qmflows-namd Documentation

Release 0.8.2

Felipe Zapata and Ivan Infante

Dec 17, 2019

Introduction

Contents:

CHAPTER 1

QMFlows-NAMD

QMFlows-NAMD is a generic python library with the purpose of computing (numerically) non-adiabatic coupling vectors (NACV) using several quantum chemical (QM) packages.

One of the main problems to calculate (numerically) NACVs by standard QM software is the computation of the overlap matrices between two electronically excited states at two consecutive time-steps that are needed in the numerical differentiation to evaluate the coupling. This happens because most of these softwares are inherently static, i.e. properties are computed for a given structural configuration, and the computation of the overlap matrices at different times requires complicated scripting tools to handle input/outputs of several QM packages.

For further information on the theory behind QMFlows-NAMD and how to use the program see the documentation.

1.1 Installation

In order to install the *QMFlows-NAMD* library you need to install first the **QMFlows** package and its environment using *Miniconda* as detailed here.

Then, to install the QMFlows-NAMD library type the following commands inside the conda environment:

- `conda install -c conda-forge rdkit cython eigen h5py libint==2.4.2 highfive pybind11`
- `pip install git+https://github.com/SCM-NV/qmflows-namd#egg=qmflows-namd --upgrade`

1.2 Advantages and Limitations

QMFlows-NAMD is based on the approximation that all excited states are represented by singly excited-state determinants. This means that the computation of the NACVs boils down to the computation of molecular orbitals (MOs) coefficients at given points of time using an electronic structure code and an overlap matrix S(t,t+dt) in atomic orbital basis (AO) computed between two consecutive time step. QMFlows-NAMD main advantage is to use an internal module to compute efficiently the atomic overlap matrix S(t, t+dt) by employing the same basis-set used in the electronic structure calculation. In this way the QM codes are only needed to retrieve the MOs coefficients at time t and t+dt. This approach is very useful because the interfacing QMFlows-NAMD to a QM code is reduced to writing a simple module that reads the MOs coefficients in the specific code format. At this moment, QMFlows-NAMD handles output formats generated by CP2K, Orca, and Gamess, but, as said, it can be easily extended to other codes.

Finally, QMFlows-NAMD can be also used in benchmarks studies to test new code developments in the field of excited state dynamics by providing a platform that uses all the functionalities of QMFlows, which automatizes the input preparation and execution of thousands of QM calculations.

In the near future, QMFlows-NAMD is expected to offer new functionalities.

1.3 Interface to Pyxaid

QMFlows-NAMD has been designed mostly to be integrated with Pyxaid, a python program that performs non-adiabatic molecular dynamic (NAMD) simulations using the classical path approximation (CPA). The CPA is based on the assumption that nuclear dynamics of the system remains unaffected by the dynamics of the electronic degrees of freedom. Hence, the electronic dynamics remains driven by the ground state nuclear dynamics. CPA is usually valid for extended materials or cluster materials of nanometric size.

In this framework, QMFlows-NAMD requires as input the coordinates of a pre-computed trajectory (at a lower level or at the same level of theory) in xyz format and the input parameters of the SCF code (HF and DFT). QMFlows-NAMD will then calculate the overlap matrix between different MOs by correcting their phase and will also track the nature of each state at the crossing seam using a min-cost algorithm . The NACVs are computed using the Hammes-Schiffer-Tully (HST) 2-point approximation and the recent Meek-Levine approach. The NACVs are then written in Pyxaid format for subsequent NAMD simulations.

1.4 Overview

The Library contains a **C++** interface to the libint2 library to compute the integrals and several numerical functions in Numpy. While the scripts are set of workflows to compute different properties using different approximations that can be tuned by the user.

1.4.1 Worflow to calculate Hamiltonians for nonadiabatic molecular simulations

The figure represents schematically a Worflow to compute the **Hamiltonians** that described the behavior and coupling between the excited state of a molecular system. These **Hamiltonians** are used by thy PYXAID simulation package to carry out nonadiabatic molecular dynamics.

