User and Developer Manual for:
Automated QM/MM Average Protein Electrostatic Configuration Approach for Flavoproteins [APEC-F 2.0]

Latest update: April 28th, 2025

1. Introduction

Welcome to the user manual for [Automated QM/MM Average Protein Electrostatic Configuration Approach for Flavoproteins: APEC-F 2.0], a tool designed to automatically model FMN and FAD-binding flavoproteins in any redox state and to account for a fully flexible solvated protein model. The first part of this manual (Sections 2 and 3) provides step-by-step instructions to install and use APEC-F 2.0. The next section (Section 4) provides more information about the code architecture for developers or users interested in making minor adjustments to the protocol. Lastly, section 5 is the troubleshooting section. For more information about the APEC approach and background, please check the associated manuscript (in preparation).

1.1 Purpose

The objective of this version of the APEC code, [APEC-F 2.0], is to deliver a fully automated approach that requires minimal user intervention while offering an updated codebase compatible with the latest open-source software versions at the time of publishing this manual. This includes seamlessly integrating APEC-FEG with current versions of OpenMolcas, Tinker, and GROMACS. By leveraging advancements in parallelization within OpenMolcas and the high-performance GPU-accelerated algorithms in GROMACS, [APEC-F 2.0] achieves a substantial improvement in computational efficiency and performance.

2. Getting Started

2.1 System Requirements

- **Operating System:** This version of APEC-F 2.0 was tested on both Ubuntu versions 20.04.6 and 22.04. It should be compatible with most modern Linux operating systems running bash, python3, and with gcc compilers available.

- **The following dependencies will be automatically installed by the setup script:**
  - `termcolor` Python package.
  - `matplotlib` Python package.
  - `SLURM` (Simple Linux Utility for Resource Management).
  - `git` (Version Control System).
- **The following open-source dependencies should be installed ahead of time by the user:**
  - `OpenMolcas and Tinker` for QM/MM calculations. Two scripts (OpenMolcas_ASEC_Install.sh and get_tinker_Openmolcas) are made available as part of the APEC-F 2.0 github repository to aid in installing those programs and to carry out a few important modifications to the source code.
    Link to repository.
  - `GROMACS` for MD calculations: Link to GROMACS
  - `Dowser` for model setup routine.Link to Dowser.
  - `VMD` for residue selection. Link to VMD.
  - `SCWRL4` for mutations: Link to SCWRL4.
- **The user should modify the following SLURM submission scripts found inside `camino` to run GROMACS and Molcas jobs on their cluster:**
  - `gromacs.slurm.sh`
  - `gromacs.slurm_GPU.sh`
  - `molcas.slurm.sh`

2.2 Quick Start

A video tutorial outlining the steps in this section is available on YouTube.
The written instructions are included below:

The `setup.sh` script can be used to install the code and set up a python virtual environment. This script downloads the full set of [APEC-F 2.0] scripts and installs the python dependecies listed above. `setup.sh` should be obtained from the following link.

The `setup.sh` script performs the following tasks:

- Downloads the code repository from Git.
- Installs `virtualenv` if it is not already installed.
- Creates a Python virtual environment.
- Installs the required Python packages within the virtual environment.

Follow these steps to set up a new APEC job:

1. Navigate to the directory containing the `setup.sh` script on GitHub.

2. Run the `setup.sh` script:

```
bash setup.sh
or:
./setup.sh
```

3. Activate the python virtual environment APEC_$USER$_timestamp:

```
cd APEC_$USER_$timestamp/bin
source activate
```

4. If the following folders are found: logs, processes, or databases please remove them:

```
rm -r logs/ processes/ databases/
```

5. To request mutations you need to create a file called "seqmut" and request the desired mutations: e.g Q430E. You can request as many mutations as you want in one sinlge run, or pick one mutation at a time.

```
nano seqmut
Q430E
```

By following the steps above, you are ready to run APEC-F 2.0. Two sample files have been provided for you to test. They are `1j8q.pdb` and `CHR_chain.xyz`. Simply run the following command to start executing the 5 Steps of APEC. Running the script in the background is recommended since the full execution is time-consuming, ensuring that the terminal remains available for other operations:

```
bash main.sh &
or:
./main.sh &
```

3. Usage

3.1 Input Example

The input is a parameters file having the following format (hereby we present an example).

