My G-CTMQC

Generated by Doxygen 1.9.1

Chapter 1

Modules Index

1.1 Modules List

Here is a list of all documented modules with brief descriptions:

Chapter 2

Module Documentation

2.1 analytical_potentials Module Reference

Diabatic low dimensional potentials.

Functions/Subroutines

- subroutine new_model_potentials (Hel, grad_BO, NAC, Q)

  Definition of electronic structure properties, ie, energies, gradients and derivative couplings on model potentials integrated in the G-CTMQC.
- subroutine check_overlap (eigenv1, eigenv2, factor)

  Check that adjacent adiabatic states are continuous in nuclear space.
- subroutine diagonalize (matrix, eigenvalues, eigenvectors)

  Diagonalization of a real symmetric matrix with the Lapac procedure dsyevd.
- subroutine non_adiabatic_couplings (energy, gradients, eigenvectors, couplings)

  Calculation of analytical non-adiabatic couplings.
- subroutine nai_potential (H, R, grad_H, R_crossing)

  Definition of the analytical model potentials for the diatomic molecule NaI following the work of Faist and Levine published in JCP (1976) DOI:10.1063/1.432555.
- subroutine ibr_potential (H, R, grad_H, R_crossing)

  Definition of the analytical model potentials for the diatomic molecule NaI following the work of Guo published in JCP (1993) DOI:10.1063/1.465285.
- subroutine doublewell_potential (H, R, grad_H, R_crossing)

  Definition of the analytical model potentials a double well.
- subroutine tully3 (H, R, grad_H)

  Definition of the analytical model potential Tully #3.
- subroutine subotnikjpca2019 (H, R, grad_H)

  Definition of the analytical model potentials for parallel constant PESs.
- subroutine phenol_potential (H, R, grad_H, R_crossing)

  Definition of the analytical model potentials for the phenol molecule following the work of Faist and Levine published in JCP (1976) DOI:10.1063/1.432555.
- subroutine plot_potential ()

  Plot the adiabatic and diabatic analytical potentials.

2.1.1 Detailed Description

Diabatic low dimensional potentials.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.1.2 Function/Subroutine Documentation

2.1.2.1 check_overlap()

```
subroutine analytical_potentials::check_overlap (
            real(kind=dp), dimension(nstates,nstates), intent(in) eigenv1,
            real(kind=dp), dimension(nstates,nstates), intent(in) eigenv2,
            real(kind=dp), dimension(nstates), intent(inout) factor )
```

Check that adjacent adiabatic states are continuous in nuclear space.

Parameters

`in`	*eigenv1,eigenv2*	eigenstates of the electronic Hamiltonian at adjecent points in nuclear space
`in,out`	*factor*	control factor for the phase relation between adjacent adiabatic states
	sum	scalar product between eigenv1 and eigenv2
	i,j	integer indices

Returns

The values of factor is returned: it is 1 if the two vectors are in phase or -1 if the two vectors are out of phase.

2.1.2.2 diagonalize()

```
subroutine analytical_potentials::diagonalize (
            real(kind=dp), dimension(nstates,nstates), intent(in) matrix,
            real(kind=dp), dimension(nstates), intent(inout) eigenvalues,
            real(kind=dp), dimension(:,:), intent(inout) eigenvectors )
```

Diagonalization of a real symmetric matrix with the Lapac procedure dsyevd.

Parameters

`in`	*matrix*	to be diagonalized
`in,out`	*eigenvalues*	of the matrix
`in,out`	*eigenvectors*	of the matrix

Parameters

	ioerr	control variable for diagonalization errors
	lwork	dimension of the array work
	dim_work	temporary dimension of the array work
	liwork	dimension of the array iwork
	dim_iwork	temporary dimension of the array iwork
	iwork	integer array
	work	double precision array

Returns

The eigenvalues and eigenvectors of the matrix are returned.

2.1.2.3 doublewell_potential()

```
subroutine analytical_potentials::doublewell_potential (
            real(kind=dp), dimension(nstates,nstates), intent(inout) H,
            real(kind=dp), intent(in) R,
            real(kind=dp), dimension(nstates,nstates), intent(inout) grad_H,
            real(kind=dp), intent(inout), optional R_crossing )
```

Definition of the analytical model potentials a double well.

Parameters

`in,out`	*H*	electronic Hamiltonian in the diabatic basis
`in,out`	*grad_H*	gradient of the electronic Hamiltonian in the diabatic basis
`in`	*R*	nuclear position
	KX,DELTA,X1,X2,X3,GAMMA,ALPHA	parameters of the potentials: Note that the used units are atomic units

Returns

The electronic Hamiltonian and its nuclear gradients are returned.

2.1.2.4 ibr_potential()

```
subroutine analytical_potentials::ibr_potential (
            real(kind=dp), dimension(nstates,nstates), intent(inout) H,
            real(kind=dp), intent(in) R,
            real(kind=dp), dimension(nstates,nstates), intent(inout) grad_H,
            real(kind=dp), intent(inout), optional R_crossing )
```

Definition of the analytical model potentials for the diatomic molecule NaI following the work of Guo published in JCP (1993) DOI:10.1063/1.465285.

Parameters

in,out	*H*	electronic Hamiltonian in the diabatic basis
in,out	*grad_H*	gradient of the electronic Hamiltonian in the diabatic basis
in	*R*	nuclear position
	A0,alpha0,r0,A1,alpha1,r1,D,A2,alpha2,B2,beta2,r12	parameters of the potentials: Note that the used units are atomic units

Returns

The electronic Hamiltonian and its nuclear gradients are returned.

2.1.2.5 nai_potential()

```
subroutine analytical_potentials::nai_potential (
            real(kind=dp), dimension(nstates,nstates), intent(inout) H,
            real(kind=dp), intent(in) R,
            real(kind=dp), dimension(nstates,nstates), intent(inout) grad_H,
            real(kind=dp), intent(inout), optional R_crossing )
```

Definition of the analytical model potentials for the diatomic molecule NaI following the work of Faist and Levine published in JCP (1976) DOI:10.1063/1.432555.

Parameters

in,out	*H*	electronic Hamiltonian in the diabatic basis
in,out	*grad_H*	gradient of the electronic Hamiltonian in the diabatic basis
in	*R*	nuclear position
	Acov,Bcov,rhocov,Ccov,Aion,Bion,rhoion,Cion,alphaMp,alphaXm,Eth,A,rho	parameters of the potentials: Note that the used units are electronvolts and and angstroms in the original definition

Returns

The electronic Hamiltonian and its nuclear gradients are returned.

2.1.2.6 new_model_potentials()

```
subroutine analytical_potentials::new_model_potentials (
            real(kind=dp), dimension(nstates,nstates), intent(inout) Hel,
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(inout) grad_BO,
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(inout) NAC,
            real(kind=dp), dimension(n_dof), intent(in) Q )
```

Definition of electronic structure properties, ie, energies, gradients and derivative couplings on model potentials integrated in the G-CTMQC.

Parameters

in,out	*Hel*	electronic Hamiltonian at the trajectory position
in,out	*grad_BO*	gradient of the adiabatic energy at the trajectory position
in,out	*NAC*	non-adiabatic coupling vector at the trajectory position
in	*Q*	trajectory position
	Ebo	adiabatic energy at the trajectory position
	grad_Hel	gradient of the electronic Hamiltonian at the trajectory position
	U	transformation matrix from the diabatic to the adiabatic basis
	factor	control factor for the phase relation between adjacent adiabatic states
	delta	spatial increment to compute numerical derivatives
	i_dof	index running on the n_dof degrees of freedom
	ix,i	integer indices

Returns

The values of the adiabatic energies are returned as the diagonal elements of Hel; the gradient of the adiabatic energies and the non-adiabatic couplings are returned in grad_BO and NAC.