`docs/_images/nac_worflow.png`

CHAPTER 2

Theory

2.1 Nonadiabatic coupling matrix

The current implementation of the nonadiabatic coupling is based on: Plasser, F.; Granucci, G.; Pittner, j.; Barbatti, M.; Persico, M.; Lischka. *Surface hopping dynamics using a locally diabatic formalism: Charge transfer in the ethylene dimer cation and excited state dynamics in the 2-pyridone dimer*. **J. Chem. Phys. 2012, 137, 22A514.**

The total time-dependent wave function $\Psi(\mathbf{R}, t)$ can be expressed in terms of a linear combination of `N` adiabatic electronic eigenstates $\phi_i(\mathbf{R}(t))$,

$$\Psi(\mathbf{R}, t) = \sum_{i=1}^{N} c_i(t)\phi_i(\mathbf{R}(t)) \quad (1)$$

The time-dependent coefficients are propagated according to

$$\frac{dc_j(t)}{dt} = -i\hbar^2 c_j(t)E_j(t) - \sum_{i=1}^{N} c_i(t)\sigma_{ji}(t) \quad (2)$$

where $E_j(t)$ is the energy of the jth adiabatic state and $\sigma_{ji}(t)$ the nonadiabatic matrix, which elements are given by the expression

$$\sigma_{ji}(t) = \langle\phi_j(\mathbf{R}(t)) \mid \frac{\partial}{\partial t} \mid \phi_i(\mathbf{R}(t))\rangle \quad (3)$$

that can be approximate using three consecutive molecular geometries

$$\sigma_{ji}(t) \approx \frac{1}{4\Delta t}(3\mathbf{S}ji(t) - 3\mathbf{S}ij(t) - \mathbf{S}ji(t - \Delta t) + \mathbf{S}ij(t - \Delta t)) \quad (4)$$

where $\mathbf{S}_{ji}(t)$ is the overlap matrix between two consecutive time steps

$$\mathbf{S}ij(t) = \langle\phi j(\mathbf{R}(t - \Delta t)) \mid \phi_i(\mathbf{R}(t))\rangle \quad (5)$$

and the overlap matrix is calculated in terms of atomic orbitals

$$\mathbf{S}ji(t) = \sum \mu C^* \mu i(t) \sum \nu C_{\nu j}(t - \Delta t)\mathbf{S}_{\mu\nu}(t) \quad (6)$$

Where :math:C_{mu i} are the Molecular orbital coefficients and $\mathbf{S}_{\mu\nu}$ The atomic orbitals overlaps.

$$\mathbf{S}\mu\nu(\mathbf{R}(t), \mathbf{R}(t-\Delta t)) = \langle \chi\mu(\mathbf{R}(t)) \mid \chi_{\nu}(\mathbf{R}(t-\Delta t) \rangle \quad (7)$$

2.2 Nonadiabatic coupling algorithm implementation

The figure belows shows schematically the workflow for calculating the Nonadiabatic coupling matrices from a molecular dynamic trajectory. The uppermost node represent a molecular dynamics trajectory that is subsequently divided in its components andfor each geometry the molecular orbitals are computed. These molecular orbitals are stored in a HDF5. binary file and subsequently calculations retrieve sets of three molecular orbitals that are use to calculate the nonadiabatic coupling matrix using equations **4** to **7**. These coupling matrices are them feed to the PYXAID package to carry out nonadiabatic molecular dynamics.

The Overlap between primitives are calculated using the Obara-Saika recursive scheme and has been implemented using the C++ libint2 library for efficiency reasons. The libint2 library uses either OpenMP or C++ threads to distribute the integrals among the available CPUs. Also, all the heavy numerical processing is carried out by the highly optimized functions in NumPy.

> The **nonadiabaticCoupling** package relies on *QMWorks* to run the Quantum mechanical simulations using the [CP2K](https://www.cp2k.org/) package. Also, the noodles is used to schedule expensive numerical computations that are required to calculate the nonadiabatic coupling matrix.

