nano-qmflows Documentation

Release 0.14.3.dev0+g522f8e0.d20231011

Felipe Zapata and Ivan Infante

Oct 11, 2023

INTRODUCTION

Contents:

CHAPTER

ONE

NANO-QMFLOWS

Nano-QMFlows is a generic python library for computing (numerically) electronic properties for nanomaterials like the 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 nano-qmflows and how to use the program see the documentation.

1.1 Installation

Pre-compiled binaries are available on pypi and can be installed on MacOS and Linux as following:

```
pip install nano-qmflows --upgrade
```

1.2 Building from source

Building Nano-QMFlows from source first requires an installation of *Miniconda* as is detailed here.

Then, to install the **nano-qmflows** library type the following commands inside the conda environment:

```
# Create the conda environment
conda create -n qmflows -c conda-forge boost eigen "libint>=2.6.0" highfive
conda activate qmflows

# Clone the repo
git clone https://github.com/SCM-NV/nano-qmflows
cd nano-qmflows

# Build and install nano-qmflows
pip install -e . --upgrade
```

Note: Older compilers, such as GCC <7, might not be compatible with the latest `eigen` version and require specification of *e.g.* `eigen=3.3`.

1.3 Advantages and Limitations

nano-qmflows 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. nano-qmflows 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 nano-qmflows to a QM code is reduced to writing a simple module that reads the MOs coefficients in the specific code format. At this moment, nano-qmflows handles output formats generated by CP2K, Orca, and Gamess, but, as said, it can be easily extended to other codes.

Finally, nano-qmflows 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, nano-qmflows is expected to offer new functionalities.

1.4 Interface to Pyxaid

nano-qmflows 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.

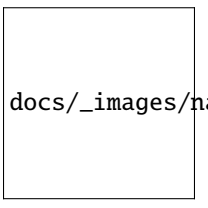
In this framework, nano-qmflows requires as input the coordinates of a pre-computed trajectory (at a lower level or at the same level of theory) in xyz format and the input parameters of the SCF code (HF and DFT). nano-qmflows will then calculate the overlap matrix between different MOs by correcting 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.

1.5 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.

1.5.1 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.



CHAPTER

TWO

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

THREE

INTRODUCTION TO THE TUTORIALS

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

- single_points
- coop_calculation
- ipr_calculation
- 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

FOUR

SINGLE POINTS CALCULATION

The single_points workflow performs single point calculations and can be used, for example, to compute the eigenvalues and energies of triplet orbitals on a singlet geometry or viceversa.

4.1 Preparing the input

4.1.1 Basic Example

We start with the very basic example of an input file to perform a single point calculation on the relaxed geometry of a Cd33Se33 system.

```
workflow:
  single_points

project_name: Cd33Se33
active_space: [50, 50]
compute_orbitals: True
path_hdf5: "Cd33Se33.hdf5"
path_traj_xyz: "Cd33Se33.xyz"
scratch_path: "/tmp/singlepoints_basic"

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 28.0
  periodic: none
  executable: cp2k.popt

  cp2k_settings_main:
    specific:
      template: pbe_main

  cp2k_settings_guess:
    specific:
      template:
        pbe_guess
```

In your working directory, create an *input_test_single_points_basic.yml* file and copy the previous input inside it, by respecting the indentation. Also copy locally the file containing the coordinates of the Cd33Se33 system in an xyz format, Cd33Se33.xyz.

Your *input_test_single_points_basic.yml* now contains all settings to perform the calculations and needs to be edited according to your system and preferences. Pay attention to the following parameters that are common to all input files: **workflow**, **project_name**, **active_space**, **path_hdf5**, **path_traj_xyz**, **scratch_path**.

- **workflow**: The workflow you need for your calculations, in this case single_points for a single point calculation.
- **project_name**: Project name for the calculations. You can choose what you wish.
- **active_space**: Range of (occupied, virtual) molecular orbitals to be computed. For example, if 50 occupied and 100 virtual should be considered in your calculations, the active space should be set to [50, 100].
- **compute_orbitals**: Specify if the energy and eigenvalues of the selected orbitals are to be computed. The default is set to True.
- **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-optimized geometry of your system. 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.

You can find the complete list of all the general options (common to all workflows) in this dictionary.

In the cp2k_general_settings, you can customize the settings used to generate the cp2k input (see available options in schema_cp2k_general_settings).

