rt_scf module

Real-time dynamics for self-consistent-field (HF and KS) objects

***class* RT_SCF**

Real-time SCF base class

Attributes:

timestep: ***float or int***

Step size (AU).

max_time: ***float or int***

Propagation end time (AU).

filename: ***str***

Output file name. Default is None, printing to the standard output.

prop: ***str***

Integrator method. Default is 2nd order interpolated Magnus.

rk4: 4th Order Runge-Kutta
magnus_step: Modified Midpoint and Unitary Transformation (MMUT)
magnus_interpol: 2nd Order Interpolated Magnus

frequency: ***int***

Observable print frequency. Default is 1, which corresponds to printing observables at every time step.

orth: ***function***

Function to calculate an orthogonalization matrix from an overlap matrix. Default is canonical orthogonalization, as defined in pyscf.scf.addons.canonical_orth().

chkfile: ***str***

Checkpoint file to save molecular orbital coefficients and current time. At default this attribute is None and a chkfile will not be created. If chkfile exists and is an existing file, calculation will restart from the conditions given in the chkfile. Note that upon restarting a calculation, fields, print controls will need to be re-declared. If chkfile exists but is not an existing file, the calculation will generate a chkfile with the given name.

verbose: ***int***

Print level. Default is 3 (info level).

ovlp: ***2D ndarray***

Atomic orbital overlap matrix.

occ: ***1D ndarray***

Orbital occupancies. Initially equivalent to the occupancy of the given scf object, however can be modified following instantiation of a rt_scf class.

fragments: ***dict***

Dictionary mapping fragment scf objects to their corresponding partitions.

nmat: ***int***

For restricted and generalized calculations, nmat is 1. For unrestricted calculations, nmat is 2.

observables: ***dict***

Print control for which observables are to be collected. Default is every observable is false, and will not be calculated or printed during propagation.

mo_coeff_print: ***ndarray***

Molecular orbitals (MOs) (in atomic orbital basis) to print molecular orbital occupancies. Default is the given scf.mo_coeff basis (i.e. the ground state MOs).

magnus_tolerance: ***float***

Only applicable when using 2nd order interpolated Magnus integrator. Specifies the convergence threshold for the interpolation scheme for each time step. Default is 1e-7.

magnus_maxiter: ***int***

Only applicable when using 2nd order interpolated Magnus integrator. Specifies the maximum number of iterations for the interpolation scheme for each time step. Default is 15.

current_time: ***float or int***

Current propagation time.

den_ao: ***ndarray***

Current density matrix (in atomic orbital basis).

fock_ao: ***ndarray***

Current unperturbed fock matrix (in atomic orbital basis).

istype(type_code)

Checks if the object is an instance of the class specified by the type_code. type_code can be a class or a str. If the type_code is a class, it is equivalent to the Python built-in function isinstance. If the type_code is a str, it checks the type_code against the names of the object and all its parent classes.

update_time()

Updates self.current_time

get_fock_orth(den_ao)

Builds Fock matrix, orthogonalized Fock matrix.

rotate_coeff_to_orth(coeff_ao)

$\mathbf{C}' = \mathbf{X}^{-1}\mathbf{C}$

rotate_coeff_to_ao(coeff_orth)

$\mathbf{C} = \mathbf{X}\mathbf{C}'$

add_potential(*args)

Add a potential class, will be called in get_fock_orth

apply_potential()

Called in get_fock_orth, calculates and applies potentials given by rt_scf._potential.

kernel()

Main

rt_ehrenfest module

RT_Ehrenfest(RT_SCF)

Child class of rt_scf for Ehrenfest dynamics. Shares same attributes/methods unless (re)defined below.

Attributes:

Ne_step ***int***
Frequency at which nuclei are updated

N_step ***int***
Frequency at which forces are update

nuc ***rt_nuclei.Nuc***
Nuclei object

get_mo_coeff_print ***function***
For Ehrenfest calculation with MO occupations as an observable, the MOs need to be updated with every mol update. By default this will recalculate the ground state SCF orbitals for updated mol.

update_time()

Updates self.current_time, updates mol if needed

update_force()

Updates self.nuc.force

update_mol()

Resets self._scf.mol with updated mol

rt_prop module

Propagator loops for RT_SCF and RT_Ehrenfest dynamics

scf_propagate(rt_scf)

Propagation loop for real-time scf calculations.

ehrenfest_propagate(rt_ehrenfest)

Propagation loop for real-time ehrenfest calculations.

init_prop(rt_scf)

Initialization function called just before propagation. Determines integrator function, and calculates initial values and parameters needed for some integrators.

rt_integrators

Integrator functions

magnus_step(rt_scf)

