, submits the job to the job scheduler using **sbatch**, and waits for the job to finish before cleaning up the working directory and copying the protein and ligand pdb files to the **$dirpath/bound** directory.

parameters_files_generation(): This function creates a new directory and two subdirectories (**$oxddir** and **$reddir**) within it. It then changes directory to **$oxddir** and calls a function (**oxidized_state_files()**) that generates some input files for a molecular dynamics (MD) simulation. The function then changes directory to **$dirpath/$1/$reddir** and calls another function (**reduced_state_files()**) that generates some input files for a reduced state MD simulation.

hashing_tleap(): This function comments out the lines in the **tleap.in** file that add the ligand to the system. It does this for both the **$oxddir** and **$reddir** directories by calling the **commenting_out_ligand_from_tleap()** function twice.

waiting_tleap(): This function waits for the tleap job submitted by **tleap_ligand()** to finish before cleaning up the working directory and copying the ligand parameter files (**$ligand.mol2** and **$ligand.frcmod**) to the **$oxddir** and **$reddir** directories.

copying_files(): This function copies a specified file from **$dirpath** to a specified directory.

both_dir(): This function calls **copying_files()** twice, once for each of the **$oxddir** and **$reddir** directories.

run_tleap(): This function runs tleap to generate input files for an MD simulation. It changes directory to **$dirpath/$1/$2** and runs tleap using the **tleap.in** file in that directory

This function takes two positional arguments (**dirctory_bond/apo** and **dir_oxy/reduced**) and changes directory to **$cwd/$dirpath/$1/$2**. It then runs the **tleap** program using the input file **tleap.in** in that directory. After **tleap** finishes, it runs a Python script called **amb2gro_top_gro.py**, which generates several simulation input files (**protein_solv.prmtop**, **protein_solv.inpcrd**, **complex.top**, **solv_ions.gro**, and **complex.pdb**) in that directory. Finally, it creates a new directory called **presimulation** and moves all input files generated by **tleap** and **amb2gro_top_gro.py** into that directory using a **for** loop.

run_tleap_dir: This function takes one positional argument (**dirctory_bond/apo**) and calls **run_tleap** twice with the same argument and two different second arguments (**$oxddir** and **$reddir**).

apo_state: This function calls two other functions: **parameters_files_generation** with an argument of **apo**, and **hashing_tleap**. **parameters_files_generation** generates several input files needed for MD simulation in the **apo** directory. **hashing_tleap** calculates SHA-1 hashes for each of the files generated by **parameters_files_generation**. Then, **apo_state** calls **run_tleap_dir** with an argument of **apo**.

bound_state: This function calls three other functions: **tleap_ligand**, **parameters_files_generation** with an argument of **bound**, and **waiting_tleap**. **tleap_ligand** generates input files for the MD simulation of a protein-ligand complex. **parameters_files_generation** generates additional input files needed for the simulation in the **bound** directory. **waiting_tleap** checks if the **tleap** process from **tleap_ligand** has finished before proceeding. Finally, **bound_state** calls **run_tleap_dir** with an argument of **bound**.

index_complex: This function takes one positional argument (**oxy/reduced**) and changes directory to **$cwd/$dirpath/bound/$1**. It then runs **gmx make_ndx** with the input file **solv_ions.gro**, outputting the index file **index.ndx**.

index_apo: This function takes one positional argument (**oxy/reduced**) and changes directory to **$cwd/$dirpath/apo/$1**. It then runs **gmx make_ndx** with the input file **solv_ions.gro**, outputting the index file **index.ndx**.

indexing_bound: This function calls **index_complex** twice with arguments of **$oxddir** and **$reddir**.

indexing_apo: This function calls **index_apo** twice with arguments of **$oxddir** and **$reddir**.

run_MD() - this is a function definition that wraps the commands required to run a molecular dynamics simulation.

gmx_mpi -quiet grompp -f em.mdp -c solv_ions.gro -n index.ndx -p complex.top -po em_mdout.mdp -pp em_processed.top -o em.tpr -maxwarn 3 - this command uses GROMACS to create an input file for the energy minimization simulation (em.tpr) using the

provided parameter and topology files (em.mdp, solv_ions.gro, index.ndx, and complex.top). The output files include em_mdout.mdp and em_processed.top.