Here you can specify the level of theory you want to use in your cp2k calculation (basis set and potential) as well as the main characteristics of your system (cell parameters and angles, periodicity, charge, multiplicity, ...).

Note that the (fast) SCF routine in cp2k is based on the Orbital Transformation (OT) method, which works on the occupied orbital subspace. To obtain the full spectrum of molecular orbitals, one should perform a full diagonalization of the Fock matrix. For this reason, to obtain and store both occupied and unoccupied orbitals, defined using the active_space keyword, we have to follow a 2-step procedure: in the first step, which in the yaml input we define as cp2k_settings_guess, we perform a single point calculation using the fast OT approach; then in the second step, defined as cp2k_settings_main, we use the converged orbitals in the first step to start a full diagonalization calculation using the DIIS procedure.

In the cp2k_settings_guess and cp2k_settings_main subsections you can provide more detailed information about the cp2k input settings to be used to compute the guess wavefunction and to perform the main calculation respectively. In this example, we have used one of the available templates, specifically customized for calculations with a PBE exchange-correlation functional. You can use the cp2k manual to further personalize your input requirements.

4.1.2 Advanced Example

We are now ready to move to a more advanced example in which we want to compute the orbitals' energies and eigenvalues for each point of a pre-computed MD trajectory for our Cd33Se33 system. The input file will look like that:

```
workflow:
  single_points

project_name: Cd33Se33
active_space: [50, 50]
dt: 1
path_hdf5: "Cd33Se33.hdf5"
path_traj_xyz: "Cd33Se33_fivePoints.xyz"
```

(continues on next page)

(continued from previous page)

```
scratch_path: "/tmp/singlepoints_advanced"
calculate_guesses: "first"

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 28.0
  periodic: none
  executable: cp2k.popt

  cp2k_settings_main:
    specific:
      template: pbe_main
      cp2k:
        force_eval:
          dft:
            scf:
              eps_scf: 1e-6

  cp2k_settings_guess:
    specific:
      template:
        pbe_guess
```

In your working directory, create an *input_test_single_points_advanced.yml* file and copy the previous input inside it (remember to respect the indentation). Also copy locally the small pre-computed MD trajectory of the Cd33Se33 system, Cd33Se33_fivePoints.xyz.

In the input file, pay particular attention to the following parameters that have been added/modified with respect to the previous example:

- **dt**: The size of the timestep used in your MD simulations (in fs).
- **path_traj_xyz**: Path to the pre-computed MD trajectory. It should be provided in xyz format.
- **calculate_guesses**: Specify whether to calculate the guess wave function only in the first point of the trajectory ("first") or in all ("all). Here, we keep the default value, first.

In this example, we also show how to further personalize the cp2k_general_settings. In particular, a cp2k subsection is added to overwrite some parameters of the pbe template and tighten the scf convergence criterion to 1e-6 (the default value in the pbe_main template is 5e-4). Please note that a specific indentation is used to reproduce the structure of a typical cp2k input file. By using this approach, you can easily personalize your input requirements by referring to the cp2k manual.

A more elaborate example would have involved the computation of the eigenvalues and energies of orbitals in the **triplet** state for each point of this **singlet** trajectory. This would have been done by simply adding `multiplicity: 3` under the cp2k_general_settings block.

4.2 Setting up the calculation

Once all settings of your yml input have been customized, you are ready to launch your single point calculation.

- First, activate the conda environment with QMFlows:

  ```
  conda activate qmflows
  ```

- Then, load the module with your version of cp2k, for example:

  ```
  module load CP2K/7.1.0
  ```

- Finally, use the command run_workflow.py to submit your calculation:

  ```
  run_workflow.py -i input_test_single_points_basic.yml
  ```

  or

  ```
  run_workflow.py -i input_test_single_points_advanced.yml
  ```

  for the advanced example.

CHAPTER

FIVE

CRYSTAL ORBITAL OVERLAP POPULATION (COOP) CALCULATION

The workflow coop_calculation allows to compute the crystal orbital overlap population between two selected elements.