Modified Midpoint and Unitary Transformation (MMUT)

$$\mathrm{C'}(t+dt) = \mathrm{U'}(t)\mathrm{C'}(t-dt)$$

$$\mathrm{U'}(t) = e^{-i\,2dt\,\mathrm{F'}(t)}$$

C', F', and U' are in the orthogonal atomic orbital basis.

magnus_interpol(rt_scf)

2nd Order Interpolated Magnus

$$\mathrm{C'}(t+dt) = \mathrm{U'}(t+\frac{dt}{2})\mathrm{C'}(t)$$

$$\mathrm{U'}(t+\frac{dt}{2}) = e^{-i\,dt\,\mathrm{F'}(t+\frac{dt}{2})}$$

C', F', and U' are in the orthogonal atomic orbital basis.

$\mathrm{F'}(t+\frac{dt}{2})$ is initially extrapolated from $\mathrm{F'}(t)$ and $\mathrm{F'}(t-\frac{dt}{2})$, which is saved from previous time steps.

$$\mathrm{F'}(t+\frac{dt}{2}) \approx 2\mathrm{F'}(t) - \mathrm{F'}(t-\frac{dt}{2})$$

Note that for the first time step, $\mathrm{F'}(t-\frac{dt}{2}) = \mathrm{F'}(t)$.
A time step is then performed to obtain $\mathrm{C'}(t+dt)$, which is used to build $\mathrm{F'}(t+dt)$. From here, an interpolation scheme is iteratively performed until $\mathrm{C'}(t+dt)$ converges to within the given rt_scf.magnus_tolerance (default is 1e-7).

$$\mathrm{F'}(t+\frac{dt}{2}) \approx 0.5\mathrm{F'}(t) + 0.5\mathrm{F'}(t+dt)$$

$$\mathrm{C'}(t+dt) = \mathrm{U'}(t+\frac{dt}{2})\mathrm{C'}(t)$$

rk4(rt_scf)

4th Order Runge-Kutta Note that the Fock matrix is not recalculated for each new k term. This is appropriate given that rk4 requires very small timesteps, and rebuilding the Fock matrix for each k term would roughly quadruple the cost.

get_integrator(rt_scf)

Returns integrator function specified by rt_scf.prop with a simple dictionary mapping.

rt_observables

Observable Functions

get_observables(rt_scf)

Calculates each observable specified in rt_scf.observables.

get_energy(rt_scf, den_ao)

Calculates energy for rt_scf and any fragments at current time.

get_charge(rt_scf, den_ao)

Calculates electronic charge for rt_scf and Mulliken charges for fragments at current time.

get_hirshfeld_charge(rt_scf, den_ao)

Calculates atomic Mulliken electronic charges for rt_scf at current time.

get_hirshfeld_charge(rt_scf, den_ao)

Calculates atomic Hirshfeld electronic charges for rt_scf at current time.

get_dipole(rt_scf, den_ao)

Calculates dipole moment for rt_scf at current time.

get_quadrupole(rt_scf, den_ao)

Calculates quadrupole moment for rt_scf at current time.

get_mag(rt_scf, den_ao)

Calculates magnetization for rt_scf at current time.

get_hirshfeld_mag(rt_scf, den_ao)

Calculates atomic Hirshfeld magnetizations for rt_scf at current time.

get_mo_occ(rt_scf, den_ao)

Calculates molecular orbital occupations for rt_scf in the basis of rt_scf.mo_coeff_print.

get_nuclei(rt_scf, den_ao)

Retrieves nuclei info for RT_Ehrenfest object

get_cube_density(rt_scf, den_ao)

Writes cube file. Will only write for propgation times given in rt_scf.cube_density_indices

rt_output

Output Function(s)

update_output(rt_scf)

Prints observables to output.

rt_utils

Utility Functions

excite(rt_scf, excitation_alpha=None, excitation_beta=None)

Removes an electron from the orbital occupation index specified by excitation_alpha or excitation_beta. For restricted or generalized calculations, specify the excitation index in excitation_alpha.

input_fragments(rt_scf, *fragments)

Adds fragment to rt_scf calculation. Some Observables will be calculated for both rt_scf and the associated fragments (energy, charge).

get_scf_orbitals(rt_ehrenfest)
get_noscf_orbitals(rt_ehrenfest)
restart_from_chkfile(rt_scf)

Loads stored information in chkfile into rt_scf for restarting a calculation.

update_chkfile(rt_scf)

Updates chkfile with current information of rt_scf, namely the molecular orbital coefficients and the current time.

print_info(rt_scf, mo_coeff_print)

Prints calculation info immediately prior to propagation

rt_vapp

class **ElectricField**

Electric field class for real-time scf objects. Recreation of NWChem's RTTDDFT electric field functions https://nwchemgit.github.io/RT-TDDFT.html#excite-excitation-rules