gmx_mpi -quiet mdrun -s em.tpr -mp em_processed.top -mn index.ndx -o em_traj.trr -x em_traj_comp.xtc -cpo em_state.cpt -c em_confout.gro -e em_ener.edr -g em_md.log -xvg xmgrace -nb gpu &>> all_output.txt - this command uses GROMACS to run the energy minimization simulation using the input file created in the previous step (em.tpr). The simulation output is saved to several files including em_traj.trr, em_traj_comp.xtc, em_state.cpt, em_confout.gro, em_ener.edr, and em_md.log. The command also redirects the standard error and standard output streams to the file all_output.txt.

echo "11 0" | gmx_mpi -quiet energy -f em_ener.edr -s em.tpr -o potential_energy.xvg -xvg xmgrace -fee -dp -mutot &>> all_output.txt - this command uses GROMACS to calculate the potential energy of the system using the output from the energy minimization simulation (em_ener.edr). The output is saved to potential_energy.xvg, and xmgrace is used to create a graph of the energy values. The command also redirects the standard error and standard output streams to the file all_output.txt.

gmx_mpi -quiet grompp -f nvt.mdp -c em_confout.gro -n index.ndx -p em_processed.top -po nvt_mdout.mdp -pp nvt_processed.top -o nvt.tpr -maxwarn 3 - this command uses GROMACS to create an input file for the NVT equilibration simulation (nvt.tpr) using the output file from the previous step (em_confout.gro) and the same parameter and topology files (nvt.mdp, index.ndx, and em_processed.top). The output files include nvt_mdout.mdp and nvt_processed.top.

Run an NVT equilibration simulation using the **gmx_mpi mdrun** command with input files **nvt.tpr**, **nvt_processed.top**, and **index.ndx**. The output files are **nvt_traj.trr**, **nvt_traj_comp.xtc**, **nvt_state.cpt**, **nvt_confout.gro**, **nvt_ener.edr**, and **nvt_md.log**. The simulation is run using GPU acceleration and the output is redirected to **all_output.txt**.

Use the **gmx_mpi energy** command to extract the temperature information from the **nvt_ener.edr** file and write it to **temperature.xvg**. The output is redirected to **all_output.txt**.

Generate an NPT equilibration input file using the **gmx_mpi grompp** command with input files **npt.mdp**, **nvt_confout.gro**, **nvt_processed.top**, **index.ndx**, **nvt_state.cpt**, and **nvt_ener.edr**. The output files are **npt.tpr**, **npt_processed.top**, and **npt_mdout.mdp**. The maximum number of warnings is set to 3.

Run an NPT equilibration simulation using the **gmx_mpi mdrun** command with input files **npt.tpr**, **npt_processed.top**, and **index.ndx**. The output files are **npt_traj.trr**, **npt_traj_comp.xtc**, **npt_state.cpt**, **npt_confout.gro**, **npt_ener.edr**, and **npt_md.log**. The simulation is run using GPU acceleration and the output is redirected to **all_output.txt**.

Use the **gmx_mpi energy** command to extract the pressure information from the **npt_ener.edr** file and write it to **pressure.xvg**. The output is redirected to **all_output.txt**.

Use the **gmx_mpi energy** command to extract the density information from the **npt_ener.edr** file and write it to **density.xvg**. The output is redirected to **all_output.txt**.

Generate an MD production input file using the **gmx_mpi grompp** command with input files **md.mdp**, **npt_confout.gro**, **npt_processed.top**, **index.ndx**, **npt_state.cpt**, and

npt_ener.edr. The output files are **md.tpr**, **md_processed.top**, and **md_mdout.mdp**. The maximum number of warnings is set to 3.

Run an MD production simulation using the **gmx_mpi mdrun** command with input files **md.tpr**, **md_processed.top**, and **index.ndx**. The output files are **md_traj.trr**, **md_traj_comp.xtc**, **md_state.cpt**, **md_confout.gro**, **md_ener.edr**, and **md_md.log**. The simulation is run using GPU acceleration and the output is redirected to **all_output.txt**.