5.1 Preparing the input

The following is an example of input file to perform the COOP calculation between Cd and Se for the Cd33Se33 system.

```
workflow:
  coop_calculation

project_name: Cd33Se33
active_space: [50, 50]
path_hdf5: "Cd33Se33.hdf5"
path_traj_xyz: "Cd33Se33.xyz"
scratch_path: "/tmp/COOP"

coop_elements: ["Cd", "Se"]

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 28.0
  periodic: none
  executable: cp2k.popt

  cp2k_settings_main:
    specific:
      template: pbe_main
      cp2k:
        force_eval:
          dft:
            scf:
              eps_scf: 1e-6

  cp2k_settings_guess:
    specific:
      template:
        pbe_guess
```

In your working directory, copy the previous input into an *input_test_coop.yml* file. Also copy locally the file containing the coordinates of the relaxed Cd33Se33 system, Cd33Se33.xyz.

Your *input_test_coop.yml* input file now contains all settings to perform the coop calculations and needs to be edited according to your system and preferences. Please note that this input is very similar to the basic example of single point calculation provided in a previous tutorial (please refer to it for a more extensive description of the above options) except for the following options: **workflow**, **coop_elements**.

- **workflow**: The workflow you need for your calculations, in this case set to coop_calculation is this case.
- **coop_elements**: List of the two elements to calculate the COOP for, here Cd and Se.

In the cp2k_general_settings, you can customize the settings used to generate the cp2k input. To help you creating your custom input requirements, please consult the cp2k manual and the templates available in nano-qmflows.

5.2 Setting up the calculation

Once all settings of your yml input have been customized, can to launch your coop calculation.

- First, activate the conda environment with QMFlows:

  ```
  conda activate qmflows
  ```

- Then, load the module with your version of cp2k, for example:

  ```
  module load CP2K/7.1.0
  ```

- Finally, use the command run_workflow.py to submit your calculation.

  ```
  run_workflow.py -i input_test_coop.yml
  ```

5.3 Results

Once your calculation has finished successfully, you will find a *COOP.txt* file in your working directory. The two columns of this file contain, respectively, the orbitals' energies and the corresponding COOP values for the selected atoms pair.

CHAPTER

SIX

INVERSE PARTICIPATION RATIO (IPR) CALCULATION

The workflow ipr_calculation returns the inverse participation ratio for the selected orbitals. For finite systems, the IPR is defined as the inverse of number of atoms that contribute to a given electronic state i. It assumes its maximum value, 1, in the case of a state localized to a single atom (1/1) and tends to 0 (1/N, where N is the total number of atoms in the system) when the wave function is distributed equally over all atoms.

6.1 Preparing the input

The following is an example of input file to perform the IPR calculation for the Cd33Se33 system.

```
workflow:
   ipr_calculation

project_name: Cd33Se33
active_space: [50, 50]
path_hdf5: "Cd33Se33.hdf5"
path_traj_xyz: "Cd33Se33.xyz"
scratch_path: "/tmp/IPR"

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 28.0
  periodic: none
  executable: cp2k.popt

  cp2k_settings_main:
    specific:
      template: pbe_main
      cp2k:
        force_eval:
          dft:
            scf:
              eps_scf: 1e-6

  cp2k_settings_guess:
    specific:
      template:
        pbe_guess
```

In your working directory, copy the previous input into an *input_test_ipr.yml* file. Also copy locally the file containing the coordinates of the relaxed Cd33Se33 system, Cd33Se33.xyz.

Your *input_test_ipr.yml* input file now contains all settings to perform the coop calculations and needs to be edited according to your system and preferences. Please note that this input is very similar to the basic example of single point calculation provided in a previous tutorial (please refer to it for a more extensive description of the above options) except for the **workflow** option, set in this case to *ipr_calculation*.

Here again you can customize the settings used to generate the cp2k input in the cp2k_general_settings. To help you creating your custom input requirements, please consult the cp2k manual and the templates available in nano-qmflows.

6.2 Setting up the calculation

Once all settings of your yml input have been customized, can to launch your ipr calculation.

- First, activate the conda environment with QMFlows:

  ```
  conda activate qmflows
  ```

- Then, load the module with your version of cp2k, for example:

  ```
  module load CP2K/7.1.0
  ```

- Finally, use the command run_workflow.py to submit your calculation.

  ```
  run_workflow.py -i input_test_ipr.yml
  ```

6.3 Results

Once your calculation has finished successfully, you will find a *IPR.txt* file in your working directory. The two columns of this file contain, respectively, the orbitals' energies and the corresponding IPR values.

CHAPTER

SEVEN

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 nano-qmflows installed.