project_name NSQ-flavo

chromo_FF_name CHR

oxidation_state Neutral-Semiquinone

pdb_file 1j8q.pdb

charge_chromophore -2

flavin_model Neutral-Semiquinone

tail FMN

box_shape cubic

edge_dist 2

size_cubicbox null

conc_NaCl 0.1

SOL y

NPT_production_temperature 298

NPT_heating_phase_time 300

equilibration_phase 79700

NPT_production_phase_time 0

GPU_use y

NVT_production_temperature 298

NVT_heating_phase 300

NVT_equilibrationphase1 9700

NVT_production_phase 10000

NVT_parallelize_production_phase n

number_parallelMD 1

NVT_GPUuse y

3.2 Parameters File Variables

The parameters file should be placed in your working directory at the same level as `New_APEC.sh`, `main.sh`, `first_MD_calculations.sh`, `first_QMMM_calculations.sh`, `nth_MD_calculations.sh`, `nth_QMMM_calculations.sh`, `init_db.py`, `update_status.py`, and `display_status.py`. It is crucial to preserve the variable names exactly as provided, as they are read by the scripts throughout the workflow. You only need to input the values or make specific selections based on your protein case. Below is a detailed explanation of each variable:

- `project_name`**:** Assign a meaningful name to the project, **ensuring it is no longer than 22 characters**. The project name should not start with a number.
- `chromo_FF_name`**:** name of the chromophore residue according to how you named it in your force field files. For instance, in flavodoxins, flavin is labeled 'CHR' (no matter what the redox state).
- `oxidation_state`**:** Specify the redox state of interest for the calculations. The available options are:

1. `Quinone`
2. `Anionic-Semiquinone`
3. `Neutral-Semiquinone`
4. `Anionic-Hydroquinone`
5. `Neutral-Hydroquinone`

For example, to run calculations for the quinone state, write `oxidation_state Quinone` (leave only one space between the variable name and the input value). Ensure the spelling matches the options exactly, including capitalization, as variations like `ASQ` or `Anionic-semiquinone` will cause the code to fail.

- **`pdb_file`:** Enter the name of your pdb file exactly as it appears in your folder. For example, "1j8q.pdb". The pdb file with the same name should be located in the same directory of your parameters file.

- **`flavin_model`:** Specify the flavin model corresponding to the chosen redox state. For this current implementation of the code, this parameter should be kept identical to what is entered for `oxidation_state`.

- **`charge_chromophore`:** Define the charge of the chromophore based on the oxidation state. For example, a Quinone, Neutral-Semiquinone, Neutral-Hydroquinone have a charge of `-2` while Anionic-Semiquinone and Anionic-Hydroquinone have a charge of `-3`.

- **`tail`:** For this parameter, you are specifying whether the "tail" moiety attached to the isoalloxazine ring corresponds to *FMN* or *FAD*. Simply input either `FMN` or `FAD` here.

- **`size_cubicbox`:** Please enter any random value here, as it will be self-populated after obtaining a volume equilibrated with NPT dynamics. Do not leave this parameter empty. We suggest using "null."

- **`box_shape`:** cubic. Given that the workflow needs to compute a final box side before NVT steps, the current APEC version has been developed for globular solutes (*e.g.*: flavoproteins), where it is straightforward to compute. This parameter is compulsory cubic for now.

- **`edge_dist`:** default value is 2 (nm). Sets the distance of the edges of the solute from the box facets in the initial GROMACS structure. This distance should be larger than the intermolecular interaction cutoffs (either electrostatics, set by `rcoulomb` in the mdp file, or van der Waals, set by `rvdw`).

- **`conc_NaCl`:** Specify the additional molar concentration of both sodium (Na^+) and chloride (Cl^-) ions to provide the simulation box with a desired ionic strength. If willing to avoid this addition, set it to 0 (do not write 0.0, nor "null" values). A value of `0` indicates that only a minimal number of counterions are used to neutralize the system, with no additional ions. For example, if your protein has a net charge of -20, the simulation will automatically add 20 Na^+ ions to achieve neutrality. The minimal number of counterions required to neutralize the system are always added but are not counted in this parameter value. Instead, specifying a value of

0.1 means that, in addition to the 20 Na^+ ions required to neutralize a -20 charged protein, an extra 0.1 M [Na^+] and 0.1 M [Cl^-] will be added to the system, creating a net balance and representing the salt concentration. The total number of ions added can be found in the Infos.dat file inside the Step folders as "Added_NAs" and "Added_CLs" .

- `SOL:` Indicate whether the system is solvated (`y`) or not (`n`). Currently, APEC-F 2.0 allows only solvated systems.

- `NPT_production_temperature:` Define the target Kelvin temperature for the NPT equilibration phase (constant number of molecules, pressure, and temperature). Default: `298` K (room temperature).

- `NPT_heating_phase_time:` ps heating time for the preparatory NPT stage, currently set to 300 ps. The system will be heated from the 0 K MM minimized structure to the desired production temperature.

- `equilibration_phase:` ps equilibration time after the previous heating time. Currently set to 79700 ps.

- `NPT_production_phase_time:` set to 0 (ps). If willing to make APEC based on an NPT workflow, this parameter will set the production time to be sampled from the following QM/MM phase of the step. At the moment, this degree of freedom is not to be exploited, as the frozen chromophore setting is not compatible with an NPT ensemble; we listed this parameter in case of new future developments.