2.1.2.7 non_adiabatic_couplings()

```
subroutine analytical_potentials::non_adiabatic_couplings (
            real(kind=dp), dimension(nstates), intent(in) energy,
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(in) gradients,
            real(kind=dp), dimension(nstates,nstates), intent(in) eigenvectors,
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(inout) couplings )
```

Calculation of analytical non-adiabatic couplings.

Parameters

in	*energy*	of the adiabatic states
in	*gradients*	of the adiabatic energies
in	*eigenvectors*	of the electronic Hamiltonian
in,out	*couplings*	matrix representing the non-adiabatic couplings
	i,j,k,l	integer indices

Returns

The matrix of non-adiabatic couplings is returned.

2.1.2.8 phenol_potential()

```
subroutine analytical_potentials::phenol_potential (
            real(kind=dp), dimension(nstates,nstates), intent(inout) H,
            real(kind=dp), intent(in) R,
            real(kind=dp), dimension(nstates,nstates), intent(inout) grad_H,
            real(kind=dp), intent(inout), optional R_crossing )
```

Definition of the analytical model potentials for the phenol molecule following the work of Faist and Levine published in JCP (1976) DOI:10.1063/1.432555.

Parameters

in,out	*H*	electronic Hamiltonian in the diabatic basis
in,out	*grad_H*	gradient of the electronic Hamiltonian in the diabatic basis
in	*R*	nuclear position
	Acov,Bcov,rhocov,Ccov,Aion,Bion,rhoion,Cion,alpha-Mp,alphaXm,Eth,A,rho	parameters of the potentials: Note that the used units are electronvolts and and angstroms in the original definition

Returns

The electronic Hamiltonian and its nuclear gradients are returned.

2.1.2.9 plot_potential()

```
subroutine analytical_potentials::plot_potential
```

Plot the adiabatic and diabatic analytical potentials.

Parameters

Rmin,Rmax	limits of the domain to plot the potentials
V	values of the diabatic potentials
epsBO	values of the adiabatic potentials
nacv	values of the non-adiabatic couplings
grad_V	values of the gradients of the diabatic potentials
U	transformation matrix from the diabatic to the adiabatic basis at the current position
save_U	transformation matrix from the diabatic to the adiabatic basis at the previous position
factor	control factor for the phase relation between adjacent adiabatic states
npoints	number of grid points in nuclear space to plot the potentials
delta	spatial increment to compute numerical derivatives
R	
ground_states_density	initial nuclear density consistent with the distribution of classical initial positions and momenta
ios	control variable for output errors
ix,i	integer indices

2.1.2.10 subotnikjpca2019()

```
subroutine analytical_potentials::subotnikjpca2019 (
            real(kind=dp), dimension(nstates,nstates), intent(inout) H,
            real(kind=dp), intent(in) R,
            real(kind=dp), dimension(nstates,nstates), intent(inout) grad_H )
```

Definition of the analytical model potentials for parallel constant PESs.

Parameters

in,out	*H*	electronic Hamiltonian in the diabatic basis
in,out	*grad↩_H*	gradient of the electronic Hamiltonian in the diabatic basis
in	*R*	nuclear position
	A,B	parameters of the potentials: Note that the used units are atomic units

Returns

The electronic Hamiltonian and its nuclear gradients are returned.

2.1.2.11 tully3()

```
subroutine analytical_potentials::tully3 (
            real(kind=dp), dimension(nstates,nstates), intent(inout) H,
            real(kind=dp), intent(in) R,
            real(kind=dp), dimension(nstates,nstates), intent(inout) grad_H )
```

Definition of the analytical model potential Tully #3.

Parameters

in,out	*H*	electronic Hamiltonian in the diabatic basis
in,out	*grad↩_H*	gradient of the electronic Hamiltonian in the diabatic basis
in	*R*	nuclear position
	a,b,c	parameters of the potentials: Note that the used units are atomic units

Returns

The electronic Hamiltonian and its nuclear gradients are returned.

2.2 atomic_masses Module Reference

The module defines nuclear masses in atomic units.

Variables

- real(kind=dp), parameter m_hp = 1836.0_dp

  Proton mass in atomic units.
- real(kind=dp), parameter m_na = 22.989769_dp

  Na mass in atomic mass units.
- real(kind=dp), parameter m_i = 126.90447_dp

  I mass in atomic mass units.
- real(kind=dp), parameter m_br = 79.9040_dp

  Br mass in atomic mass units.
- real(kind=dp), parameter m_h = 1.007825_dp

  H mass in atomic mass units.
- real(kind=dp), parameter m_o = 15.9994_dp

  O mass in atomic mass units.

2.2.1 Detailed Description

The module defines nuclear masses in atomic units.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.3 classical_evolution Module Reference

The module contains a collection of subroutines used in the classical evolution of the nuclei.

Functions/Subroutines

- subroutine update_position (x, v)

  Update of the classical positions according to the velocity Verlet algorithm.
- subroutine update_velocity (v, force)

  Update of the classical velocities according to the velocity Verlet algorithm.
- subroutine non_adiabatic_force (coeff, force, acc_force, k_li, trajlabel)

  Definition of the classical nuclear force depending on the type of calculation that is executed.

2.3.1 Detailed Description

The module contains a collection of subroutines used in the classical evolution of the nuclei.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.3.2 Function/Subroutine Documentation

2.3.2.1 non_adiabatic_force()

```
subroutine classical_evolution::non_adiabatic_force (
            complex(kind=qp), dimension(nstates), intent(in) coeff,
            real(kind=dp), dimension(n_dof), intent(inout) force,
            real(kind=dp), dimension(n_dof,nstates), intent(in) acc_force,
            real(kind=dp), dimension(nstates,nstates), intent(in) k_li,
            integer, intent(in) trajlabel )
```

Definition of the classical nuclear force depending on the type of calculation that is executed.

Parameters

in	*trajlabel*	label indicating the trajectory number in the swarm
in	*coeff*	coefficients of the expansion of the electronic time-dependent wavefunction in the basis used for the dynamics
in	*acc_force*	gradient of the adiabatic or diabatic force accumulated over time along the trajectory trajlabel
in	*k_li*	quantity related to the quantum momentum and responsible for decoherence; it is identically zero for Ehrenfest and surface hopping calculations
in,out	*force*	classical force used to evolve the trajectory trajlabel
	i,j	integer indices
	i_dof	index running on the n_dof degrees of freedom
	check	control variable for allocation errors
	my_rho	temporary array of the electronic density matrix

Returns

The value of the classical force at the position of the trajectory is returned.

2.3.2.2 update_position()

```
subroutine classical_evolution::update_position (
            real(kind=dp), dimension(n_dof), intent(inout) x,
            real(kind=dp), dimension(n_dof), intent(in) v )
```

Update of the classical positions according to the velocity Verlet algorithm.

Parameters

in,out	*x*	nuclear position
in	*v*	nuclear velocity
	my↩_x	temporary nuclear position for internal calculations

Returns

The updated nuclear position is returned.

2.3.2.3 update_velocity()

```
subroutine classical_evolution::update_velocity (
            real(kind=dp), dimension(n_dof), intent(inout) v,
            real(kind=dp), dimension(n_dof), intent(in) force )
```

Update of the classical velocities according to the velocity Verlet algorithm.

Parameters

in	*force*	nuclear force
in,out	*v*	nuclear velocity
	my↩ _v	temporary nuclear velocity for internal calculations

Returns

The updated nuclear velocity is returned.