7.1 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: 2

job_scheduler:
  free_format: "
  #! /bin/bash\n
  #SBATCH --job-name=Cd33Se33\n
  #SBATCH -N 1\n
  #SBATCH -t 00:15:00\n
  #SBATCH -p short\n
  source activate qmflows\n
  module load cp2k/3.0\n\n"

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 28.0
```

(continues on next page)

(continued from previous page)

```
file_cell_parameters: "test/test_files/file_distribute_cell_parameters.txt"
periodic: none
executable: cp2k.popt

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.

7.2 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.

7.3 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_01.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 -o chunk_01.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_01.hdf5 using the *removeHDF5folders.py* script. To run the script, use:

  ```
  removeHDF5folders.py -hdf5 chunk_01.hdf5
  ```

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 -hdf5 chunk_01.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.

7.4 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.

7.5 Reporting a bug or requesting a feature

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

CHAPTER

EIGHT

ABSORPTION SPECTRUM

This other workflow computes the excited states energies, transition dipole moments and oscillator strength using the STDDFT approach.

8.1 Preparing the input

8.1.1 Basic Example

Below is a basic example of input file for the computation of the first 400 (20*20, as setted in the active_space) lowest lying excited states of a guanine molecule at the sTDA level of approximation.

```
workflow:
  absorption_spectrum

project_name: guanine
active_space: [20, 20]
compute_orbitals: True
path_hdf5: "guanine.hdf5"
path_traj_xyz: "guanine.xyz"
scratch_path: "/tmp/absorption_spectrum_basic"

xc_dft: pbe
tddft: stda

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 25.0
  periodic: none
  executable: cp2k.popt

  cp2k_settings_main:
    specific:
      template: pbe_main

  cp2k_settings_guess:
    specific:
      template: pbe_guess
```

In your working directory, create an *input_test_absorption_spectrum_basic.yml* file and copy the previous input inside it, by paying attention to preserve the correct indentation. Also copy locally the file containing the coordinates of the relaxed geometry of the guanine in an xyz format, guanine.xyz.

At this point, your *input_test_absorption_spectrum_basic.yml* contains all the settings to perform the excited states calculation and needs to be edited according to your system and preferences. First, let's recall some parameters that are common to all input files: **workflow**, **project_name**, **active_space**, **path_hdf5**, **path_traj_xyz**, **scratch_path**.

- **workflow**: The workflow you need for your calculations, in this case absorption_spectrum to compute excited states properties.
- **project_name**: Project name for the calculations. You can choose what you wish.
- **active_space**: Range of (doubly occupied, virtual) molecular orbitals to be computed. For example, if 50 occupied and 100 virtual should be considered in your calculations, the active space should be set to [50, 100]. The range will automatically be appended with additional singly occupied MOs based on the systems (user-specified) multiplicity.
- **compute_orbitals**: Specify if the energy and eigenvalues of the selected orbitals are to be computed. The default is set to True so we will not consider it in the advanced examples.
- **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-optimized geometry of your system. 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.

You can find the complete list of these general options in this dictionary.

Also pay particular attention to the following parameters, specific to the absorption_spectrum workflow:

- **xc_dft**: Type of exchange-correlation functional used in your DFT calculations.
- **tddft**: Type of approximation used in the excited states calculations. The Single Orbital (sing_orb), sTDA (stda) and sTDDFT (stddft) approximations are available.

In the cp2k_general_settings, you can customize the settings used to generate the cp2k input of the initial single point calculations (from which Molecular Orbital energies and coefficients are retrieved). For more details about this section please refer to the available tutorial on single point calculations. To further personalize the input requirements, also consult the cp2k manual and the templates available in nano-qmflows.

8.1.2 Advanced Example

We are now ready to move to a more advanced example in which we want to compute the excited states of our guanine molecule starting from a pre-computed MD trajectory rather than a single geometry. The input file will look like that:

```
workflow:
  absorption_spectrum

project_name: guanine
active_space: [20, 20]
dt: 1
path_hdf5: "guanine.hdf5"
path_traj_xyz: "guanine_twentyPoints.xyz"
scratch_path: "/tmp/absorption_spectrum_advanced1"
calculate_guesses: "first"
```

(continues on next page)

(continued from previous page)

```
xc_dft: pbe
tddft: stda
stride: 4

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 25.0
  periodic: none
  executable: cp2k.popt

  cp2k_settings_main:
    specific:
      template: pbe_main

  cp2k_settings_guess:
    specific:
      template: pbe_guess
```

In your working directory, create an *input_test_absorption_spectrum_advanced.yml* file and copy the previous input inside it (remember to respect the indentation). Also copy locally the small pre-computed MD trajectory of the guanine system, guanine_twentyPoints.xyz.