- `GPU_use:` Indicates whether NPT molecular dynamics is run on GPU. Default: "y".

- `NVT_production_temperature:` Set the target Kelvin temperature for the NVT production phase (constant number of molecules , volume, and temperature). Default: `298` K.

- `NVT_heating_phase:` ps heating time of all NVT productions, currently set to 300 ps. In each of Steps 0, 1, 2, 3, 4, 5, etc., the system will be heated from the 0 K MM minimized structure to the desired production temperature (default: 298 K). MD velocities are not transferred between steps, so at the start of each step the system is again reheated to gradually anneal it from 0 K to room temperature, ensuring the chromophore and surrounding environment are properly equilibrated before initiating dynamics.

- `NVT_equilibrationphase1:` ps equilibration time after previous heating. Currently set to 9700 ps. This yields a total of 10000 ps equilibration splitted into 300 ps heating and 9700 ps equilibration.

- `NVT_production_phase:` Total production time in NVT trajectories. Conformations evolved during this time span will be sampled to provide the general MM embedding of the QM moiety during the following optimization step. Currently set to 10000 ps.

- `NVT_parallelize_production_phase:` Indicate if willing to perform two parallel, independent NVT trajectories (y), or a single, longer one (n). Currently set to "n"

due to the relatively short dynamics. At this stage, RMSD plots are printed in the output of the NVT simulation. By default, the y-values are set to ax.set_ylim(0,0.3), which usually is sufficient to capture RMSDs for smaller flavoproteins. The RMSD plots can be re-generated by the user, who can modify the ax.set_ylim variable and replot the .png image using python3 plot_RMSDs.py script.

- `number_parallelMD`: This variable is read if `NVT_parallelize_production_phase` is set to "y". Otherwise, it should be set to "n".
- `NVT_GPUuse`: Indicates whether NVT molecular dynamics is run on GPU. GPUs are useful to sped up calculations if available. Default: "y".

4. Code Architecture

4.1 A note on changes to the MD protocol in APEC-F 2.0

Hereby we discuss the main MD edits with respect to the APEC versions used in prior publications.

The main change is that APEC-F 2.0 now only uses an 80 ns NPT simulation once at the start of the protocol to determine the solvent box size, and then all simulations are run using NVT throughout the multi-step process. At the NPT stage, the chromophore is kept flexible, as the pressure coupling implies a coordinate rescaling algorithm that is not compatible with freezing any part of the system. As a consequence, the very last minimization step prior to the NPT trajectory is with flexible chromophore, too. In the updated protocol, we determine the volume of the box using the "edge_dist" variable which determines the distance from the edge of the box to the protein (by default, 2 nm).

```
$gropath/gmx editconf -f $Project.gro -bt $box_shape -d
   $edge_dist -o ${Project}_box_init.gro -c
```

After the preparative Sim_NPT folder step, one gets equilibrated volume and plugs it without the -d 2 editconf flag, but this time using the -box option with the updated size of the box that will be calculated from the output of NPT simulation and updated to the given size_cubicbox parameter initially starting with a null value.

```
$gropath/gmx editconf -f $Project.gro -bt $box_shape -box
    $box $box $box -o ${Project}_box_init.gro -c
```

With this approach, we set the box size based on the size of the system and leave a 2 nm buffer region so that the solute does not get close enough to the edge to interact with its PME image.

During NVT simulations and throughout almost all the APEC machinery, the chromophore is kept frozen using the `freezegrps` option. The only exception is during the NPT step and prior minimization steps, where the `freezegrps` option in the `dynamic_sol_NPT.mdp` file is commented out using semicolons (";;"), the "Solvent_box.sh" script includes the following lines that control the freezing option:

```
if [[ $initial == "YES" ]]; then
sed -i "s/freezegrps = GroupDyna/;freezegrps = GroupDyna/
   g" min_sol.mdp
sed -i "s/freezedim = Y Y Y/;freezedim = Y Y Y/g" min_sol
   .mdp
```

```
fi
```

The "Initial" variable is initialized to "YES" in the Infos.dat text file present in the camino prior to the start of the APEC workflow. The variable is then automatically switched to NO after the NPT trajectory. We modify the value of the "Initial" parameter in the Infos.dat in the first_MD_calculations.sh script and set it to NO. The last few lines of Solvent_box.sh script indicates whether the next script to be run would be MD_NPT.sh or MD_NVT.sh based on value of variable Initial in the Infos.dat.

```
if [[ $initial == "YES" ]]; then
  cp $templatedir/ASEC/MD_NPT.sh .
  ./update_infos.sh "Next_script" "MD_NPT.sh" Infos.dat
  ./update_infos.sh "MD_ensemble" "NVT" Infos.dat
else
  cp $templatedir/ASEC/MD_NVT.sh .
  ./update_infos.sh "Next_script" "MD_NVT.sh" Infos.dat
  ./update_infos.sh "MD_ensemble" "NVT" Infos.dat
fi
```