Attributes:

field_type: ***str***

Determines field type. Either "delta", "gaussian", "hann", or "resonant".

amplitude: ***list***

Field strength, given in a.u.

center: ***float or int***

For delta, gaussian and hann fields. Field is centered around this time value, given in a.u. Default is 0.

frequency: ***float or int***

Frequency for gaussian, hann, and resonant fields, given in a.u. Default is 0.

width: ***float or int***

Width in time for gaussian and hann fields, given in a.u. Default is 0.

phase: ***float or int***

Radial phase for gaussian, hann, and resonant fields. Default is 0.

delta_energy

Single impulse function. Energy equals amplitude at field center and zero everywhere else.

gaussian_energy

Gaussian enveloped field.

$$E(t) = A * \frac{e^{-(t-center)^2}}{2 * width^2} * \sin(\omega t + \phi)$$

A is the amplitude, ω is the frequency, and ϕ is the phase shift.

hann_energy

$\sin^2$ enveloped field.

$$E(t) = A * \sin^2(-\frac{\pi}{width} * (t - center - \frac{width}{2})) * \sin(\omega t + \phi)$$

A is the amplitude, ω is the frequency, and ϕ is the phase shift.

resonant_energy

Continuous oscillating field.

$$E(t) = A * \sin(\omega t + \phi)$$

A is the amplitude, ω is the frequency, and ϕ is the phase shift.

calculate_field_energy(rt_scf)

Returns field energy based on current propagation time stored in rt_scf.

calculate_potential(rt_scf)

Calculates applied potential.

$$V_{app} = \sum_{j=x,y,z} -D^j E^j(t)$$

where D is the electronic dipole matrix given by

$$D_{ij} = -\langle \Psi_i | \hat{r} | \Psi_j \rangle$$

and $E(t)$ is the current field energy given by calculate_field_energy().

rt_cap

***class* MOCAP**

Molecular Orbital Complex Absorbing Potential (MOCAP), as defined in https://doi.org/10.1021/ct400569s

Attributes:

expconst: ***float or int***

Zeta value in exponential term.

emin: ***float or int***

Energy boundary for absorbing potential.

prefac: ***float or int***

Pre-exponential factor in exponential term.

maxval: ***float or int***

Limit on MOCAP strength.

calculate_cap(rt_scf, fock_ao)

Calculates MOCAP damping matrix.

calculate_potential(rt_scf)

Returns complex absorbing potential.

rt_nuclei

Nuclei Object for Real-Time Ehrenfest

class **Nuc**

Takes in mol object

Attributes:

basis: ***str or dict***

labels: ***list***

mass: ***1D ndarray***

pos: ***1D ndarray***

vel: ***1D ndarray***

force: ***1D ndarray***

spin: ***int***

charge: ***int***

get_mol()

Returns new mol object

get_ke()

Calculates nuclear kinetic energy

sample_vel(beta)

blank

get_pos(timestep)

Updates position.

get_vel(timestep)

Updates velocity.

ehrenfest_force

get_force(rt_ehrenfest)

$$\text{Force} = -\frac{\partial \mathbf{E}}{\partial r} = -\frac{\partial \mathbf{V}_{NN}}{\partial r} - \text{Tr}(\mathbf{P}'\frac{d\mathbf{h}'}{dr}) - \frac{1}{2}\text{Tr}(\mathbf{P}'\frac{\partial \mathbf{V}'_{\text{eff}}}{\partial r}) + \text{Tr}(\mathbf{F}'\mathbf{X}^{-1}\frac{d\mathbf{X}}{dr}\mathbf{P}' + \mathbf{P}'\frac{d\mathbf{X}}{dr}\mathbf{X}^{-1}\mathbf{F}') \quad (1)$$

rt_spec

Time Resolved Spectra Functions

abs_spec(time, dipole, filename, kick_str=1, pad=None, damp=None, preprocess_zero=True)

Transforms time-dependent dipole moment into frequency domain, calculates spectrum.

basis_utils

Basis Utility Functions

print2molden(scf, filename=None, mo_coeff=None)

Creates molden file.

noscfbasis(scf, *fragments, reorder=True)

Creates molecular orbital basis from fragment scf objects. If reorder is True the occupied molecular orbitals will appear first.

hirshfeld

Hirshfeld Interface w/ https://github.com/frobnitzem/hirshfeld

hirshfeld_partition(scf, den_ao=None, grids=None, atom_weights=None

Returns density matrix partitioned on numerical integration grid.

get_weights(mol)

Returns atomic weights, calculated from external pyscf.Hirshfeld module.