Use the **gmx_mpi check** command to perform a consistency check on the output files from the MD simulation (**md_traj_comp.xtc**, **md.tpr**, **md_confout.gro**, **md_ener.edr**, and **index.ndx**) and write a report in LaTeX format to **check.txt**. The output is redirected to **check.txt**.

The operation involves using the **gmx_mpi** program to perform trajectory conversion on the **md_traj_comp.xtc** file. This includes centering the molecule, applying periodic boundary conditions, and generating an output file in the **xtc** format. The output is directed to a file called **all_output.txt**.

The operation uses **gmx_mpi** to convert the **md_traj_center.xtc** file to a **pdb** format file called **start.pdb**. The **dump** option is used to extract the first frame of the trajectory. The output is also directed to **all_output.txt**.

The operation involves using **gmx_mpi** to perform fitting on the trajectory data contained in the **md_traj_center.xtc** file. The fitting method used is rotation plus translation, and the output is directed to the **md_fit.xtc** file. The output is also directed to **all_output.txt**.

The operation involves using **gmx_mpi** to perform molecular dynamics simulation. The **grompp** program is used to prepare input files for the simulation. The **mdrun** program is then used to perform the simulation, with various input and output files specified. The output is also directed to **all_output.txt**.

The operation involves using **gmx_mpi** to calculate energy-related properties of the simulated system. The **energy** program is used to calculate single-point energies, with various output files and options specified. The output is directed to **spe.txt**.

The operation involves using **gmx_mpi** to calculate the distance between specified atoms in the simulated system. The **distance** program is used to calculate various distance-related properties, with various output files and options specified. The output is directed to **distance.txt**.

Finally, a new directory called **postsimulation_files** is created, and all files except for certain specified files are moved into this directory. The specified files include **all_output.txt**, **md_fit.xtc**, **md_traj_center.xtc**, **md_traj_comp.xtc**, **postsimulation_files**, and **presimulation**. The **mv** command is used to move the files, and a **for** loop is used to loop over all files in the current directory.

All output from the commands executed in the "run_MD()" function is redirected to "all_output.txt", "spe.txt", and "distance.txt" files.

all_dir_md(): This function takes a single letter job name as a positional parameter and submits a SLURM job for running MD simulations using the specified job name. It calls another function named **slurm_script()** to create a SLURM script for running the MD simulations and submits the job using the **sbatch** command.

fs_md(): This function takes three positional parameters - (i) "apo" or "bound" to indicate whether the simulation is for apo or bound state, (ii) "oxi" or "reduced" to indicate the state of the system, and (iii) a single or two-letter job name for the MD simulation. It calls the **all_dir_md()** function to submit the MD job for all directories.

apo_all_md_submit(): This function calls the **fs_md()** function twice, once for "apo/oxi" and once for "apo/reduced", to submit MD jobs for all directories for the apo state. It stores the job IDs for both MD jobs.

bound_all_md_submit(): This function calls the **fs_md()** function twice, once for "bound/oxi" and once for "bound/reduced", to submit MD jobs for all directories for the bound state. It stores the job IDs for both MD jobs.

copy_files_spec(): This function takes three positional parameters - (i) "apo" or "bound" to indicate whether the simulation is for apo or bound state, (ii) "oxy" or "red" to indicate the state of the system, and (iii) a filename with extension. It copies the specified file from the postsimulation_files directory to a new file with "copied-" prefix.

apo_copy_files_spec(): This function calls the **copy_files_spec()** function for four files (ie_ener.edr, ie_processed.top, index.ndx, ie.tpr) for both "oxy" and "red" states of the apo system.

apo_copy_files_spec_done(): This function calls the **apo_copy_files_spec()** function twice, once for the "oxy" state and once for the "red" state, to copy the specified files for both states of the apo system.

bound_copy_files_spec(): This function calls the **copy_files_spec()** function for four files (ie_ener.edr, ie_processed.top, index.ndx, ie.tpr) for both "oxy" and "red" states of the bound system.

bound_copy_files_spec_done(): This function calls the **bound_copy_files_spec()** function twice, once for the "oxy" state and once for the "red" state, to copy the specified files for both states of the bound system.