CHAPTER 3

Introduction to the Tutorials

The *qmflows-namd* packages offers the following set of workflows to compute different properties:

- derivative_coupling
- absorption_spectrum
- distribute_absorption_spectrum

3.1 Known Issues

3.1.1 Distribution of the workflow over multiple nodes

CP2K can uses multiple nodes to perform the computation of the molecular orbitals using the **MPI** protocol. Unfortunately, the *MPI* implementation for the computation of the *derivative coupling matrix* is experimental and unestable. The practical consequences of the aforemention issues, is that **the calculation of the coupling matrices are carried out in only 1 computational node**. It means that if you want ask for more than 1 node to compute the molecular orbitals with *CP2K*, once the workflow starts to compute the *derivative couplings* only 1 node will be used at a time and the rest will remain idle wating computational resources.

3.2 Reporting a bug or requesting a feature

To report an issue or request a new feature you can use the github issues tracker.

CHAPTER 4

Derivative coupling calculation

These tutorials focus on how to compute non-adiabatic coupling vectors between molecular orbitals belonging at two different time steps, t and t+dt, of a pre-computed molecular dynamics trajectory. What this program does is to compute at each point of the trajectory, the electronic structure using DFT, and then the overlap integrals $\langle\psi_i(t) \mid \psi_j(t+dt)>$. These integrals are stored and finally used to compute numerically the non-adiabatic couplings. These and the orbital energies are written in a format readable by PYXAID to perform surface hopping dynamics. When using this tutorial, ensure you have the latest version of QMFlows and QMFlows-NAMD installed.

4.1 Preparing the input

The following is an example of the inputfile for the calculation of derivative couplings for the Cd33Se33 system. The calculations are carried out with the CP2k package, which you need to have pre-installed. The level of theory is DFT/PBE.

```
workflow: distribute_derivative_couplings
project_name: Cd33Se33
dt: 1
active_space: [10, 10]
algorithm: "levine"
tracking: False
path_hdf5: "test/test_files/Cd33Se33.hdf5"
path_traj_xyz: "test/test_files/Cd33Se33_fivePoints.xyz"
scratch_path: "/tmp/namd"
workdir: "."
blocks: 5

job_scheduler:
  scheduler: SLURM
  nodes: 1
  tasks: 24
  wall_time: "24:00:00"
  load_modules: "source activate qmflows\nmodule load cp2k/3.0"
```

(continues on next page)

(continued from previous page)

```yaml


cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  path_basis: "test/test_files"
  cell_parameters: 28.0
  cell_angles: [90.0, 90.0, 90.0]

  cp2k_settings_main:
    specific:
      template: pbe_main

  cp2k_settings_guess:
    specific:
      template: pbe_guess
```

The previous input can be found at input_test_distribute_derivative_couplings.yml. Copy this file to a folder where you want start the QMFlows calculations.

The *input_test_distribute_derivative_couplings.yml* file contains all settings to perform the calculations and needs to be edited according to your system and preferences. Pay attention to the following parameters: *project_name, dt, active_space, algorithm, tracking, path_hdf5, path_traj_xyz, scratch_path, workdir, blocks*.

- **project_name**: Project name for the calculations. You can choose what you wish.
- **dt**: The size of the timestep used in your MD simulations.
- **active_space**: Range of *(occupied, virtual)* molecular orbitals to computed the derivate couplings. For example, if 50 occupied and 100 virtual should be considered in your calculations, the active space should be set to [50, 100].
- **algorithm**: Algorithm to calculate derivative couplings can be set to 'levine' or '3points'.
- **tracking**: If required, you can track each state over the whole trajectory. You can also disable this option.
- **path_hdf5**: Path where the hdf5 should be created / can be found. The hdf5 is the format used to store the molecular orbitals and other information.
- **path_traj_xyz**: Path to the pre-computed MD trajectory. It should be provided in xyz format.
- **scratch_path**: A scratch path is required to perform the calculations. For large systems, the .hdf5 files can become quite large (hundredths of GBs) and calculations are instead performed in the scratch workspace. The final results will also be stored here.
- **workdir**: This is the location where the logfile and the results will be written. Default setting is current directory.
- **blocks**: The number of blocks (chunks) is related to how the MD trajectory is split up. As typical trajectories are quite large (+- 5000 structures), it is convenient to split the trajectory up into multiple chunks so that several calculations can be performed simultaneously. Generally around 4-5 blocks is sufficient, depending on the length of the trajectory and the size of the system.
- **write_overlaps**: The overlap integrals are stored locally. This option is usually activated for debugging.
- **overlaps_deph**: The overlap integrals are computed between t=0 and all othe times: <psi_i (t=0) | psi_j (t + dt)>. This option is of interest to understand how long it takes to a molecular orbital to dephase from its starting configuration. This option is disabled by default.