This means that running Step_0 for the first time (using initial_MD_calculations.sh), the MD_NPT.sh will be called after Solvent_box.sh and then Step_0 will be renamed to initial_Step_0. After the box size has been determined by the NPT run, the real Step_0 is initiated and repeats minimization and dynamics using the NVT ensemble and keeping the chromophore frozen. We change the value of the Initial variable in the Infos.dat from YES to NO only once in the first_MD_calculations.sh before running Solvent_box.sh, using the following command:

```
sed -i 's/^Initial.*/Initial NO/' Infos.dat
```

We don't need to worry about later Steps as this change of Initial variable value to NO will persist across all Steps, as the Infos.dat will be copied from Step_x to Step_x+1.

Another change is that we now use a time step of 2 fs (old value: 1 fs), with concomitant plug of LINCS H-bonds constraints. Plugging the latter is important when adopting a time step of 2 fs, as any bond with an H atom usually oscillates too rapidly to be properly described with the larger timestep. In practice, the LINCS algorithm is the equivalent of slowing down the stretching of bonds involving an H in a way that does not undermine the reliability of the oscillation description.

With the increase of the timestep from 1 to 2 fs, sampling time is approximately doubled at the same computational cost. There are two advantages of this approach with respect to the old version of APEC. First, we start sampling later (e.g., after 10 ns instead of 5 ns), meaning that we have a higher chance to start the production simulation from a well-equilibrated structure. Second, we are likely to sample more uncorrelated frames (as they are spaced by 100 ps not 50 ps anymore), thus allowing the system to evolve for a longer time. In the past, we typically run 2 replicas of 7.5 ns NVT productions with a 1 fs time step (15 ns in total); instead, we now run a single 20 ns trajectory with a 2 fs time step. Therefore, now the NVT phase is globally even less computationally demanding.

4.2 Code overview

The automated code design is organized into the following scripts:

- **main.sh**: The primary script responsible for executing all the following scripts in sequence, from Steps 0 through the last one. Based on our experience on several classes of small flavoproteins tested so far, we estimate 5 steps of the APEC cycle being generally sufficient to yield a converged structure and spectroscopic properties. By default, 5 steps are requested by the main.sh script, but this can be modified by the user. If needed to change the requested flow of iterative steps, one can simply change the sequence in the main.sh specifically in this line:

  ```
  for step in $(seq 1 5); do
  ```

  Depending on the intended usage of APEC-F 2.0, we advise the user to run multiple steps of APEC and test the result for their property or energy of interest. By doing so, the user checks for convergence and obtains statistical information about the error / sensitivity of their calculations to different APEC configurations.

  - **initial_MD_calculations.sh**: Executes Step_0 using NPT to determine the volume of the system and updates the box size in the `parameters` file. This is the only step where NPT simulations are used. All subsequent steps will use NVT-based dynamics simulation while keeping the flavin geometry frozen.
  - **first_MD_calculations.sh**: Runs the workflow of the Molecular Dynamics (MD) calculations specifically for Step_0 .
  - **first_QMMM_calculations.sh**: Executes the Quantum Mechanics/Molecular Mechanics (QM/MM) calculations for Step_0 .
  - **nth_MD_calculations.sh**: Performs MD calculations for Steps 1 through 5.
  - **nth_QMMM_calculations.sh**: Runs the QM/MM calculations for Steps 1 through 5.

- **init_db.py**: Each of the MD scripts above also automatically runs the init_db.py script, which initializes the database of script names and prepares a table that will be used to monitor status, run-times, and number of retries triggered for each script. A new database is generated at each step (0 to 5) as soon as that step starts. The database files are stored in the same folder inside of `/databases` as APEC_0.db for Step 0, APEC_1.db for Step 1, etc.

- **update_status.py**: This script is called automatically at the start and end of each inner script (e.g., New_APEC.sh, NewStep.sh, Solvent_box.sh, MD_NVT.sh, ASEC.sh, etc.) which are automated to be run through first_MD_calculations.sh, first_QMMM_calculations.sh, 1_MD_calculations.sh, etc, to update the execution status of each script. The status options include:

  - **NOT_RUN**: Displayed in black, indicating the script has not been executed.
  - **RUNNING**: Displayed in yellow, indicating the script is currently executing.
  - **PASSED**: Displayed in green, indicating successful execution.
  - **FAILED**: Displayed in red, indicating an error occurred during execution.