2.4 coefficients_evolution Module Reference

Evolution of the electronic coefficients.

Functions/Subroutines

- subroutine **evolve_coeff** (v, coeff, k_li, E_old, NAC_old, trajlabel)
- subroutine rk4_coeff (v, coeff, k_li, E, NAC, trajlabel)

  Numerical integration of the non-linear differential equation describing the electronic evolution of the coefficients.
- complex(kind=qp) function cdot (state, kfunction, v, coeff, k_li, E_int, NAC_int, trajlabel)

  Total time derivative of the electronic coefficients as given in the Ehrenfest algorithm, surface hopping algorithm, and CT-MQC.

2.4.1 Detailed Description

Evolution of the electronic coefficients.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.4.2 Function/Subroutine Documentation

2.4.2.1 cdot()

```
complex(kind=qp) function coefficients_evolution::cdot (
            integer, intent(in) state,
            complex(kind=qp), intent(in) kfunction,
            real(kind=dp), dimension(n_dof), intent(in) v,
            complex(kind=qp), dimension(nstates), intent(in) coeff,
            real(kind=dp), dimension(nstates,nstates), intent(in) k_li,
            real(kind=dp), dimension(nstates), intent(in) E_int,
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(in) NAC_int,
            integer, intent(in) trajlabel )
```

Total time derivative of the electronic coefficients as given in the Ehrenfest algorithm, surface hopping algorithm, and CT-MQC.

Parameters

in	*state*	electronic state for which the time derivative of the coefficients is computer
in	*trajlabel*	label of the trajectory along which the equation is integrated
in	*v*	velocity of trajectory along which the time derivative of the coefficient is calculated
in	*k_li*	term accounting for decoherence effects in CT-MQC
in	*coeff*	electronic coefficientes
in	*kfunction*	function appearing in the expression of the time increment
	cdot	time derivative of the coefficient
	nonadiabatic_sum	off-diagonal contribution to the time derivative of the coefficients
	my_coeff	local temporary values of the coefficients
	i	integer index
	my_gap	energy-gap threshold between the spin-diabatic states to tune the effect of the spin-orbit coupling

Returns

The values of the time derivative of the electronic coefficientes is returned.

2.4.2.2 rk4_coeff()

```
subroutine coefficients_evolution::rk4_coeff (
            real(kind=dp), dimension(n_dof), intent(in) v,
            complex(kind=qp), dimension(nstates), intent(inout) coeff,
            real(kind=dp), dimension(nstates,nstates), intent(in) k_li,
            real(kind=dp), dimension(nstates), intent(in) E,
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(in) NAC,
            integer, intent(in) trajlabel )
```

Numerical integration of the non-linear differential equation describing the electronic evolution of the coefficients.

Parameters

in	*trajlabel*	label of the trajectory along which the equation is integrated
in	*v*	velocity of trajectory along which the equation is integrated

Parameters

`in`	*k_li*	term accounting for decoherence effects in CT-MQC
`in,out`	*coeff*	electronic coefficientes
	i	integer index
	k1,k2,k3,k4	functions appearing in the expression of the time increment
	kfunction	function appearing in the expression of the time increment
	my_coeff	local temporary values of the coefficients
	normalization	norm of the electronic wavefunction after a time step

Returns

The values of the electronic coefficientes are returned after one step of dynamics.

2.5 coherence_corrections Module Reference

Calculations of quantities for decoherence corrections in CT-MQC.

Functions/Subroutines

- subroutine accumulated_boforce (coeff, force, trajlabel)

  The adiabatic (or spin-(a)diabatic) force is integrated in time along a trajectory.
- subroutine quantum_momentum (Rcl, acc_force, BOsigma, k_li, qmom, qmom_type)

  Calculation of the quantum momentum by reconstructing the nuclear density as a sum of Gaussians centered at the positions of the trajectories.
- subroutine acc_force_ec (Vcl, coeff, acc_force, acc_force_E)

  The accumulated force is corrected to satisfy energy conservation.
- subroutine quantum_momentum (Rcl, acc_force, BOsigma, k_li, qmom, qmom_type)

  Calculation of the quantum momentum by reconstructing the nuclear density as a sum of Gaussians centered at the positions of the trajectories.

2.5.1 Detailed Description

Calculations of quantities for decoherence corrections in CT-MQC.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.5.2 Function/Subroutine Documentation

2.5.2.1 acc_force_ec()

```
subroutine coherence_corrections::acc_force_ec (
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Vcl,
            complex(kind=qp), dimension(ntraj,nstates), intent(in) coeff,
            real(kind=dp), dimension(ntraj,n_dof,nstates), intent(in) acc_force,
            real(kind=dp), dimension(ntraj,n_dof,nstates), intent(inout) acc_force_E )
```

The accumulated force is corrected to satisfy energy conservation.

Parameters

in	*Vcl*	classical velocities
in	*coeff*	electronic coefficientes
in	*acc_force*	accumulated force along the trajectories
in,out	*acc_force↩ _E*	new accumulated force
	i_dof	index running on the n_dof degrees of freedom
	i_traj	index running on the n_traj degrees number of trajectories
	istate	index running on the nstates of electronic states
	threshold	electronic population threshold to compute the acc forces
	R_threshold	velocity threshold to compute the new acc forces
	nvec	vector in the direction of the new acc force

2.5.2.2 accumulated_boforce()

```
subroutine coherence_corrections::accumulated_boforce (
            complex(kind=qp), dimension(nstates), intent(in) coeff,
            real(kind=dp), dimension(n_dof,nstates), intent(inout) force,
            integer, intent(in) trajlabel )
```

The adiabatic (or spin-(a)diabatic) force is integrated in time along a trajectory.

Parameters

in	*trajlabel*	label of the trajectory along which the equation is integrated
in	*coeff*	electronic coefficientes
in,out	*force*	integrated force along the trajectory
	i	integer index
	i_dof	index running on the n_dof degrees of freedom
	check	control variable for allocation errors
	threshold	electronic population threshold to accumate the force
	mean_force	average electronic force weighted by the electronic population

Returns

The value of the adiabatic (or spin-(a)diabatic) force is returned if the electronic population of the corresponding state is larger than threshold and smaller that one minus the threshold.

2.5.2.3 quantum_momentum() [1/2]

```
subroutine coherence_corrections::quantum_momentum (
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Rcl,
            real(kind=dp), dimension(ntraj,n_dof,nstates), intent(in) acc_force,
            complex(kind=qp), dimension(ntraj,nstates,nstates), intent(in) BOsigma,
            real(kind=dp), dimension(ntraj,nstates,nstates), intent(inout) k_li,
            real(kind=dp), dimension(:,:,:), allocatable qmom,
            integer, dimension(n_dof,ntraj,npairs), intent(inout) qmom_type )
```

Calculation of the quantum momentum by reconstructing the nuclear density as a sum of Gaussians centered at the positions of the trajectories.