In the input file, pay particular attention to the following parameters that have been added/modified with respect to the previous example:

- **dt**: The size of the timestep used in your MD simulations (in fs).
- **path_traj_xyz**: Path to the pre-computed MD trajectory. It should be provided in xyz format.
- **calculate_guesses**: Specify whether to calculate the guess wave function only in the first point of the trajectory ("first") or in all ("all). Here, we keep the default value, first.
- **stride**: Controls the accuracy of sampling of geometries contained in the MD trajectory of reference. For example, our value of stride: 4 indicates that the spectrum analysis will be performed on 1 out of 4 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 (twenty points in our example); only afterwards it will sample the number of structures on which the spectrum analysis will be performed (here six structures corresponding to points 0, 4, 8, 12, 16, 20).
  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: 4 would in practice mean that you are sampling 1 point out of 40 points in the trajectory.

8.2 Setting up the calculation

Once all settings of your yml input have been customized, you are ready to launch your single point calculation.

- First, activate the conda environment with QMFlows:

  ```
  conda activate qmflows
  ```

- Then, load the module with your version of cp2k, for example:

  ```
  module load CP2K/7.1.0
  ```

- Finally, use the command run_workflow.py to submit your calculation:

  ```
  run_workflow.py -i input_test_absorption_spectrum_basic.yml
  ```

for the basic example.

8.3 Results

Once your calculation has finished successfully, you will find one (or more) *output_n_stda.txt* file(s) in your scratch directory (with *n* being the index of the geometry at which the spectrum analysis has been performed). The first two lines of the file *output_0_stda.txt* generated in our basic example are reported below.

```
# state    energy       f      t_dip_x    t_dip_y    t_dip_y    weight   from   energy
↪to     energy     delta_E
    1      4.566    0.03832   -0.51792   -0.25870    0.08573    0.50158  20     -5.175
↪21     -1.261      3.914
```

For each excited state (line), the first six columns contain, from left to right:

- *# state*: Assigned index, in ascending order of energy. Here, the lowest excitation is reported and corresponds to # state 1.
- *energy*: Transition energy, in eV.
- *f*: Oscillator strength, dimensionless.
- *t_dip_x*, *t_dip_y*, *t_dip_z*: Transition dipole moment components along x, y and z.

The next six columns report some useful information about the dominant single orbital transition for the excited state under examination:

- *weight*: Weight in the overall transition. Always 1.0000 in the Single Orbital approximation.
- *from*: Index of the initial occupied orbital in the active space.
- *energy*: Energy of the initial occupied orbital.
- *to*: Index of the final virtual orbital in the active space.
- *energy*: Energy of the final virtual orbital.
- *delta_E*:Energy of the dominant single orbital transition. Corresponds to the excited state energy in the Single Orbital approximation.

Copy the output file(s) to your working directory and plot the absorption spectrum using the script convolution.py:

```
convolution.py -nm True
```

In case of multiple output files, the returned absorption spectrum is an average over all sampled strucutures, unless you define the index of a specific sample using the -n option.

8.4 Reporting a bug or requesting a feature

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

CHAPTER

NINE

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.

In this tutorial, we want to compute the excited states at **each point** of a pre-computed MD trajectory for the guanine system. Please note that this trajectory has already been used in the advanced example of the absorption_spectrum tutorial, where the spectrum analysis was performed on 1 out of 4 points only. Here we take advantage of the distribution workflow to increase by four times the accuracy of sampling with no substantial variation of the computational cost in terms of time by dividing the job in five chunks (each taking charge of 4 points out of 20).