- **display_status.py**: This script can be run by the user at any time to print the live status. It prints the execution status and run times of all scripts. It is primarily used to track the progress of the calculations and to get an idea about timings. To

monitor the flow of scripts, including each script's status and which step you are currently at, please run `python3 display_status.py`. You can also create an alias and add it to the `.bashrc` file by following these steps:

1. Open the `.bashrc` file:

   ```
   vi ~/.bashrc
   ```

2. Add the following line to create an alias:

   ```
   alias display='python3 display_status.py'
   ```

3. Save the file and reload the `.bashrc` file:

   ```
   source ~/.bashrc
   ```

Following these steps will enable you to monitor the status using the command `display` instead of running `python3 display_status.py` each time. Please recall that you should run these scripts in the working directory where the `display_status.py` script resides.

- **processes.sh**: In case the user wishes to terminate an APEC run prematurely before the last step, they should use the `processes.sh` script. This script identifies and kills shell script processes running in the current directory for a specific user. It performs the following steps:

  1. Retrieves the current directory and the first 6 characters of the username.
  2. Lists the PIDs of shell scripts (`.sh`) running under the user's name, excluding the `processes.sh` script itself.
  3. Defines a function `check_and_kill`, which checks if a process is running in the current directory and kills it if true.
  4. Iterates through each PID and calls the `check_and_kill` function to terminate processes in the current directory.
  5. Prints a confirmation message once all relevant processes are killed.

  To run the script, execute `processes.sh` in the terminal. This will kill all processes running shell scripts in the current directory for the logged-in user. It should be used when restarting a project under the same directory.

- **APEC_SetupPhoton.sh**: This script prepares a PDB file for QM/MM calculations. It includes the following steps:

  1. **Input and Cleaning:** The script prompts the user to specify the PDB file to process. It filters the file to retain only the `ATOM`, `TER`, and `HETATM` records, creating a cleaned version of the PDB.
  2. **Chain Identification:** The script identifies and stores all unique chain identifiers present in the cleaned PDB file. APEC-F 2.0 is currently only able to treat monomeric systems. If multiple chains are present, the user is prompted to select one. The selected chain is extracted into a separate monomer PDB file.

3. **Chromophore Extraction:** The script identifies the chromophore (e.g., FMN or FAD) in the monomer file. It extracts the coordinates and atom labels into an XYZ file while also creating a PDB file that excludes the chromophore for subsequent QM/MM preparation.
4. **Oxidation State Setup:** The user specifies the oxidation state of the chromophore. Based on this, the script prepares molecular files (e.g., `mol2` and `xyz`) with appropriate connectivities and hydrogenation levels.

4.3 Step Overview

The code execution process is structured around three main directories that facilitate tracking, database management, and logging for each step of the workflow:

- **Processes Folder:**
  - Stores process IDs (`PID`) for the `main.sh` script.
  - Each process ID corresponds to a specific script, such as:
    - `first_MD_calculations.pid` for `first_MD_calculations.sh`, which runs the MD calculations for Step 0.
    - `1_MD_calculations.pid` for `1_MD_calculations.sh`, which runs the MD calculations for Step 1.
    - `1_QMMM_calculations.pid` for `1_QMMM_calculations.sh`, which runs the QMMM calculations for Step 1.
    - `5_MD_calculations.pid` for `5_MD_calculations.sh`, which runs the MD calculations for Step 5.
    - `5_QMMM_calculations.pid` for `5_QMMM_calculations.sh`, which runs the QMMM calculations for Step 5.
  - These files allow the system to monitor script execution and ensure a step is complete before proceeding.
- **Databases Folder:**
  - Contains SQLite databases for managing the execution status of each step.
  - Each database stores detailed information for its respective step, including:
    - Start and end times of the execution.
    - Status indicators (`NOT_RUN`, `RUNNING`, `PASSED`, `FAILED`).
    - Other relevant data associated with the step.
- **Logs Folder:**
  - Contains log files for each workflow step and its associated scripts.
  - Logs track `success_messages` from each script.
  - A script is marked as successful only if it produces the expected success message in its log file.
  - If a success message is missing, the script is marked as `FAILED`, and the workflow halts before proceeding to the next step.

4.4 Individual Script Handling

Function Explanations

1. Polling Script Execution: Two Implementations of `polling_loop`

The `polling_loop` function is used to monitor the execution of scripts and ensure their completion by periodically checking for specific success criteria. Two distinct implementations are provided below:

1. **First Implementation: Basic Polling Loop**

   - This implementation monitors the completion of a script by checking for predefined success messages in output files.
   - The function starts by running the script and initializes a timer to track the polling period.
   - It loops until either all success conditions are met or the timeout period is reached.
   - Within the loop:
     - The function iterates through each success message and corresponding file.
     - If all success messages are found in their respective files, the script is marked as passed.
   - If the timeout period is reached without meeting the success conditions, the script is marked as failed.