Parameters

`in`	*BOsigma*	electronic density matrix
`in`	*Rcl*	positions of the trajectories
`in`	*acc_force*	force accumulated along the trajectory
`in,out`	*k_li*	term accounting for decoherence effects in CT-MQC
	itraj,jtraj	indices running on the Ntraj trajectories
	i_dof	index running on the n_dof degrees of freedom
	index_ij	index running on the pairs of electronic states
	istate,jstate	indices running on the electronic states
	gamma	variances of the Gaussians centered at the positions of the trajectories and used to reconstruct the nuclear density
	g_i	sum of Gaussians
	prod_g_i	product of one-dimensional Gaussians to construct a multi-dimensional Gaussian
	w_ij	see paper DOI:...
	slope_i	slope of the quantum momentum when it is approximated as a linear function
	ratio	y-intercept when the quantum momentum is approximated as a linear function
	num_old	numerator in the expression of the y-intercept to approximate the quantum momentum as a linear function when the condition of no-population-transfer between two electronic states is imposed for zero values of the non-adiabatic couplings
	num_new	numerator in the analytical expression of the y-intercept to approximate the quantum momentum as a linear function
	num	numerator in the expression of the y-intercept of the linear quantum momentum
	denom	denominator in the expression of the y-intercept of the linear quantum momentum
	qmom	quantum momentum
	threshold	for the selection of the num_old or num_old (M_parameter $*$ threshold is the applied distance criterion)

Returns

The value of k_li is returned.

This is calculated each time; Consider saving them with a save varaiable

Sigma is standard deviation of wave function; not density, Thus, we need factor 2 in alpha (slope) and in the gauss later

Define alph (slope) for models/special cases

CALCULATE "gaus matrix" As above, the factor 2 is taken care of We take advantage of the symmetry for efficiency

DENSITY AT EACH TRAJECTORY

standard intercept R_ic Normal intercept, without respecting pop consersevation at NAC=0 fancy intercept R_fi Intercept respecting pop consersevation at NAC=0

Periodicity part 1

Quantum momentum Start with 0 Priority: fancy intercept If too far away: normal intercept If still too far away it stays zero

double fancy intercept R_fi Intercept respecting pop consersevation at NAC=0 only for same Carsten

Quantum momentum Start with 0, will stay 0 if Carsten is zero

Intercepts selected by Carsten of traj/state This may not work for older compilers

dont forget the slope! factor 0.5 from using the density insead of wavefunction in def of qmom

output k_li

clean up crew, could be skipped I think

2.5.2.4 quantum_momentum() `[2/2]`

```
subroutine coherence_corrections::quantum_momentum (
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Rcl,
            real(kind=dp), dimension(ntraj,n_dof,nstates), intent(in) acc_force,
            complex(kind=qp), dimension(ntraj,nstates,nstates), intent(in) BOsigma,
            real(kind=dp), dimension(ntraj,nstates,nstates), intent(inout) k_li,
            real(kind=dp), dimension(n_dof,ntraj,npairs), intent(inout) qmom,
            integer, dimension(n_dof,ntraj,npairs), intent(inout) qmom_type )
```

Calculation of the quantum momentum by reconstructing the nuclear density as a sum of Gaussians centered at the positions of the trajectories.

Parameters

`in`	*BOsigma*	electronic density matrix
`in`	*Rcl*	positions of the trajectories
`in`	*acc_force*	force accumulated along the trajectory
`in,out`	*k_li*	term accounting for decoherence effects in CT-MQC
	itraj,jtraj	indices running on the Ntraj trajectories
	i_dof	index running on the n_dof degrees of freedom
	index_ij	index running on the pairs of electronic states
	istate,jstate	indices running on the electronic states
	gamma	variances of the Gaussians centered at the positions of the trajectories and used to reconstruct the nuclear density
	g_i	sum of Gaussians
	prod_g_i	product of one-dimensional Gaussians to construct a multi-dimensional Gaussian
	w_ij	see paper DOI:...
	slope_i	slope of the quantum momentum when it is approximated as a linear function
	ratio	y-intercept when the quantum momentum is approximated as a linear function
	num_old	numerator in the expression of the y-intercept to approximate the quantum momentum as a linear function when the condition of no-population-transfer between two electronic states is imposed for zero values of the non-adiabatic couplings
	num_new	numerator in the analytical expression of the y-intercept to approximate the quantum momentum as a linear function
	num	numerator in the expression of the y-intercept of the linear quantum momentum
	denom	denominator in the expression of the y-intercept of the linear quantum momentum
	qmom	quantum momentum

Returns

The value of k_li is returned.

2.6 electronic_problem Module Reference

On-the-fly electronic-structure calculations.

Functions/Subroutines

- subroutine boproblem (Q, trajlabel)

  Electronic energies (adiabatic or spin-(a)diabatic), forces and non-adiabatic couplings are compueted at the trajectory position.
- subroutine check_nac_overlap (NACij, NACij_old)

  Arbitrary sign changes in the NACs due to the diagonalization are fixed.

2.6.1 Detailed Description

On-the-fly electronic-structure calculations.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.6.2 Function/Subroutine Documentation

2.6.2.1 boproblem()

```
subroutine electronic_problem::boproblem (
            real(kind=dp), dimension(n_dof), intent(in) Q,
            integer, intent(in) trajlabel )
```

Electronic energies (adiabatic or spin-(a)diabatic), forces and non-adiabatic couplings are compueted at the trajectory position.

Parameters

in	*trajlabel*	label of the trajectory
in	*Q*	position of the trajectory
	istate	integer index running over the electronic states
	V	array of (a)diabatic Hamiltonian
	G	array of gradients of the (a)diabatic Hamiltonian
	NAC	array of non-adiabatic couplings
	NAC_old	array of non-adiabatic couplings at previous time step
	initialize	logical to initialize the QMLLibrary potentials

Returns

Energies, forces and non-adiabatic couplings are stored in the arrays BOenergy, BOforce, coup.

2.6.2.2 check_nac_overlap()

```
subroutine electronic_problem::check_nac_overlap (
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(inout) NACij,
            real(kind=dp), dimension(nstates,nstates,n_dof), intent(in) NACij_old )
```

Arbitrary sign changes in the NACs due to the diagonalization are fixed.

Parameters

in	*NACij_old*	NAC between two electronic states at time t
in,out	*NACij*	NAC between two electronic states at time t+dt param snac_old magnitude of NAC vector at time t param snac magnitude of NAC vector at time t+dt ovlp overlap between NAC_ij(t) and NAC_ij(t+dt) eps threshold for the overlap

Returns

NAC at time t+dt with right sign respect to previous timestep

2.7 kinds Module Reference

Definiton of kinds.

Variables

- integer, parameter dp =kind(0.0D0)

  Indicator for real double precision.
- integer, parameter qp =selected_real_kind(12)

  Indicator for real quadrupole precision.

2.7.1 Detailed Description

Definiton of kinds.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.8 output Module Reference

Output subroutines that print: electronic populations and coherences as functions of time; electronic coefficients as functions of positions at different time steps; positions, momenta and energies at different time steps.

Functions/Subroutines

- subroutine plot (BOsigma, Rcl, Vcl, time)

  Subroutine which writes electronic populations and coherences in output and calls additional output subroutines.
- subroutine plot_coefficients (BOsigma, Rcl, time)

  Subroutine which writes electronic coeffecients as functions of the trajectory positions at some selected time steps along the dynamics.
- subroutine plot_r_p_e (Rcl, Vcl, time)

  Subroutine which writes electronic coeffecients as functions of the trajectory positions at some selected time steps along the dynamics.
- subroutine plot_qmom (Rcl, qmom, qmom_type, time)

  Subroutine which outputs information about the spurious transfer condition (STC), ie, sum_traj Q(fl-fk)rho_ll∗rho_kk, and dE/dt.
- subroutine plot_stc (k_ll, BOsigma, acc_force_E, Vcl, time)

  Subroutine which outputs information about the spurious transfer condition (STC), ie, sum_traj Q(fl-fk)rho_ll∗rho_kk, and dE/dt.
- subroutine compute_energy (my_rho, e_BO, trajlabel)

  The subroutine computes the expectation value of the electronic Hamiltonian on the time-dependent electronic wavefunction, yielding the gauge-invariant part of the TDPES in CT-MQC or the mean Ehrenfest potential.
- subroutine initialize_output

  The files where electronic populations, coherences and the energy of the ensemble of trajectories are written are opened in this subroutine.
- subroutine finalize_output

  The files where electronic populations, coherences and the energy of the ensemble of trajectories are written are closed in this subroutine.