9.1 Preparing the input

The input is described in YAML format as showed in the following example:

```
workflow:
  distribute_absorption_spectrum

project_name: guanine_distribution
active_space: [20, 20]
dt: 1
path_hdf5: "guanine.hdf5"
path_traj_xyz: "guanine_twentyPoints.xyz"
scratch_path: "/tmp/distribute_absorption_spectrum"
calculate_guesses: "first"

xc_dft: pbe
tddft: stda
stride: 1

blocks: 5
workdir: "."

job_scheduler:
  free_format: "
  #! /bin/bash\n
  #SBATCH --job-name=guanine_distribution\n
  #SBATCH -N 1\n
  #SBATCH -t 1:00:00\n
  #SBATCH -p short\n
```

(continues on next page)

(continued from previous page)

```
  source activate qmflows\n
  module load CP2K/7.1.0\n\n"

cp2k_general_settings:
  basis:  "DZVP-MOLOPT-SR-GTH"
  potential: "GTH-PBE"
  cell_parameters: 25.0
  periodic: none
  executable: cp2k.popt

  cp2k_settings_main:
    specific:
      template: pbe_main

  cp2k_settings_guess:
    specific:
      template: pbe_guess
```

In your working directory, create an *input_test_distribute_absorption_spectrum.yml* file and copy the previous input inside it (remember to respect the indentation). Also copy locally the small pre-computed MD trajectory of the guanine system, guanine_twentyPoints.xyz.

In the input file, pay particular attention to the following parameters that have been added/modified with respect to the previous tutorial (advanced example):

- **stride**: Controls the accuracy of sampling of geometries contained in the MD trajectory of reference. Here, a value of stride: 1 indicates that the spectrum analysis will be performed on each point 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. For example, applying a stride: 10 would in practice mean that you are sampling 1 point out of 100 points in the trajectory.
- **blocks**: Indicates into how many blocks has the job to be split. This will generate as many chunks' folders in your working directory.
- **workdir**: Path to the chunks' folders.

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

9.2 Setting up the calculation

Once all settings in *input_test_distribute_absorption_spectrum.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_absorption_spectrum.yml
  ```

A number of new folders are created. In each folder you will find a submission file, launch.sh, a sub-trajectory file (containing the points assigned to that chunk), chunk_xyz, 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.

9.3 Results

Once the calculations are completed, you will find multiple *output_n_stda.txt* files in your scratch directories (with *n* being the index of the geometry at which the spectrum analysis has been performed). The first two lines of the file *output_0_stda.txt*, found in /tmp/distribute_absorption_spectrum/scratch_chunk_0/ are reported below.

```
# state    energy       f      t_dip_x    t_dip_y    t_dip_y    weight   from   energy␣
↪to     energy     delta_E
    1      4.566    0.03832   -0.51792   -0.25870    0.08573    0.50158  20     -5.175␣
↪21     -1.261      3.914
```

For each excited state (line), the first six columns contain, from left to right:

- *# state*: Assigned index, in ascending order of energy. Here, the lowest excitation is reported and corresponds to # state 1.
- *energy*: Transition energy, in eV.
- *f*: Oscillator strength, dimensionless.
- *t_dip_x*, *t_dip_y*, *t_dip_z*: Transition dipole moment components along x, y and z.

The next six columns report some useful information about the dominant single orbital transition for the excited state under examination:

- *weight*: Weight in the overall transition. Always 1.0000 in the Single Orbital approximation.
- *from*: Index of the initial occupied orbital in the active space.
- *energy*: Energy of the initial occupied orbital.
- *to*: Index of the final virtual orbital in the active space.
- *energy*: Energy of the final virtual orbital.
- *delta_E*:Energy of the dominant single orbital transition. Corresponds to the excited state energy in the Single Orbital approximation.

Copy all the output files to your working directory and plot the absorption spectrum (averaged over all sampled structures) using the script convolution.py:

```
convolution.py -nm True
```

To plot the absorption spectrum of a specific sample, for example our point 0, use the -n option.