The **job_scheduler** can be found below these parameters. Customize these settings according to the system and environment you are using to perform the calculations.

In the **cp2k_general_settings**, you can customize the settings used to generate the cp2k input. You can use the cp2k manual to create your custom input requirements. Remember to provide a path to the folder with the cp2k basis set anc potential files.

4.2 Setting up the calculation

Once all settings in *input_test_distribute_derivative_couplings.yml* have been customized, you will need to create the different chunks.

- First, activate QMFlows:

  ```
  conda activate qmflows
  ```

- Use the command *distribute_jobs.py* to create the different chunks:

  ```
  distribute_jobs.py -i input_test_distribute_derivative_couplings.yml
  ```

A number of new folders are created. In each folder you will find a launch.sh file, a chunk_xyz file and an input.yml file. In the input.yml file, you will find all your settings. Check for any possible manual errors.

- If you are satisfied with the inputs, submit each of your jobs for calculation.

You can keep track of the calculations by going to your scratch path. The location where all points of the chunks are calculated is your assigned scratch path plus project name plus a number.

The overlaps and couplings between each state will be calculated once the single point calculations are finished. The progress can be tracked with the .log file in your working directory folders. The calculated couplings are meaningless at this point and need to be removed and recalculated, more on that later.

4.3 Merging the chunks and recalculating the couplings

Once the overlaps and couplings are all calculated, you need to merge the different chunks into a single chunk, as the overlaps between the different chunks still need to be calculated. For this you will use the *mergeHDF5.py* command that you will have if you have installed QMFlows correctly.

You are free to choose your own HDF5 file name but for this tutorial we will use *chunk_01234.hdf5* as an example.

- Merge the different chunk into a single file using the *mergeHDF5.py* script:

  ```
  mergeHDF5.py -i chunk_0.hdf5 chunk_1.hdf5 chunk_2.hdf5 chunk_3.hdf5
  chunk_4.hdf5 -o chunk_01234.hdf5
  ```

Follow -i with the names of different chunks you want to merge and follow -o the name of the merged HDF5 file.

- Remove the couplings from the chunk_01234.hdf5 using the *removeHDF5folders.py* script. To run the script, use:

  ```
  removeHDF5folders.py -pn PROJECTNAME -HDF5 chunk_01234.hdf5
  ```

Replace PROJECTNAME with your project name.

Using the script in this manner will only allow the couplings to be removed.

Note:

If required, you can remove all overlaps by by adding -o at the end of the previous command:

```
removeHDF5folders.py -pn PROJECTNAME -hdf5 chunk_01234.hdf5 -o
```

- Create a new subfolder in your original working directory and copy the *input.yml* file that was created for chunk 0 (when running the *distribute_jobs.py* script) to this folder.
- Edit the *input.yml* file to include the path to the merged .hdf5, the full MD trajectory, and a new scratch path for the merged hdf5 calculations.
- Relaunch the calculation.

Once the remaining overlaps and the couplings have been calculated successfully, the hdf5 files and hamiltonians will be written to both the working directory as well as the scratch folder in a format suitable for PYXAID to run the non-adiabatic excited state molecular dynamics. If requested, also the overlap integrals can be found in the working directory.

Note: There are several way to declare the parameters of the unit cell, you can passed to the cell_parameters variable either a number, a list or a list or list. A single number represent a cubic box, while a list represent a parallelepiped and finally a list of list contains the ABC vectors describing the unit cell. Alternatively, you can pass the angles of the cell using the cell_angles variable.