recalculating_spe(): This function takes two positional parameters - (i) "bound" or "apo" to indicate whether the simulation is for apo or bound state, and (ii) "oxy" or "reduced" to indicate the state of the system. It performs a series of operations to recalculate the single-point energy (SPE) and distance of the system. It uses GROMACS **mdrun** and **energy** commands to perform the calculations and writes the results to a file named **final_spe_file.txt**.

apo_distance_and_energy(): This function calls the **recalculating_spe()** function twice, once for the **apo** and **oxy** directory and once for the **apo** and **reduced** directory. It calculates the free energy change (**DeltaG**) and distance (**V**) between these two states and stores the results in a file named **apo-FCETP.txt**.

bound_distance_and_energy(): This function calls the **recalculating_spe()** function twice, once for the **bound** and **oxy** directory and once for the **bound** and **reduced** directory. It calculates the free energy change (**DeltaG**) and distance (**V**) between these two states and stores the results in a file named **bound-FCETP.txt**.

final_calculation(): This function takes one positional parameter (**apo** or **bound**) and calculates the free energy change (**DeltaG**) and distance (**V**) between the two states using the results stored in the **apo-FCETP.txt** or **bound-FCETP.txt** file. It then calculates **lambda**, which is the difference in free energy between two intermediate states, and converts the free energy change and lambda from kJ/mol to electronvolts (eV), and stores the results in the **apo-FCETP.txt** or **bound-FCETP.txt** file.

Here are the steps that the function performs:

Define a nested function named **fun_grep_energy_all()**. This function reads energy and distance data from two files and assigns the values to variables.

Call the **fun_grep_energy_all()** function with the positional argument passed to the script.

Calculate lambda and DeltaG using the energy values obtained from step 2.

Calculate the average of D1 and D2 and assign the value to variable V.

Print the energy and distance values, lambda, DeltaG, ev_lambda, and ev_deltag to a file named "<positional argument>-FCETP.txt".

Convert lambda from kj/mol to eV using the **kjol2ev()** function.

Print the converted value of lambda to the "<positional argument>-FCETP.txt" file.

Convert lambda from eV to kJ/mol and assign the value to variable lambda.

Assign the value of ev_deltag to variable DeltaG.

Call a Python script named "ET_Cal.py" with four arguments: lambda, DeltaG, V, and the positional argument passed to the script.

Print "job ended" to the console.

Define a second function named **append_last_step()**. This function calls another script named "postmd_steps.sh" with a single argument: "GROMACS_CYPWareData.txt".

postmd_steps.sh

The script named **postmd_steps.sh** performs the following commands:

Export environment variables from an input file.

Get the current working directory.

Source certain scripts: **welcome.sh**, **files.sh**, and **commands.sh**.

Print a welcome message (part 2).

Display the contents of the input file.

Check if a variable named **calculation_type** has been set. If not, exit with an error message.

Perform different actions based on the value of the **calculation_type** variable:

If the value is "apo", perform APO ET calculation using **apo_copy_files_spec_done**, **apo_distance_and_energy**, and **final_calculation** functions.

If the value is "bound", perform bound ET calculation using **bound_copy_files_spec_done**, **bound_distance_and_energy**, and **final_calculation** functions.

If the value is "both", perform both APO and bound ET calculations using the same functions mentioned in the previous two steps, in the order of bound and apo calculation.

If the value is anything else, exit with an error message.

ET_Cal.py

The **ET_Cal.py** script takes 4 input arguments from the command line, calculates several variables, and outputs them to a file with the provided filename.

Here are the steps in detail:

Import the **math** and **sys** modules:

Parse the input arguments from the command line using the **sys.argv** list:

The **float()** function is used to convert the input arguments from strings to floating-point numbers.

Print the filename with a **.txt** extension:

Calculate the values of **Hab** and **hab2** using the input arguments:

math.exp() is used to calculate the exponential function e^x.

Print the values of **Hab** and **hab2** with 10 decimal places:

The **{:.10f}** syntax formats the floating-point number with 10 decimal places.

Calculate the value of **K** using the variables **term1** to **term4**:

Calculate the natural logarithm of **K**:

Print the values of **lamda**, **delta_G**, **term1** to **term4**, **K**, and **log_num**:

Write the variable names and values to a file with the provided filename.