   ```
   polling_loop "MD_NVT.sh" NVT_success_messages[@]
      NVT_files_to_check[@]
   ```

   This implementation is primarily used for scripts where multiple conditions must be validated across multiple files.

2. **Second Implementation: SLURM Job Monitoring**

   - This version monitors jobs submitted to a SLURM scheduler using the `squeue` command.
   - The function starts the script and retrieves its job ID from a specified file.
   - A loop periodically checks the SLURM job queue for the presence of the job ID associated with the script.
     - If the job ID is no longer found, the function checks for success messages in a specified output file to determine if the script has passed.
     - If the success message is found, the script is marked as passed; otherwise, it is marked as failed.
   - If the job ID remains in the queue beyond the timeout period, the script is marked as failed.

   ```
   polling_loop "script_name" "jobid_file" "output_file"
       "success_message"
   ```

   This implementation is used for scripts involving SLURM job scheduling and requires real-time job monitoring.

Comparison of the Implementations

- The first implementation focuses on general-purpose file-based polling, making it suitable for workflows where multiple conditions are checked across different files. (for example MD_NVT.sh)
- The second implementation is specifically tailored for SLURM-based job monitoring, integrating job ID tracking with success validation from output files.

2. restart_after_5_days Function

The `restart_after_5_days` function is designed to handle long-running computational tasks (optimizations in particular) that may require restarting due to timeouts. Below is a breakdown of the key components:

Function Overview:

- `extension` (argument 1): Specifies the calculation's file extension or identifier (e.g., `VDZP_Opt`).
- `script` (argument 2): The name of the script (e.g., `ASEC.sh`) that is being executed.

Steps:

1. The function starts by setting the status of the script to `RUNNING` using `update_status.py`.
2. It then changes to the directory `$Project_{extension}` where the calculations are stored.
3. A `try_counter` is initialized to 0, and the script enters a loop to retry the task up to 5 times. Each attempt is triggered every 5 days. If an optimization is not completed within 5 days, the function is triggered to complete the optimization and restart from where it stopped. A `try_0` folder is created after the first 5 days, `try_1` is created after 10 days of optimization, and `try_2` is created after 15 days, etc .. The total number of allowed restarts is limited to 5 to prevent the script from entering an infinite loop.
4. For each attempt, it:
   - Creates a new directory `try_0, try_1, etc.` to organize the results of each try.
   - Moves the files from the current directory into the new try directory.
   - Copies the latest `xyz` file and other necessary files like `amber99sb.prm`, `molcas-job.sh`, and others into the new try folder.
   - Submits the job with `sbatch molcas-job.sh` and stores the job ID.
5. After submission, it enters a while loop to monitor the job status using the `squeue` command. If the job is running, it waits and checks periodically.
6. If the job is not found in the queue, the function waits for a few minutes before checking again.

7. If the job finishes, it checks the output file (`$Project_{extension}.out`) for the phrase `Happy landing!`, which indicates a successful completion of a Molcas job.

8. If the output is not successful, the function moves on to the next try.

9. If the job reaches the timeout limit (5 trys), the function breaks the loop and exits with an error message.

Usage in ASEC.sh: In the `ASEC.sh` script, the `restart_after_5_days` function is used after the completion of the `polling_loop` function. If the job completes successfully and the output file contains `Happy landing!`, the script marks the job as `PASSED` using `update_status.py`. Otherwise, if the job has not successfully completed, it calls the `restart_after_5_days` function to restart the task, passing the necessary arguments for the extension and script name.

Example usage in ASEC.sh:

```
polling_loop "ASEC.sh" "$CURRENT_DIR/Step_0/calculations/\${
   Project}_VDZP_Opt/jobid"

if [ -f "\${Project}_VDZP_Opt/\${Project}_VDZP_Opt.out" ];
   then
    if grep -iq "Happy landing!" "$CURRENT_DIR/Step_0/
       calculations/\${Project}_VDZP_Opt/\${Project}_VDZP_Opt
       .out"; then
        python "$CURRENT_DIR/update_status.py" "ASEC.sh"
           PASSED $CURRENT_DIR
    else
        restart_after_5_days VDZP_Opt ASEC.sh
    fi
else
    restart_after_5_days VDZP_Opt ASEC.sh
fi
```

3. `check_value_within_tolerance`

This function validates whether a given value (e.g., system total charge) lies within a specified tolerance range.

- **Validation:** If the value is zero, the status is `PASSED`. Otherwise, it checks if the value lies within the range `[-tolerance, +tolerance]`. By default, we have set the tolerance to ± 0.005 charge.
- **Failure Handling:** If out of range, an error message indicates whether the value is too high or too low, and the script status is set to `FAILED`.

Example: A system total charge of 0.006 exceeds a tolerance of 0.005, causing the script to fail.