4.4 Restarting a Job

Both the *molecular orbitals* and the *derivative couplings* for a given molecular dynamic trajectory are stored in a HDF5. The library check wether the *MO* orbitals or the coupling under consideration are already present in the HDF5 file, otherwise compute it. Therefore if the workflow computation fails due to a recoverable issue like:

- Cancelation due to time limit.
- Manual suspension or cancelation for another reasons.

Then, in order to restart the job you need to perform the following actions:

- **Do Not remove** the file called `cache.db` from the current work directory.

4.5 Reporting a bug or requesting a feature

To report an issue or request a new feature you can use the github issues tracker.

CHAPTER 5

Absorption Spectrum

This other workflow compute the absorption spectrum for different snapshots in a MD trajectory.

```
workflow:
  absorption_spectrum
project_name:
  Cd
xc_dft:
  pbe
tddft:
  stda
active_space: [10, 15]

path_hdf5:
  "test/test_files/Cd.hdf5"
path_traj_xyz:
  "test/test_files/Cd.xyz"
scratch_path:
  "/tmp/namd/absorption_spectrum"

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  path_basis: "test/test_files"
  cell_parameters: 5.0
  cell_angles: [90.0, 90.0, 90.0]
  periodic: none

  cp2k_settings_main:
    specific:
      template: pbe_main

  cp2k_settings_guess:
    specific:
      template: pbe_guess
```

5.1 Reporting a bug or requesting a feature

To report an issue or request a new feature you can use the github issues tracker.

CHAPTER 6

Distribute Absorption Spectrum

This workflow computes the absorption spectra for a given molecular system and returns a set of files in TXT format. The principle of distribution workflow is dividing the work in multiple, separated, instances (chunks), in order to be able to split time-consuming jobs into smaller, quicker ones. The input is described in YAML format as showed in the following example:

```yaml
workflow:
  distribute_absorption_spectrum
project_name:
  DMSO_PbBr4
xc_dft:
  pbe
tddft:
  stda
active_space: [20, 20]
stride:
  10
path_hdf5:
  "/scratch-shared/frazac/tutorial/guanine.hdf5"
path_traj_xyz:
  "/home/frazac/IVAN/Guanine/MD/qmworks-cp2k-pos-1.xyz"
scratch_path:
  "/scratch-shared/frazac/guanine/"
workdir: "."
blocks: 5

job_scheduler:
  scheduler: SLURM
  nodes: 1
  tasks: 24
  wall_time: "1:00:00"
  queue_name: "short"
  load_modules: "source activate qmflows\nmodule load eb\nmodule load CP2K/5.1-foss-
↪2017b"
```

(continues on next page)

(continued from previous page)

```
cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  path_basis: "/home/frazac/cp2k_basis"
  periodic: "xyz"
  charge: 0
  cell_parameters: 25.0
  cell_angles: [90.0,90.0,90.0]

  cp2k_settings_main:
    specific:
      template: pbe_main

  cp2k_settings_guess:
     specific:
      template: pbe_guess
```

The input *template_distribute_absorption_spectrum.yml* file contains all settings to perform the calculations and needs to be edited according to your system and preferences. Pay attention to the following parameters, which are specific for this workflow:

- **stride**: this parameter controls the accuracy of sampling of geometries contained in the MD trajectory of reference. For example, a value of stride: 10 indicates that the spectrum analysis will be performed on 1 out of 10 points in the reference trajectory. Two important things have to be pointed out:
  1. The workflow will perform SCF calculations for each point in the trajectory; only afterwards it will sample the number of structures on which the spectrum analysis will be performed
  2. Down-sampling issues might arise from the number of points that are actually printed during the MD calculations. Some programs, indeed, offer the possibility to print (in the output file) only one point out of ten (or more) calculated. In this case, applying a stride: 10 would in practice mean that you are sampling 1 point out of 100 points in the trajectory.
- **blocks**: this parameter indicates into how many blocks has the job to be split. This will generate as many chunks' folders in your working directory, all of each containing th

Note: TRIPLETs

6.1 Reporting a bug or requesting a feature

To report an issue or request a new feature you can use the github issues tracker.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search