2.8.1 Detailed Description

Output subroutines that print: electronic populations and coherences as functions of time; electronic coefficients as functions of positions at different time steps; positions, momenta and energies at different time steps.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.8.2 Function/Subroutine Documentation

2.8.2.1 compute_energy()

```
subroutine output::compute_energy (
            complex(kind=qp), dimension(nstates,nstates), intent(in) my_rho,
            real(kind=dp), dimension(nstates), intent(in) e_BO,
            integer, intent(in) trajlabel )
```

The subroutine computes the expectation value of the electronic Hamiltonian on the time-dependent electronic wavefunction, yielding the gauge-invariant part of the TDPES in CT-MQC or the mean Ehrenfest potential.

Parameters

in	*trajlabel*	label of the trajectory
in	*my_rho*	electronic density matrix
in	*e_BO*	adiabatic or spin-(a)diabatic energy
	i	integer index

Returns

The value of the TDPES is returned, where "TDPES" means either the gauge-invariant part of the TDPES in CT-MQC or the mean Ehrenfest potential.

2.8.2.2 initialize_output()

```
subroutine output::initialize_output
```

The files where electronic populations, coherences and the energy of the ensemble of trajectories are written are opened in this subroutine.

Parameters

ios	control variable for output errors

2.8.2.3 plot()

```
subroutine output::plot (
            complex(kind=qp), dimension(ntraj,nstates,nstates), intent(in) BOsigma,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Rcl,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Vcl,
            integer, intent(in) time )
```

Subroutine which writes electronic populations and coherences in output and calls additional output subroutines.

Parameters

in	*time*	time step
in	*Rcl*	positions of the trajectories
in	*Vcl*	velocities of the trajectories
in	*BOsigma*	electronic density matrix
	i,j	integer indices
	itraj	integer index running over the Ntraj trajectories
	index_ij	integer index running over the pairs of electronic states

Returns

Energies, forces and non-adiabatic couplings are stored in the arrays BOenergy, BOforce, coup.

2.8.2.4 plot_coefficients()

```
subroutine output::plot_coefficients (
            complex(kind=qp), dimension(ntraj,nstates,nstates), intent(in) BOsigma,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Rcl,
            integer, intent(in) time )
```

Subroutine which writes electronic coeffecients as functions of the trajectory positions at some selected time steps along the dynamics.

Parameters

in	*time*	time step
in	*Rcl*	positions of the trajectories
in	*BOsigma*	electronic density matrix
	idx	index labelling the output files from 000 to 999
	filename	name of the output file
	itraj	integer index running over the Ntraj trajectories
	ios	control variable for output errors

Returns

In the directory coeff the files coeff.XXX.dat are created, labelled from 000 to 999 (those indices label the time steps).

2.8.2.5 plot_qmom()

```
subroutine output::plot_qmom (
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Rcl,
            real(kind=dp), dimension(n_dof,ntraj,npairs), intent(in) qmom,
            integer, dimension(n_dof,ntraj,npairs), intent(in) qmom_type,
            integer, intent(in) time )
```

Subroutine which outputs information about the spurious transfer condition (STC), ie, sum_traj Q(fl-fk)rho_ll∗rho_kk, and dE/dt.

Parameters

in	*time*	time step

2.8.2.6 plot_r_p_e()

```
subroutine output::plot_r_p_e (
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Rcl,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Vcl,
            integer, intent(in) time )
```

Subroutine which writes electronic coeffecients as functions of the trajectory positions at some selected time steps along the dynamics.

Parameters

in	*time*	time step
in	*Rcl*	positions of the trajectories
in	*Vcl*	velocities of the trajectories
	idx	index labelling the output files from 000 to 999
	filename	name of the output file
	itraj	integer index running over the Ntraj trajectories
	ios	control variable for output errors

Returns

In the directory trajectories the files RPE.XXX.dat are created, labelled from 000 to 999 (those indices label the time steps).

2.8.2.7 plot_stc()

```
subroutine output::plot_stc (
            real(kind=dp), dimension(ntraj,nstates,nstates), intent(in) k_ll,
            complex(kind=qp), dimension(ntraj,nstates,nstates), intent(in) BOsigma,
            real(kind=dp), dimension(ntraj,n_dof,nstates), intent(in) acc_force_E,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) Vcl,
            integer, intent(in) time )
```

Subroutine which outputs information about the spurious transfer condition (STC), ie, sum_traj Q(fl-fk)rho_ll∗rho_kk, and dE/dt.

Parameters

in	*time*	time step
in	*BOsigma*	electronic density matrix
in	*k_ll*	qmom∗acc_force
	itraj	integer index running over the Ntraj trajectories
	ios	control variable for output errors

Returns

In main directory STC.dat is created

2.9 shopping Module Reference

Surface hopping tools to compute the hop probability, the active state the energy rescaling after the hop, and the energy decoherence correction.

Functions/Subroutines

- subroutine hopping (my_rho, v, r, trajlabel, k_li)

  The hopping probability for the surface hopping procedure is computed according to the fewest switches procedure.
- subroutine choose_bostate (v, r, hop_prob, trajlabel, k_li, my_rho)

  According to the fewest switches algorithm, the new active state is selected.
- subroutine momentum_correction (v, r, old_occ_state, trajlabel, k_li, my_rho)

  Nuclear velocities are rescaled along the direction of the non-adiabatic couplings to impose energy conservation in case a hop to a new potential energy surface has occured.
- subroutine decoherence_coorection (coeff, v, trajlabel)

  Energy decoherence corrections are applied to surface hopping coefficients according to Granucci and Persico JCP 2007 DOI: 10.1063/1.2715585.
- subroutine **xi_for_model_system** (dist, deltaE, xi, Rcl, Vcl, trajlabel)
- subroutine hopping (my_rho, v, r, trajlabel)

  The hopping probability for the surface hopping procedure is computed according to the fewest switches procedure.
- subroutine choose_bostate (v, hop_prob, trajlabel)

  According to the fewest switches algorithm, the new active state is selected.
- subroutine momentum_correction (v, old_occ_state, trajlabel)

  Nuclear velocities are rescaled along the direction of the non-adiabatic couplings to impose energy conservation in case a hop to a new potential energy surface has occured.

2.9.1 Detailed Description

Surface hopping tools to compute the hop probability, the active state the energy rescaling after the hop, and the energy decoherence correction.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.9.2 Function/Subroutine Documentation

2.9.2.1 choose_bostate() [1/2]

```
subroutine shopping::choose_bostate (
            real(kind=dp), dimension(n_dof), intent(inout) v,
            real(kind=dp), dimension(nstates), intent(inout) hop_prob,
            integer, intent(in) trajlabel )
```

According to the fewest switches algorithm, the new active state is selected.

Parameters

in	*trajlabel*	label of the trajectory
in,out	*v*	nuclear velocity
in,out	*hop_prob*	hopping probability for each electronic state
	myrand	random number
	prob_sum	cumulative hopping probability
	i_state,j_state	integer indices running over the nstates electronic states
	old_occ_state	previous active state

Returns

The value of the hopping probability is returned, along with the new nuclear velocity in case a hop occurred.

2.9.2.2 choose_bostate() [2/2]

```
subroutine shopping::choose_bostate (
            real(kind=dp), dimension(ntraj,n_dof), intent(inout) v,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) r,
            real(kind=dp), dimension(nstates), intent(inout) hop_prob,
            integer, intent(in) trajlabel,
            real(kind=dp), dimension(nstates,nstates), intent(in), optional k_li,
            complex(kind=dp), dimension(nstates,nstates), intent(in), optional my_rho )
```

According to the fewest switches algorithm, the new active state is selected.

Parameters

in	*trajlabel*	label of the trajectory
in,out	*v*	nuclear velocity
in,out	*hop_prob*	hopping probability for each electronic state
	myrand	random number
	prob_sum	cumulative hopping probability
	i_state,j_state	integer indices running over the nstates electronic states
	old_occ_state	previous active state

Returns

The value of the hopping probability is returned, along with the new nuclear velocity in case a hop occurred.

2.9.2.3 decoherence_coorection()

```
subroutine shopping::decoherence_coorection (
            complex(kind=qp), dimension(nstates), intent(inout) coeff,
            real(kind=dp), dimension(n_dof), intent(in) v,
            integer trajlabel )
```

Energy decoherence corrections are applied to surface hopping coefficients according to Granucci and Persico JCP 2007 DOI: 10.1063/1.2715585.

Parameters

in,out	*coeff*	electronic coefficients
in	*v*	nuclear velocity
	decay_time	characteristic time over which the electronic coeffecients of the non-activate states are exponentially damped
	deltaE	potential energy difference between the active states and the other electronic states
	kinetic_energy	nuclear kinetic energy along the trajectory
	sum_rho	sum of the populations of the non-active states
	i_dof	integer index running over the n_dof degrees of freedom
	i_state	integer index running over the nstates electronic states
	trajlabel	label of the trajectory

Returns

The value of the new nuclear velocity is returned in case a hop occurred.

2.9.2.4 hopping() [1/2]

```
subroutine shopping::hopping (
            complex(kind=dp), dimension(nstates,nstates), intent(in) my_rho,
            real(kind=dp), dimension(n_dof), intent(inout) v,
            real(kind=dp), dimension(n_dof), intent(in) r,
            integer, intent(in) trajlabel )
```

The hopping probability for the surface hopping procedure is computed according to the fewest switches procedure.

Parameters

in	*trajlabel*	label of the trajectory
in	*my_rho*	electronic density matrix
in,out	*v*	nuclear velocity
	i_state	integer index running over the nstates electronic states
	i_dof	integer index running over the n_dof degrees of freedom
	scal2	scalar product beteween the nuclear velocity and the non-adiabatic coupling
	Re_rhoij	real part of the elememts of the electronic density matrix
	rhojj	population of the electronic states
	hop_prob	hopping probability for each electronic state

Returns

The value of the nuclear velocity is returned, and it is modified to impose energy conservation if a hop occurred.

2.9.2.5 hopping() [2/2]

```
subroutine shopping::hopping (
            complex(kind=dp), dimension(nstates,nstates), intent(in) my_rho,
            real(kind=dp), dimension(ntraj,n_dof), intent(inout) v,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) r,
            integer, intent(in) trajlabel,
            real(kind=dp), dimension(nstates,nstates), intent(in), optional k_li )
```

The hopping probability for the surface hopping procedure is computed according to the fewest switches procedure.

Parameters

in	*trajlabel*	label of the trajectory
in	*my_rho*	electronic density matrix
in,out	*v*	nuclear velocity
	i_state	integer index running over the nstates electronic states
	i_dof	integer index running over the n_dof degrees of freedom
	scal2	scalar product beteween the nuclear velocity and the non-adiabatic coupling
	Re_rhoij	real part of the elememts of the electronic density matrix
	rhojj	population of the electronic states
	hop_prob	hopping probability for each electronic state

Returns

The value of the nuclear velocity is returned, and it is modified to impose energy conservation if a hop occurred.

2.9.2.6 momentum_correction() [1/2]

```
subroutine shopping::momentum_correction (
            real(kind=dp), dimension(n_dof), intent(inout) v,
            integer, intent(in) old_occ_state,
            integer, intent(in) trajlabel )
```

Nuclear velocities are rescaled along the direction of the non-adiabatic couplings to impose energy conservation in case a hop to a new potential energy surface has occured.

Parameters

in,out	*v*	nuclear velocity
in	*trajlabel*	label of the trajectory
in	*old_occ_state*	previous active state

Parameters

	deltaE	potential energy difference between the old and the new electronic states
	scal1	squared modulus of the non-adiabatic couplings divided by the nuclear mass
	scal2	scalar product betweem the nuclear velocity and the non-adiabatic couplings
	energy_check	criterion to identify the possibility of jump
	scaling_factor	factor to rescal the velocities along the non-adiabatic couplings
	i_dof	integer index running over the n_dof degrees of freedom

Returns

The value of the new nuclear velocity is returned in case a hop occurred.

2.9.2.7 momentum_correction() [2/2]

```
subroutine shopping::momentum_correction (
            real(kind=dp), dimension(ntraj,n_dof), intent(inout) v,
            real(kind=dp), dimension(ntraj,n_dof), intent(in) r,
            integer, intent(in) old_occ_state,
            integer, intent(in) trajlabel,
            real(kind=dp), dimension(nstates,nstates), intent(in), optional k_li,
            complex(kind=dp), dimension(nstates,nstates), intent(in), optional my_rho )
```

Nuclear velocities are rescaled along the direction of the non-adiabatic couplings to impose energy conservation in case a hop to a new potential energy surface has occured.

Parameters

in,out	*v*	nuclear velocity
in	*trajlabel*	label of the trajectory
in	*old_occ_state*	previous active state
	deltaE	potential energy difference between the old and the new electronic states
	scal1	squared modulus of the non-adiabatic couplings divided by the nuclear mass
	scal2	scalar product betweem the nuclear velocity and the non-adiabatic couplings
	energy_check	criterion to identify the possibility of jump
	scaling_factor	factor to rescal the velocities along the non-adiabatic couplings
	i_dof	integer index running over the n_dof degrees of freedom

Returns

The value of the new nuclear velocity is returned in case a hop occurred.

2.10 time_evolution Module Reference

Complete time evolution of Ntraj trajectories with Ehrenfest, surface hopping and CT-MQC , along with time initialization and finalization.

Functions/Subroutines

- subroutine evolution

  Three algorithms are used to evolve classical nuclear trajectories along with the electronic coefficients: Ehrenfest dynamics, trajectory surface hopping and CT-MQC.
- subroutine input_summary

  Summary of the input is written on the terminal.
- subroutine initialize_local_vars

  Variables used in the evolution subroutine are inizialized.
- subroutine finalize_local_vars

  Variables used in the evolution subroutine are deallocated.

Variables

- real(kind=dp), dimension(:,:), allocatable **rcl**
- real(kind=dp), dimension(:,:), allocatable **vcl**
- real(kind=dp), dimension(:), allocatable **classical_force**
- real(kind=dp), dimension(:,:,:), allocatable **my_force**
- real(kind=dp), dimension(:,:,:), allocatable **k_li**
- real(kind=dp), dimension(:,:), allocatable **tdvp**
- real(kind=dp), dimension(:), allocatable **boenergy_old**
- real(kind=dp), dimension(:,:,:), allocatable **coup_old**
- real(kind=dp), dimension(:,:,:), allocatable **qmom**
- integer, dimension(:,:,:), allocatable **qmom_type**
- real(kind=dp), dimension(:,:,:), allocatable **my_force_e**
- complex(kind=qp), dimension(:,:,:), allocatable **bosigma**
- complex(kind=qp), dimension(:,:), allocatable **bocoeff**

2.10.1 Detailed Description

Complete time evolution of Ntraj trajectories with Ehrenfest, surface hopping and CT-MQC , along with time initialization and finalization.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.10.2 Function/Subroutine Documentation

2.10.2.1 evolution()

```
subroutine time_evolution::evolution
```

Three algorithms are used to evolve classical nuclear trajectories along with the electronic coefficients: Ehrenfest dynamics, trajectory surface hopping and CT-MQC.

Parameters

time	time step
itraj	integer index running over the Ntraj trajectories
i,j	integer indices

2.10.2.2 finalize_local_vars()

```
subroutine time_evolution::finalize_local_vars
```

Variables used in the evolution subroutine are deallocated.

Parameters

check	control factor for deallocation errors

2.10.2.3 initialize_local_vars()

```
subroutine time_evolution::initialize_local_vars
```

Variables used in the evolution subroutine are inizialized.

Parameters

itraj	integer index running over the Ntraj trajectories
i,j	integer indices

2.11 tools Module Reference

Numerical tools for initialization and finalization of the dynamically-allocated vectors, along with initialization of random numbers generators.

Functions/Subroutines

- subroutine initialize_dynamics_vars
  Initialization of dynamics variables.
- subroutine initialize_trajectory_vars
  Initialization of trajectory variables.
- subroutine finalize
  Deallocation of dynamically-allocayed arrays.
- subroutine generate_random_seed

Inizialization of random number generator for the initial conditions.

- subroutine generate_random_seed_hop

  Inizialization of random number generator for the trajectory hops.

2.11.1 Detailed Description

Numerical tools for initialization and finalization of the dynamically-allocated vectors, along with initialization of random numbers generators.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.11.2 Function/Subroutine Documentation

2.11.2.1 finalize()

```
subroutine tools::finalize
```

Deallocation of dynamically-allocayed arrays.

Parameters

check	control factor for deallocation errors

2.11.2.2 generate_random_seed()

```
subroutine tools::generate_random_seed
```

Inizialization of random number generator for the initial conditions.

Parameters

seed	seed for the random number generator
n,i	integer indices

2.11.2.3 generate_random_seed_hop()

```
subroutine tools::generate_random_seed_hop
```

Inizialization of random number generator for the trajectory hops.

Parameters

seed	seed for the random number generator
n,i	integer indices

2.11.2.4 initialize_dynamics_vars()

```
subroutine tools::initialize_dynamics_vars
```

Initialization of dynamics variables.

Parameters

check	control factor for allocation errors

2.11.2.5 initialize_trajectory_vars()

```
subroutine tools::initialize_trajectory_vars
```

Initialization of trajectory variables.

Parameters

check	control factor for allocation errors

2.12 trajectories_selection Module Reference

Subroutine under construction to "manually" select the coupled trajectories in CT-MQC.

Functions/Subroutines

- subroutine **select_coupled_trajectories**

2.12.1 Detailed Description

Subroutine under construction to "manually" select the coupled trajectories in CT-MQC.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.13 variables Module Reference

The module defines all common variables.

Variables

- real(kind=dp), parameter hbar = 1.0_dp

    Reduced Planck constant.
- real(kind=dp), parameter zero = 0.0000000010_dp

    "Numerical" zero
- real(kind=dp), parameter au_to_ang = 0.52917721067121_dp

    Conversion factor from bohr to angstrom.
- real(kind=dp), parameter au_to_ev = 27.2114_dp

    Conversion factor from Hartree to electronvolts.
- real(kind=dp), parameter amu_to_au = 1836.0_dp

    Conversion factor from atomic mass units to atomic units.
- complex(kind=qp) im_unit = (0.0_dp, 1.0_dp)

    Imaginary unit.
- real(kind=dp) pi = 3.14159265359_dp

    pi
- complex(kind=qp) cmp = CMPLX(0.0_dp, 1.0_dp, qp)

    pi
- character(len=5) typ_cal = "EHREN"

    Type of dynamics that is executed: EHREN for Ehrenfest dynamics, TSHLZ for surface hopping with Landau-Zener hopping probability, TSHFS for surface hopping with fewest-switches hopping probability, CTMQC for CT-MQC, read in input.
- character(len=100) model_potential = "unknown"

    Name of the model potential as it is defined in QuantumModelLib, read in input.
- integer option = 1

    Only used for Tully models and can be 1, 2, or 3, read in input.
- logical new_potential = .FALSE.

    It is FALSE if the QuantumModelLib potential library is used; it is TRUE if the potentials in analytical_potentials.f90 are used, read in input.
- integer n_dof = 1

    Number of degrees of nuclear freedom, read in input.
- integer nstates = 2

    Number of electronic states, read in input.
- integer npairs = 1

    Number of pairs of electronic states.
- logical, dimension(100) periodic_variable = .FALSE.

    It is TRUE for each periodic nuclear coordinate, read in input.
- real(kind=dp), dimension(100) periodicity = 0.0_dp

    Periodicity of the correspoding nuclear nuclear coordinate in unit of pi, read in input.
- character(len=2) type_deco = ""

    Type of decoherence scheme applied on surface hooping: CT based on coupled trajectories and on quantum momentum, ED which is the energy-decoherence correction, read in input.
- real(kind=dp) c_parameter = 0.1_dp

    Value of the parameter C in the energy-decoherence correction used in surface hopping, read in input.
- integer jump_seed = -100

    Seed for the random number generator used for the probability jump is surface hopping, read in input.

- integer initial_condition_seed = -100

  Seed for the random number generator used for the selection of initial conditions, read in input.
- real(kind=dp) adia_nrg_gap = 10000.0D0

  Energy treshold to compute the non-adiabatic coupling vectors for the classical force or the Landau-Zener probability.
- real(kind=dp) lz_dist_cutoff = 0.20D0

  Distance cutoff from the crossing region to compute the Landau-Zener probability.
- logical nrg_check = .FALSE.

  It is TRUE if the spin-orbit coupling is switched-off when the energy gap between spin-diabatic states is above a certain treshold, read in input.
- real(kind=dp) nrg_gap = 10000.0D0

  Energy treshold to switch-off the spin-orbit coupling, read in input.
- logical spin_dia = .FALSE.

  It is TRUE when the spin-diabatic basis is used in CT-MQC, read in input.
- logical qmom_force = .TRUE.

  It is TRUE when quantum-momentum force is used in CT-MQC, read in input.
- logical f_correction = .FALSE.

  It is TRUE when energy conserving acc_force is used in CT-MQC (CTMQC-E), read in input.
- real(kind=dp) r_threshold = 0.001_dp

  R_threshold only used when f_correction= TRUE to determine cut-off for computation of modified acc force in CTMQC-E.
- real(kind=dp), dimension(100) m_parameter = 100.0_dp

  M_parameter is used only when cl_qmom = TRUE to determine "how far" each trajectory has to search to find its neighbours, read in input.
- integer rescaling_type = 2

  type of momentum rescaling after hop: 0 = isotropical, 2 = along NACV, 1 = along NACV, if frustrated isotropical
- logical reflect_frust = .FALSE.

  It is TRUE if you want to invert the full velocity vector after frustrated hop.
- logical force_hops = .FALSE.

  Adds a threshold to force hops in CTTSH.
- real(kind=dp) hop_thr = 0.6

  Above this threshold of population a trajectory is forced to hop.
- logical energy_sharing = .FALSE.

  Option to share energy for CCT TSH.
- integer sharing_type = 0

  Types of energy sharing: 0 = equity based, 1 = overlap based, 2 = Qmom based.
- logical doubleintercept = .true.

  If true then Qmom is calculated using double intercept.
- real(kind=dp) dt = 0.1_dp

  Time step, read in input.
- real(kind=dp) final_time = 0.0_dp

  Length of the simulations, read in input.
- real(kind=dp), dimension(100) r_init

  Mean positions to initialize nuclear positions, read in input.
- real(kind=dp), dimension(100) k_init

  Mean momenta to initialize nuclear momenta, read in input.
- real(kind=dp), dimension(100) mass_input = 0.0_dp

  Nuclear masses, read in input.
- real(kind=dp), dimension(100) sigmar_init

  Position variances to initialize nuclear positions, read in input.
- real(kind=dp), dimension(100) sigmap_init = -100.0_dp

  Momentum variances to initialize nuclear momenta, only necessary for non-Wigner sampling, read in input.

- integer ntraj = 100

  Number of nuclear trajectories, read in input.
- integer nsteps = 100

  Total number of dynamics time steps.
- integer nesteps = 20

  Number of electronic time-steps per nuclear time-step.
- integer dump = 1

  Number of time steps after which the output is dumped, read in input.
- integer n_init_bo = 1

  Number of initially populated electronic state(s), read in input.
- integer, dimension(100) init_bostate = -1

  Initial electronic state(s), read in input.
- real(kind=dp), dimension(100) weight_initbo = 1.0_dp

  Weight(s) of the initially populated electronic state(s), read in input.
- real(kind=dp), dimension(100) phase_initbo = 0.0_dp

  Phase(s) of the initially populated electronic state(s), read in input.
- real(kind=dp), dimension(:), allocatable r0

  Mean nuclear positions.
- real(kind=dp), dimension(:), allocatable r02

  Mean nuclear positions squared.
- real(kind=dp), dimension(:), allocatable k0

  Mean nuclear momenta.
- integer, dimension(:), allocatable initial_bostate

  BO states with non-zero initial occupation.
- real(kind=dp), dimension(:), allocatable weight_bostate

  Occupation of the BO states with non-zero initial occupation.
- real(kind=dp), dimension(:), allocatable phase_bostate

  Phases of the BO coefficients.
- real(kind=dp), dimension(:), allocatable period

  Periodicity of the periodic nuclear nuclear coordinate in unit of pi.
- logical, dimension(:), allocatable periodic_in

  It is TRUE for each periodic nuclear coordinate.
- real(kind=dp), dimension(:), allocatable mass

  Nuclear masses.
- real(kind=dp), dimension(:), allocatable sigma

  Position variances to initialize nuclear positions.
- real(kind=dp), dimension(:), allocatable var_momentum

  Momentum variances to initialize nuclear momenta.
- real(kind=dp), dimension(:,:,:), allocatable boforce

  Gradients of the electronic energies, either adiabatic or spin-(a)diabatic.
- real(kind=dp), dimension(:,:,:,:), allocatable coup

  Non-adiabatic couplings.
- complex(kind=qp), dimension(:,:,:), allocatable coup_so

  Spin-orbit coupling.
- real(kind=dp), dimension(:,:), allocatable boenergy

  Electronic energies, either adiabatic or spin-(a)diabatic.
- real(kind=dp), dimension(:), allocatable bo_pop

  Populations of the electronic states computed from the electronic coefficients.
- real(kind=dp), dimension(:), allocatable bo_pop_sh

  Populations of the electronic states computed in surface hopping as the ratio of trajectories running in each state over the total number of trajectories.

- real(kind=dp), dimension(:), allocatable dia_pop

  diabatic Populations of the electronic states computed from the electronic coefficients
- real(kind=dp), dimension(:), allocatable bo_coh

  Electronic coherences.
- real(kind=dp) ctmqc_e

  Trajectory-averaged CTMQC Energy.
- real(kind=dp), dimension(:,:), allocatable initial_positions

  Initial nuclear positions.
- real(kind=dp), dimension(:,:), allocatable initial_momenta

  Initial nuclear momenta.
- real(kind=dp), dimension(:), allocatable weight

  Weight of each trajectory (usually it is equal to unity)
- real(kind=dp), dimension(:), allocatable tdpes

  Gauge invariant part of the TDPES in CT-MQC calculation and mean Ehrenfest potential in Ehrenfest dynamics.
- real(kind=dp), dimension(:), allocatable density

  Nuclear density.
- integer, dimension(:), allocatable occ_state

  Active or force state in surface hopping.
- integer, dimension(:), allocatable lz_hop

  Keeps track of jumps in Landau-Zener surface hopping.
- integer count_traj

  Counts the trajectories that go through the avoided crossing.
- integer, dimension(:), allocatable list_coupled_trajectories

  List of coupled trajectories in CT-MQC (in the current version all trajectories are coupled)
- real(kind=dp), dimension(:,:,:), allocatable vec0

  NACV used to assure the Phase following from QmodelLib.
- real(kind=dp), dimension(:,:,:), allocatable previous_eigenv

  If NEW_POTENTIAL == .TRUE. this variable is needed to check the phase of neighboring eigenvectors.
- real(kind=dp), dimension(:,:,:,:), allocatable previous_coup

  Checks that the NACVs are continous along a trajectory.
- logical **initial_coup** = .TRUE.
- character(len=400) positions_file = ""

  Path to the file here initial positions are listed in case they are generated by another program, read in input.
- character(len=400) momenta_file = ""

  Path to the file here initial momenta are listed in case they are generated by another program, read in input.
- character(len=400) output_folder = "./"

  Path to the directory where the output is written, read in input; note that in such directory, two sub-directories (coeff and trajectories) have to be created.

2.13.1 Detailed Description

The module defines all common variables.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.14 wigner_distribution Module Reference

Sampling of the initial conditions based on the harmonic Wigner distribution using the Box-Muller algorithm.

Functions/Subroutines

- subroutine initial_conditions

  If initial conditions are not provided, they are sampled according to Gaussian distributions.
- real(kind=dp) function, dimension(my_nrand) gaussian_distribution (xi, nrand, var, x0, my_nrand)

  Box-Muller transform to generate normally distributed random number starting with uniformly distributed random numbers between 0 and 1.

2.14.1 Detailed Description

Sampling of the initial conditions based on the harmonic Wigner distribution using the Box-Muller algorithm.

Author

Federica Agostini, Institut de Chimie Physique, University Paris-Saclay.

2.14.2 Function/Subroutine Documentation

2.14.2.1 gaussian_distribution()

```
real(kind=dp) function, dimension(my_nrand) wigner_distribution::gaussian_distribution (
             real(kind=dp), dimension(nrand), intent(in) xi,
             integer, intent(in) nrand,
             real(kind=dp), intent(in) var,
             real(kind=dp), intent(in) x0,
             integer, intent(in) my_nrand )
```

Box-Muller transform to generate normally distributed random number starting with uniformly distributed random numbers between 0 and 1.

Parameters

in	*nrand*	amount of normally distributed random numbers to be generated
in	*my_nrand*	amount of normally distributed random numbers that are needed
in	*xi*	array of uniformly distributed random numbers
in	*var*	variance of the Gaussian distribution
in	*x0*	mean value of the Gaussian distribution
	y_tmp	normally distributed random numbers
	y	normally distributed random numbers that are returned by the function
	i,j	integer indices

Returns

Normally distributed random numbers are generated.

2.14.2.2 initial_conditions()

```
subroutine wigner_distribution::initial_conditions
```

If initial conditions are not provided, they are sampled according to Gaussian distributions.

Parameters

xi	array of random numbers uniformally distributed
check	control factor allocation errors
nrand	integer index
i	integer index
ios	control factor output errors

Returns

Initial positions and initial momenta are generated.

Index