4. `run_script`

This function is meant for scripts that do not use SLURM to submit jobs and therefore does not require monitoring. The function executes a given script, monitors its output, and validates its success based on a predefined success message.

- **Input Parameters:**
  - `script`: The name of the script to execute.
  - `success_msg`: The message indicating successful execution of the script.
- **Execution Process:**
  1. Updates the status of the script to `RUNNING`.
  2. Executes the script and logs its output to a specified log file.
  3. Searches the log file for the success message:
     - If found, updates the status to `PASSED` and exits successfully.
     - If not found, retries the script execution up to two times, updating the status to `FAILED` after each failed attempt.

  1. **Example 1: Executing `New_APEC.sh`**
     - The script `New_APEC.sh` is executed using `run_script`. The success message expected in the log file is:
       `"Folder Step_0 created! cd Step_0/, then ./NewStep.sh"`
     - If the script runs successfully, the directory changes to `Step_0` using `cd Step_0`.
     - If the success message is not found after retries, the workflow exits with a failure code.
     - Command:

```
run_script "New_APEC.sh" "Folder Step_0
   created! cd Step_0/, then ./NewStep.sh"
   && cd Step_0 || exit 1
```

  2. **Example 2: Running `NewStep.sh`**
     - The next script, `NewStep.sh`, is executed with the success message:
       `"Continue executing: Solvent_box.sh"`

- **Failure Handling:** If the success message is still not found after two retries, the function exits with a failure code.

5. Additional Logic

- **Temperature Validation:** The averaged NVT temperature is compared to the production temperature. It passes if the temperature is less than the target or within 10 degrees from it.
- **Error Handling:** Timeout and validation checks provide feedback to ensure robustness and traceability.

5. Troubleshooting

5.1 Where to look

APEC-F 2.0 is meant to facilitate and automate the construction of QM/MM models efficiently. However, we strongly advise all users and developers to check, at least when running the code for the first time on a new system, that things are going as expected.

The first place to check on the progress of the calculations or look for potential errors is in the `logs` folder. There, you will find the output of both the main wrappers (`initial_MD_calculations.log`,`first_MD_calculations.log`, `first_QMMM_calculations.log`, `1_MD_calculations.log`, etc.) as well as for each of the individual scripts inside of the Step folders. It is useful to go through those files and check for errors.

Next, it is recommended that the user check the results of the MD in Step_0 or Step_1. Specifically, inside each step there is a Dynamic/output that includes an `md.log` file, a `$Project_box_sol.ndx` file, and scripts to compute and plot RMSDs. It is advisable that users check all of those files. The `ndx` file can be visualized along with the corresponding `$Project_box_sol.gro` using vmd. With the exception of jobs run inside `initial-Step_0`, visualization of any other MD simulations should reveal that the flavin is frozen but bound to the protein binding pocket where it is expected to be.

Also inside Step_0 or Step_1 should be another folder named `calculations`. It is recommended that users check on progress of the QM/MM geometry optimization which occurs inside the folder `$Project_VDZP_Opt`. There, the `$Project_VDZP_Opt.out`, `$Project_VDZP_Opt.err`, and `$Project_VDZP_Opt.Tinker.log` can all be good files to check for debugging purposes. The `.molden` files there can be viewed with the Molden visualization package to check the progress of the flavin geometry optimization and molecular orbitals.

Between these three folders (`logs, Dynamics`, and `calculations`), the user can follow the progression of the calculation and will be able to catch issues early if they occur.

5.2 Some common issues

Below is a discussion of some of the most common issues and how to troubleshoot them.

- **Error: Missing Dependency**
  - **Solution:** Run `pip install -r requirements.txt` to ensure all dependencies are installed. Also ensure that GROMACS, OpenMolcas, Tinker, and a SLURM cluster are installed and properly configured before starting the project.
- **Error: Virtual Environment Not Started**
  - **Solution:** Activate the virtual environment by running `source env/bin/activate`.
- **Error: Wrong Python Version**
  - **Solution:** Ensure that Python 3.9 or above is installed. To verify your Python version, run: `python3 --version`.

- **Error: File input/output error or mkdir: cannot create directory - Remote I/O error**
  - **Solution:** I/O errors are often associated with a full hard drive. In such cases, it is useful to check the trajectory using Gromacs:

    ```
    gmx check -f your_trajectory.trr
    ```

    If the job ID is present, this means the trajectory has been generated and copied back, but the simulation might not have completed the entire run (e.g., it may have finished only 20 ns instead of the requested 80 ns). This issue can arise due to disk memory problems, being over quota, or having a full disk. If you would like to reduce the size of the MD trajectory output, you can reduce the print frequency by adjusting the following parameters in the MDP files:

    ```
    nstxout = 5000
    nstvout = 5000
    nstfout = 5000
    ```

    to

    ```
    nstxout = 500000
    nstvout = 0
    nstfout = 0
    ```

    This would be useful to be done at level of dynamic_sol_NPT.sh, for the 80 ns NPT run, as we are mainly concerned with the volume and we don't really need structural data. For best practices, monitor your disk usage and clear any old files you no longer need. It may also be useful to delete large files generated by the Molcas portion of the code; pay particular attention to scratch files you may not need, such as OrdInt or NqGrid files from Molcas. You can also save space just by removing the "grid_it" module from Molcas input files, unless you need it. To check the sizes of directories or files, use:

    ```
    du -sh *
    ```

    You can also sort the results to identify large folders using:

    ```
    du -sh * | sort -h
    ```

    To remove .trr or .xtc files, use the following command:

    ```
    find . -name "*.trr*" -exec rm -f {} +
    find . -name "*.xtc*" -exec rm -f {} +
    ```

    Be careful with these commands, as a typo (e.g., a space between the star and the extension) could delete all your files.

    If an error occurs early in the APEC-F 2.0 run, it is recommended to remove Step_0, logs, processes, and database folders and run a fresh simulation with:

    ```
    ./main.sh &
    ```

    If instead the error occurs at a later step, e.g., the MD steps of Step_2, you can manually remove Step_2, go to Step_1, and run:

    ```
    ./Next_Iteration.sh
    ```

Also, clear the APEC2.db file from the database folder, any "2_MD_calculations.sh" files from the logs folder, and any "2_QMMM_calculations.sh" files from the logs folder. Additionally, remove any "2*.pid" files from the processes folder.

```
rm -r Step_2
cd logs
rm -r Step_2
rm 2*
cd processes
rm 2*
cd databases
rm APEC_2.db
```

When doing this, ensure to comment out the following lines in the main.sh:

```
./initial_MD_calculations.sh >logs/
   initial_MD_calculations.log 2>&1 &
echo $! >processes/initial_MD_calculations.pid
wait_for_script "processes/initial_MD_calculations.
   pid"

mv logs/Step_0 logs/initial-Step_0

mkdir -p logs/Step_0
./first_MD_calculations.sh > logs/
   first_MD_calculations.log 2>&1 &
echo $! > processes/first_MD_calculations.pid
wait_for_script "processes/first_MD_calculations.pid"

./first_QMMM_calculations.sh > logs/
   first_QMMM_calculations.log 2>&1 &
echo $! > processes/first_QMMM_calculations.pid
wait_for_script "processes/first_QMMM_calculations.
   pid"
```

Also, modify the loop from:

```
for step in $(seq 1 5); do
```

to:

```
for step in $(seq 2 5); do
```

Also, modify the following:

```
rm -rf databases/*
rm -rf logs/*
rm -rf processes/*
```

to:

```
echo "rm -rf databases/*"
echo "rm -rf logs/*"
echo "rm -rf processes/*"
```

Then, run:

```
./main.sh &
```

This will allow the automated version to restart from Step_2.

- **Error: Invalid Input Format**
  - **Example:** As a user, you are not explicitly asked to enter any meaningful value for size_cubicbox parameter as this parameter will be self-populated after running NPT dynamics. However, it is compulsory to not keep this parameter empty and to enter any random value or character such as null. Otherwise, the parameter will not be populated and the dynamics will get stuck at level of Solvent_box.sh when running it again after the pre-initial APEC preparation (i.e, after storing the first Step_0 as initial-Step_0/ and running the actual Step_0 starting with a volume equilibrated with NPT in an NVT-frozen chromophore fashion.
  - **Solution:** Verify that your input file adheres to the expected format as detailed in Section 3.1 of the documentation.
- **Error: Display Status Not Returning Data**
  - **Solution:** Make sure to activate the virtual environment before running `./main.sh &`. After executing `./main.sh &`, run the `python3 display_status.py` command in the terminal to ensure that the status is displayed. If the following error appears: `sqlite3.OperationalError: no such table: APEC_STATUS`, make sure to:

```
rm -r Step_0
rm -r logs
rm -r databases
rm -r processes
./processes.sh
./main.sh &
```

- **Error: GROMACS and OpenMolcas jobs are not running after being submitted with sbatch**
  - **Solution:** The user should customize their SLURM scripts found inside the `camino` path. Specifically, the `gromacs.slurm.sh, gromacs.slurm_GPU.sh`, and `molcas.slurm.sh` scripts are called regularly by APEC-F 2.0 and should be configured to properly run GROMACS and OpenMolcas on the user's cluster. A scratch directory specified in those scripts should also be accessible and have sufficient space.

6. Contact Information

If you encounter any issues or have questions, please contact us using the details below:

- **Email:** sgozem@gsu.edu, selhajj1@student.gsu.edu
- **GitHub:** Submit an Issue on GitHub