```
convolution.py -n 0 -nm True
```

9.4 Reporting a bug or requesting a feature

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

For a more detailed description of **nano-qmflows** read the documentation

CHAPTER

TEN

COMMAND LINE INTERFACE

10.1 Running a workflow

Comman line interface to run the workflows.

Usage:
run_workflow.py -i input.yml

Available workflow:

- absorption_spectrum
- derivative_couplings
- single_points
- ipr_calculation
- coop_calculation

10.2 Workflows distribution

Command line interface to split a given workflow into several chunks.

Usage:
distribute_jobs.py -i input.yml

THE USER MUST CHANGES THESE VARIABLES ACCORDING TO HER/HIS NEEDS:

- project_name
- path to the basis and Cp2k Potential
- **CP2K:**
  - Range of Molecular oribtals printed by CP2K
  - Cell parameter
- Settings to Run Cp2k simulations
- Path to the trajectory in XYZ

The slurm configuration is optional but the user can edit it:

property default

- nodes 2

- tasks 24
- time 48:00:00
- name namd

Otherwise the user can fill the the `free_format` property with her own configuration in the yaml input file.

CHAPTER

ELEVEN

CP2K INTERFACE

Module to configure and run CP2K jobs.

11.1 Index

prepare_job_cp2k(settings, dict_input, guess_job)	*(scheduled)* Generate a qmflows.packages.CP2K job.

11.2 API

nanoqm.schedule.scheduleCP2K.**prepare_job_cp2k**(*settings: Settings*, *dict_input: _data.ComponentsData*, *guess_job: None | PromisedObject*) → CP2K

(scheduled) Generate a qmflows.packages.CP2K job.

Parameters

- **settings** – Input for CP2K
- **dict_input** – Input for the current molecular geometry
- **guess_job** – Previous job to read the guess wave function

Returns
job to run

Return type
qmflows.packages.CP2K

CHAPTER

TWELVE

DERIVATIVE COUPLINGS

Compute the nonadiabatic coupling using different methods.

The available methods are:

- *3-point numerical differentiation <https://doi.org/10.1063/1.4738960>*
- *levine <dx.doi.org/10.1021/jz5009449>*

Phase correction is also available.

12.1 Index

`calculate_couplings_3points`(dt, mtx_sji_t0, ...)	Calculate the non-adiabatic interaction matrix using 3 geometries.
`calculate_couplings_levine`(dt, w_jk, w_kj)	Compute coupling using the Levine approximation.
`compute_overlaps_for_coupling`(config, ...)	Compute the Overlap matrices used to compute the couplings.
`correct_phases`(overlaps, mtx_phases)	Correct the phases for all the overlaps.

12.2 API

nanoqm.integrals.nonAdiabaticCoupling.**calculate_couplings_3points**(*dt: float*, *mtx_sji_t0: ndarray*, *mtx_sij_t0: ndarray*, *mtx_sji_t1: ndarray*, *mtx_sij_t1: ndarray*) → ndarray

Calculate the non-adiabatic interaction matrix using 3 geometries.

see: https://aip.scitation.org/doi/10.1063/1.467455 the contracted Gaussian functions for the atoms and molecular orbitals coefficients are read from a HDF5 File.

Parameters

- **dt** – Integration step (atomic units)
- **mtx_sji_t0** – Sji Overlap matrix at time t0
- **mtx_sij_t0** – SiJ Overlap matrix at time t0
- **mtx_sji_t1** – Sji Overlap matrix at time t1

- **mtx_sij_t1** – SiJ Overlap matrix at time t1

Returns
Coupling matrix

Return type
np.ndarray

nanoqm.integrals.nonAdiabaticCoupling.**calculate_couplings_levine**(*dt: float*, *w_jk: ndarray*, *w_kj: ndarray*) → ndarray

Compute coupling using the Levine approximation.

Compute the non-adiabatic coupling according to: *Evaluation of the Time-Derivative Coupling for Accurate Electronic State Transition Probabilities from Numerical Simulations*. Garrett A. Meek and Benjamin G. Levine. dx.doi.org/10.1021/jz5009449 | J. Phys. Chem. Lett. 2014, 5, 23512356

Notes

In numpy sinc is defined as sin(pi * x) / (pi * x)

Parameters

- **dt** – Integration step (atomic units)
- **w_jk** – Overlap matrix
- **mtx_sij_t0** – Overlap matrix

Returns
Coupling matrix

Return type
np.ndarray

nanoqm.integrals.nonAdiabaticCoupling.**compute_overlaps_for_coupling**(*config: _data.GeneralOptions*, *pair_molecules: tuple[MolXYZ, MolXYZ]*, *coefficients: tuple[Matrix, Matrix]*) → Matrix

Compute the Overlap matrices used to compute the couplings.

Parameters

- **config** – Configuration of the current task
- **pair_molecule** – Molecule to compute the overlap
- **coefficients** – Molecular orbital coefficients for each molecule

Returns
containing the overlaps at different times

Return type
Matrix

nanoqm.integrals.nonAdiabaticCoupling.**correct_phases**(*overlaps: ndarray*, *mtx_phases: ndarray*) → ndarray

Correct the phases for all the overlaps.

CHAPTER

THIRTEEN

MOLECULAR ORBITALS

Molecular orbitals calculation using CP2K and *QMFlows <https://github.com/SCM-NV/qmflows>*.

13.1 API

nanoqm.schedule.components.**calculate_mos**(*config: _data.GeneralOptions*) → PromisedObject

Look for the MO in the HDF5 file and compute them if they are not present.

The orbitals are computed by splitting the jobs in batches given by the `restart_chunk` variables. Only the first job is calculated from scratch while the rest of the batch uses as guess the wave function of the first calculation inthe batch.

The config dict contains:

- geometries: list of molecular geometries
- project_name: Name of the project used as root path for storing data in HDF5.
- path_hdf5: Path to the HDF5 file that contains the numerical results.
- folders: path to the directories containing the MO outputs
- settings_main: Settings for the job to run.
- calc_new_wf_guess_on_points: Calculate a new Wave function guess in each of the geometries indicated. By Default only an initial guess is computed.
- enumerate_from: Number from where to start enumerating the folders create for each point in the MD

Return type

paths to the datasets in the HDF5 file containging both the MO energies and MO coefficients

CHAPTER

FOURTEEN

INTEGRALS

Compute multipole integrals using *Libint2 <https://github.com/evaleev/libint/wiki>*.

The interface to the C++ Libint2 library is located at the parent folder, in the *libint* folder.

14.1 Index

get_multipole_matrix(config, inp, multipole)	Retrieve the *multipole* number *i* from the trajectory.
compute_matrix_multipole(mol, config, multipole)	Compute a *multipole* matrix: overlap, dipole, etc.

14.2 API

nanoqm.integrals.multipole_matrices.**get_multipole_matrix**(*config: _data.AbsorptionSpectrum*, *inp: _data.AbsorptionData*, *multipole: Literal['overlap', 'dipole', 'quadrupole']*) → NDArray[f8]

Retrieve the *multipole* number *i* from the trajectory. Otherwise compute it.

Parameters

- **config** – Global configuration to run a workflow
- **inp** – Information about the current point, e.g. molecular geometry.
- **multipole** – Either overlap, dipole or quadrupole.

Returns
Tensor containing the multipole.

Return type
np.ndarray

nanoqm.integrals.multipole_matrices.**compute_matrix_multipole**(*mol: list[AtomXYZ]*, *config: _data.GeneralOptions*, *multipole: Literal['overlap', 'dipole', 'quadrupole']*) → NDArray[f8]

Compute a *multipole* matrix: overlap, dipole, etc. for a given geometry *mol*.

The multipole is Computed in spherical coordinates.

Note: for the dipole and quadrupole the super_matrix contains all the matrices stack all the 0-axis.

Parameters

- **mol** – Molecule to compute the dipole
- **config** – Dictionary with the current configuration
- **multipole** – kind of multipole to compute

Returns
Matrix with entries <i | x^i y^j z^k | j>

Return type
np.ndarray

CHAPTER

FIFTEEN

WORKFLOWS

The following workflows are available:

nanoqm.workflows.workflow_coop.**workflow_crystal_orbital_overlap_population**(*config: _data.COOP*) → NDArray[f8]

Compute the Crystal Orbital Overlap Population.

nanoqm.workflows.workflow_coupling.**workflow_derivative_couplings**(*config: _data.DerivativeCoupling*) → ResultPaths | tuple[ResultPaths, ResultPaths]

Compute the derivative couplings for a molecular dynamic trajectory.

Parameters
config – Dictionary with the configuration to run the workflows

Return type
Folders where the Hamiltonians are stored.

nanoqm.workflows.workflow_ipr.**workflow_ipr**(*config: _data.IPR*) → np.ndarray

Compute the Inverse Participation Ratio main function.

nanoqm.workflows.workflow_single_points.**workflow_single_points**(*config: _data.SinglePoints*) → tuple[list[tuple[str, str, str]], list[str]]

Perform single point calculations for a given trajectory.

Parameters
config – Input to run the workflow.

Return type
List with the node path to the molecular orbitals in the HDF5.

nanoqm.workflows.workflow_stddft_spectrum.**workflow_stddft**(*config: AbsorptionSpectrum*) → None

Compute the excited states using simplified TDDFT.

Both restricted and unrestricted orbitals calculations are available.

Parameters
config – Dictionary with the configuration to run the workflows

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX