Libra Summer School and Workshop 2024

Alexey Akimov

University at Buffalo, SUNY

July 9, 2024

Defining Hamiltonians in Libra

Wavefunction and selection of representation

University at Buffalo
The State University of New York

Akimov, A. V. Fundamentals of Trajectory-Based Methods for Nonadiabatic Dynamics. In *Comprehensive Computational Chemistry*; Elsevier, 2024; pp 235–272. https://doi.org/10.1016/B978-0-12-821978-2.00034-9.

$|\Psi\rangle$ Abstract wavefunction

$\{|\boldsymbol{r}\rangle : \hat{\boldsymbol{r}}|\boldsymbol{r}\rangle = \boldsymbol{r}|\boldsymbol{r}\rangle\}$ Position states (Hilbert space)

$\{|\boldsymbol{k}\rangle : \hat{\boldsymbol{k}}|\boldsymbol{k}\rangle = \boldsymbol{k}|\boldsymbol{k}\rangle\}$ Momentum states (Hilbert space)

$\Psi(\boldsymbol{r}) = \langle \boldsymbol{r}|\Psi\rangle$ Wavefunction in a **position representation** – representation in the basis of position states

$$1 = \int d\boldsymbol{r}'|\boldsymbol{r}'\rangle\langle \boldsymbol{r}'| \quad \text{Complete basis}$$

Indeed

$$|\Psi\rangle = \int d\boldsymbol{r}'|\boldsymbol{r}'\rangle\,\langle \boldsymbol{r}'|\Psi\rangle = \int d\boldsymbol{r}'\,|\boldsymbol{r}'\rangle\Psi(\boldsymbol{r}')$$

$\Psi(\boldsymbol{r}')$ is essentially an expansion coefficient in the basis of coordinate states $\{|\boldsymbol{r}\rangle\}$
Can be regarded as DVR (grid representation of the wavefunction)

$\Psi(\boldsymbol{k}) = \langle \boldsymbol{k}|\Psi\rangle$ Likewise, the **momentum representation** of a wavefunction

Different Hilbert spaces:

$\Psi(\boldsymbol{r}) = \langle \boldsymbol{r}|\Psi\rangle$; $|\boldsymbol{r}\rangle, |\Psi\rangle \in \mathcal{H}_{\boldsymbol{r}}$ only electrons

$\Psi(\boldsymbol{R}) = \langle \boldsymbol{R}|\Psi\rangle$; $|\boldsymbol{R}\rangle, |\Psi\rangle \in \mathcal{H}_{\boldsymbol{R}}$ only nuclei

$\Psi(\boldsymbol{r}, \boldsymbol{R}) = \langle \boldsymbol{r}, \boldsymbol{R}|\Psi\rangle$; $|\Psi\rangle \in \mathcal{H}_r \otimes \mathcal{H}_{\boldsymbol{R}}$ both electrons and nuclei

$\Psi(\boldsymbol{r}) = \langle \boldsymbol{R}|\Psi\rangle$; $|\Psi\rangle \in \mathcal{H}_r \otimes \mathcal{H}_{\boldsymbol{R}}$ projection on $|\boldsymbol{R}\rangle \in \mathcal{H}_{\boldsymbol{R}}$; $\Psi(\boldsymbol{r}; \boldsymbol{R}) \in \mathcal{H}_r$ operator of electronic DOF (electronic coordinate operator), but a function of nuclear DOF - $\boldsymbol{R}$

Working in Different Hilbert Spaces: Exact-Factorization Example

University at Buffalo
The State University of New York

Han, D.; Akimov, A.V. *J. Chem. Theory Comput.* **2024,** 20, 5022–5042

electron-nuclear state $|\Psi(t)\rangle \in \mathcal{H}_{\boldsymbol{r}\times\boldsymbol{R}} = \mathcal{H}_{\boldsymbol{r}} \otimes \mathcal{H}_{\boldsymbol{R}}$

Electronic Hilbert space Electronic Hilbert space

$|\Psi(t)\rangle = |\chi(t), \Phi(t)\rangle$

$|\boldsymbol{R}\rangle \in \mathcal{H}_{\boldsymbol{R}}$

$|\boldsymbol{r}\rangle \in \mathcal{H}_{\boldsymbol{r}}$

Using the resolution-of-identity: $|\Psi(t)\rangle = \int d\,\boldsymbol{R}|\boldsymbol{R}\rangle\langle\boldsymbol{R}|\Psi(t)\rangle = \int d\,\boldsymbol{R}|\boldsymbol{R}\rangle\langle\boldsymbol{R}|\chi(t), \Phi(t)\rangle = \int d\,\boldsymbol{R}|\boldsymbol{R}\rangle\langle\boldsymbol{R}|\chi(t)\rangle|\Phi_{\boldsymbol{R}}(t)\rangle$

marginal nuclear state $|\chi(t)\rangle \in \mathcal{H}_{\boldsymbol{R}}$

conditional electronic state $|\Phi_{\boldsymbol{R}}(t)\rangle \in \mathcal{H}_{\boldsymbol{r}}$

Nuclear wavefunction in position representation: $\chi(\boldsymbol{R}, t) = \langle\boldsymbol{R}|\chi(t)\rangle$

Electronic wavefunction for fixed nuclear geometry: $\Phi(\boldsymbol{r}, t; \boldsymbol{R}) = \langle\boldsymbol{r}|\Phi_{\boldsymbol{R}}(t)\rangle$

$$|\Psi(t)\rangle = \int d\boldsymbol{R}|\boldsymbol{R}\rangle\chi(\boldsymbol{R}, t)|\Phi_{\boldsymbol{R}}(t)\rangle \in \mathcal{H}_{\boldsymbol{r}\times\boldsymbol{R}}$$

molecular state $|\Psi(\boldsymbol{R}, t)\rangle$

$$|\Psi(\boldsymbol{R}', t)\rangle = \langle\boldsymbol{R}'|\Psi(t)\rangle = \int d\boldsymbol{R}\langle\boldsymbol{R}'|\boldsymbol{R}\rangle\chi(\boldsymbol{R}, t)|\Phi_{\boldsymbol{R}}(t)\rangle = \int d\boldsymbol{R}\delta(\boldsymbol{R}' - \boldsymbol{R})\chi(\boldsymbol{R}, t)|\Phi_{\boldsymbol{R}}(t)\rangle$$
$$= \chi(\boldsymbol{R}', t)|\Phi_{\boldsymbol{R}'}(t)\rangle \in \mathcal{H}_{\boldsymbol{r}}$$

Shorthand notation. Adiabatic and Diabatic Representations

University at Buffalo
The State University of New York

$\psi_i(\boldsymbol{r}) = \langle \boldsymbol{r}|i\rangle = \langle \boldsymbol{r}|\psi_i\rangle$ electronic coordinates, i-th basis state

Shorthand notation for the entire basis: $|\boldsymbol{\psi}\rangle = (|\psi_1\rangle, |\psi_2\rangle, \dots, |\psi_N\rangle)$

Matrix elements (scalars) → $A_{ij} = \langle \psi_i|\hat{A}|\psi_j\rangle \leftrightarrow A = \langle \boldsymbol{\psi}|\hat{A}|\boldsymbol{\psi}\rangle$ ← Matrix

Basis:

Adiabatic (Hamiltonian is diagonal):
$\langle \psi_{adi,i}|\hat{H}_{el}|\psi_{adi,j}\rangle = 0, \forall i \neq j$

Diabatic (NACs are exactly zero):
$\langle \psi_{dia,i}|\nabla_{\boldsymbol{R}}|\psi_{dia,j}\rangle = 0, \forall \boldsymbol{R}$

$$|\boldsymbol{\psi}_{adi}\rangle = |\boldsymbol{\psi}_{dia}\rangle U$$

$H_{adi} = \langle \boldsymbol{\psi}_{adi}|\hat{H}_{el}|\boldsymbol{\psi}_{adi}\rangle$ Hamiltonians $H_{dia} = \langle \boldsymbol{\psi}_{dia}|\hat{H}_{el}|\boldsymbol{\psi}_{dia}\rangle$

Transformation between bases $H_{dia}U = SUH_{adi}$ $H_{adi} = U^{+}H_{dia}U = \widetilde{H}_{dia}$

Wavefunction should be invariant w.r.t. the basis representation $|\Psi(t)\rangle = |\boldsymbol{\psi}_{adi}(t)\rangle C_{adi}(t) = |\boldsymbol{\psi}_{dia}(t)\rangle C_{dia}(t)$

TD-SE in the Shorthand notation

University at Buffalo
The State University of New York

Wavefunction should be invariant w.r.t. the basis representation

$$|\Psi(t)\rangle = |\boldsymbol{\psi}_{adi}(t)\rangle C_{adi}(t) = |\boldsymbol{\psi}_{dia}(t)\rangle C_{dia}(t)$$

$$i\hbar \frac{\partial}{\partial t}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle$$

$$|\boldsymbol{\psi}\rangle = \left(|\psi_1\rangle, |\psi_2\rangle, \dots, |\psi_N\rangle\right)$$

$$C = (c_1, c_2, \dots, c_N)^T$$

$$i\hbar \frac{\partial}{\partial t}|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle C_{rep} + i\hbar|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle \frac{\partial}{\partial t} C_{rep} = \hat{H}|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle C_{rep}$$

Projecting:

$$i\hbar\langle\boldsymbol{\psi}_{\boldsymbol{rep}}(t)|\frac{\partial}{\partial t}|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle C_{rep} + i\hbar\langle\boldsymbol{\psi}_{\boldsymbol{rep}}(t)|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle \frac{\partial}{\partial t} C_{rep} = \langle\boldsymbol{\psi}_{\boldsymbol{rep}}(t)|\hat{H}|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle C_{rep}$$

$$\boxed{i\hbar S_{rep}\frac{\partial}{\partial t} C_{rep} = \left[\langle\boldsymbol{\psi}_{\boldsymbol{rep}}(t)|\hat{H}|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle - i\hbar\langle\boldsymbol{\psi}_{\boldsymbol{rep}}(t)|\frac{\partial}{\partial t}|\boldsymbol{\psi}_{\boldsymbol{rep}}(t)\rangle\right] C_{rep}}$$

Implementation in Libra classes

University at Buffalo
The State University of New York

Wavefunction language $|\Psi(t)\rangle = |\boldsymbol{\psi}_{adi}(t)\rangle \boldsymbol{C}_{adi}(t) = |\boldsymbol{\psi}_{dia}(t)\rangle \boldsymbol{C}_{dia}(t)$

Density matrix language $\hat{\rho} = |\Psi\rangle\langle\Psi|$

$$P_{adi} = \langle\boldsymbol{\psi}_{adi}|\hat{\rho}|\boldsymbol{\psi}_{adi}\rangle = \langle\boldsymbol{\psi}_{adi}|\boldsymbol{\psi}_{adi}\rangle \boldsymbol{C}_{adi}\boldsymbol{C}_{adi}^{+}\langle\boldsymbol{\psi}_{adi}|\boldsymbol{\psi}_{adi}\rangle = \boldsymbol{I}\boldsymbol{C}_{adi}\boldsymbol{C}_{adi}^{+}\boldsymbol{I} = \boldsymbol{C}_{adi}\boldsymbol{C}_{adi}^{+}$$

$$P_{dia} = \langle\boldsymbol{\psi}_{dia}|\hat{\rho}|\boldsymbol{\psi}_{dia}\rangle = \langle\boldsymbol{\psi}_{dia}|\boldsymbol{\psi}_{dia}\rangle \boldsymbol{C}_{dia}\boldsymbol{C}_{dia}^{+}\langle\boldsymbol{\psi}_{dia}|\boldsymbol{\psi}_{dia}\rangle = \boldsymbol{S}\boldsymbol{C}_{dia}\boldsymbol{C}_{dia}^{+}\boldsymbol{S}$$

dyn_variables
- ndia, nadi, ndof, ntraj
- ampl_dia, ampl_adi
- dm_adi, dm_dia

So the conversions of the density matrices is:

$$\boldsymbol{U}\boldsymbol{P}_{dia}\boldsymbol{U}^{+} = \boldsymbol{U}^{+}\boldsymbol{S}\boldsymbol{C}_{dia}\boldsymbol{C}_{dia}^{+}\boldsymbol{S}\boldsymbol{U} = \boldsymbol{C}_{adi}\boldsymbol{C}_{adi}^{+} = \boldsymbol{P}_{adi} \rightarrow \boldsymbol{P}_{dia} = \boldsymbol{U}^{-1}\boldsymbol{P}_{adi}(\boldsymbol{U}^{-1})^{+} = \boldsymbol{U}^{+}\boldsymbol{S}\boldsymbol{P}_{adi}(\boldsymbol{U}^{+}\boldsymbol{S})^{+} = \boldsymbol{U}^{+}\boldsymbol{S}\boldsymbol{P}_{adi}\boldsymbol{S}\boldsymbol{U}$$

Coefficient transformation:

$$\boldsymbol{C}_{dia} = \boldsymbol{U}\boldsymbol{C}_{adi} \leftrightarrow \boldsymbol{C}_{adi} = \boldsymbol{U}^{-1}\boldsymbol{C}_{dia} \leftrightarrow \boldsymbol{C}_{adi} = \boldsymbol{U}^{+}\boldsymbol{S}\boldsymbol{C}_{dia}$$

$\boldsymbol{H}_{dia}\boldsymbol{U} = \boldsymbol{S}\boldsymbol{U}\boldsymbol{H}_{adi}$ and also computes derivative couplings and adiabatic gradients

computes $\boldsymbol{H}_{dia}$, $\boldsymbol{D}_{dia}$, and $\nabla\boldsymbol{H}_{dia}$ according to given methods (e.g. Python modules with Hamiltonian models)

nHamiltonian
- ampl_dia2adi
- ampl_adi2dia
- compute_adiabatic()
- compute_diabatic()

Implementation in the nHamiltonian class

University at Buffalo
The State University of New York

$$|\boldsymbol{\psi}\rangle = (|\psi_1\rangle, |\psi_2\rangle, \dots, |\psi_N\rangle)$$

$$\boldsymbol{H}_{dia} = \langle \boldsymbol{\psi}_{dia} | \hat{H} | \boldsymbol{\psi}_{dia} \rangle$$

Hamiltonian matrix elements

$$\boldsymbol{H}_{adi} = \langle \boldsymbol{\psi}_{adi} | \hat{H} | \boldsymbol{\psi}_{adi} \rangle$$

Unitary (similarity) transformation

$$\boldsymbol{H}_{adi} = \boldsymbol{U}^{+} \boldsymbol{H}_{dia} \boldsymbol{U} = \widetilde{\boldsymbol{H}}_{dia}$$

$$|\boldsymbol{\psi}_{adi}\rangle = |\boldsymbol{\psi}_{dia}\rangle \boldsymbol{U}$$

First-order derivative coupling vectors

$$\boldsymbol{D}_{adi} \equiv \langle \boldsymbol{\psi}_{adi} | \boldsymbol{\nabla} \boldsymbol{\psi}_{adi} \rangle$$

$$\boldsymbol{D}_{dia} \equiv \langle \boldsymbol{\psi}_{dia} | \boldsymbol{\nabla} \boldsymbol{\psi}_{dia} \rangle$$

nHamiltonian

- CMATRIX* ham_dia, nac_dia, hvib_dia
- CMATRIX* ham_adi, nac_adi, hvib_adi
- CMATRIX* ovlp_dia, time_overlap_dia
- CMATRIX* ovlp_adi, time_overlap_adi
- CMATRIX* basis_transform
- vector<CMATRIX*> dc1_adi, dc1_dia
- vector<CMATRIX*> d1ham_adi, d1ham_dia
- compute_adiabatic()

$$\boldsymbol{D}_{dia} = \left\langle \boldsymbol{\psi}_{dia} \middle| \frac{\partial}{\partial t} \middle| \boldsymbol{\psi}_{dia} \right\rangle$$

Scalar NACs

$$\boldsymbol{D}_{adi} = \left\langle \boldsymbol{\psi}_{adi} \middle| \frac{\partial}{\partial t} \middle| \boldsymbol{\psi}_{adi} \right\rangle$$

"Vibronic" Hamiltonian

$$H_{vib,dia} = H_{dia} - i\hbar D_{dia}$$

$$H_{vib,adi} = H_{adi} - i\hbar D_{adi}$$

Time-overlaps (transition density matrices)

$$\langle \boldsymbol{\psi}_{dia}(t - \Delta t) | \boldsymbol{\psi}_{dia}(t) \rangle = \boldsymbol{St}_{dia}(t - \Delta t, t) \approx I$$

$$\langle \boldsymbol{\psi}_{adi}(t - \Delta t) | \boldsymbol{\psi}_{adi}(t) \rangle = \boldsymbol{St}_{adi}(t - \Delta t, t)$$

$$\langle \boldsymbol{\psi}_{dia} | \boldsymbol{\psi}_{dia} \rangle = \boldsymbol{S}$$

$$\langle \boldsymbol{\psi}_{adi} | \boldsymbol{\psi}_{adi} \rangle = \boldsymbol{I}$$

The diabatic basis is not necessarily orthonormal

How to compute NACs?

$$\begin{cases} \boldsymbol{U}^{+} \nabla \boldsymbol{H}_{dia} \boldsymbol{U} - \left(\widetilde{\boldsymbol{D}}_{dia}^{+} \boldsymbol{H}_{adi} + \boldsymbol{H}_{adi} \widetilde{\boldsymbol{D}}_{dia} \right) = \nabla \boldsymbol{H}_{adi} - \left(\boldsymbol{D}_{adi}^{+} \boldsymbol{H}_{adi} + \boldsymbol{H}_{adi} \boldsymbol{D}_{adi} \right) \\ \widetilde{\nabla \boldsymbol{H}_{dia}} - \left(\widetilde{\boldsymbol{D}}_{dia}^{+} \widetilde{\boldsymbol{H}}_{dia} + \widetilde{\boldsymbol{H}}_{dia} \widetilde{\boldsymbol{D}}_{dia} \right) = \nabla \boldsymbol{H}_{adi} - \left(\boldsymbol{D}_{adi}^{+} \boldsymbol{H}_{adi} + \boldsymbol{H}_{adi} \boldsymbol{D}_{adi} \right) \end{cases}$$

Then use special structure of the matrix

Nonadiabatic Couplings

University at Buffalo
The State University of New York

Properties of the NACs

$$\bar{\boldsymbol{D}}_{dia}^{+} + \bar{\boldsymbol{D}}_{dia} = \boldsymbol{\nabla S}$$

$$\bar{\boldsymbol{D}}_{adi} + \bar{\boldsymbol{D}}_{adi}^{+} = \boldsymbol{\nabla} S_{adi} = 0 \rightarrow (D_{adi}^{\alpha})^{+} = -D_{adi}$$

This is a well-known property!

$$D_{adi}^{\alpha} = \widetilde{D}_{dia}^{\alpha} + U^{+} S \nabla_{\alpha} U$$

$D_{rep,ij}^{\alpha} \equiv \langle \psi_{rep,i} | \nabla_{\alpha} \psi_{rep,j} \rangle$ is a scalar

$\boldsymbol{D}_{rep,ij} \equiv \langle \psi_{rep,i} | \nabla \psi_{rep,j} \rangle$ understood as a column-vector

$\bar{\boldsymbol{D}}_{rep} \equiv \langle \boldsymbol{\psi}_{rep} | \boldsymbol{\nabla} \boldsymbol{\psi}_{rep} \rangle$ understood as a vector of matrices $D_{rep}^{\alpha} = \langle \boldsymbol{\psi}_{rep} | \nabla_{\alpha} \boldsymbol{\psi}_{rep} \rangle$

Important observations

the equation becomes an identity when $U = I$

$$\underbrace{\widetilde{\nabla_{\alpha} H_{dia}} - \left((D_{adi}^{\alpha})^{+} \widetilde{H}_{dia} + \widetilde{H}_{dia} \widetilde{D}_{dia}^{\alpha} \right)}_{U^{+} \langle \boldsymbol{\psi}_{dia} | \nabla_{\alpha} H | \boldsymbol{\psi}_{dia} \rangle U} = \underbrace{\nabla_{\alpha} H_{adi} - \left((D_{adi}^{\alpha})^{+} H_{adi} + H_{adi} D_{adi}^{\alpha} \right)}_{\langle \boldsymbol{\psi}_{adi} | \nabla_{\alpha} H | \boldsymbol{\psi}_{adi} \rangle}$$

Quantum-Classical Hamiltonian and Ehrenfest forces

University at Buffalo
The State University of New York

$$H^{MF}(\boldsymbol{R},\boldsymbol{P};\Psi) = \frac{\langle\Psi|H^{qc}(\boldsymbol{R},\boldsymbol{P})|\Psi\rangle}{\langle\Psi|\Psi\rangle} = \frac{1}{2}\boldsymbol{P}^T M^{-1}\boldsymbol{P} + \frac{C_{adi}^{+}H_{adi}C_{adi}}{C_{adi}^{+}C_{adi}} = \frac{1}{2}\boldsymbol{P}^T M^{-1}\boldsymbol{P} + \frac{C_{dia}^{+}H_{dia}C_{dia}}{C_{dia}^{+}SC_{dia}}$$

nHamiltonian

- Ehrenfest_energy_adi
- Ehrenfest_energy_dia

TD-SE for the amplitudes

$$i\hbar S\frac{dC_{rep}}{dt} = \left(H_{rep} - i\hbar d_{rep}\right)C_{rep}$$

$$\frac{dH^{MF}}{dt} = 0$$

Energy conservation

$$\dot{\boldsymbol{R}} = M^{-1}\boldsymbol{P}$$

Enforcing the "classical" form of equations of motion for nuclear DOFs $\dot{\boldsymbol{P}} = f^{MF}(\boldsymbol{R},\Psi_{\mathrm{rep}})$

$$f_n^{MF} \equiv f_{n,adi}^{MF} = \frac{1}{C_{adi}^{+}C_{adi}}C_{adi}^{+}F_{adi,n}^{HF}C_{adi} = f_{n,dia}^{\mathrm{MF}} = \frac{1}{C_{dia}^{+}SC_{dia}}C_{dia}^{+}F_{dia,n}^{HF}C_{dia}$$

nHamiltonian

- Ehrenfest_forces_adi
- Ehrenfest_forces_dia
- Ehrenfest_forces_tens_adi
- Ehrenfest_forces_tens_dia

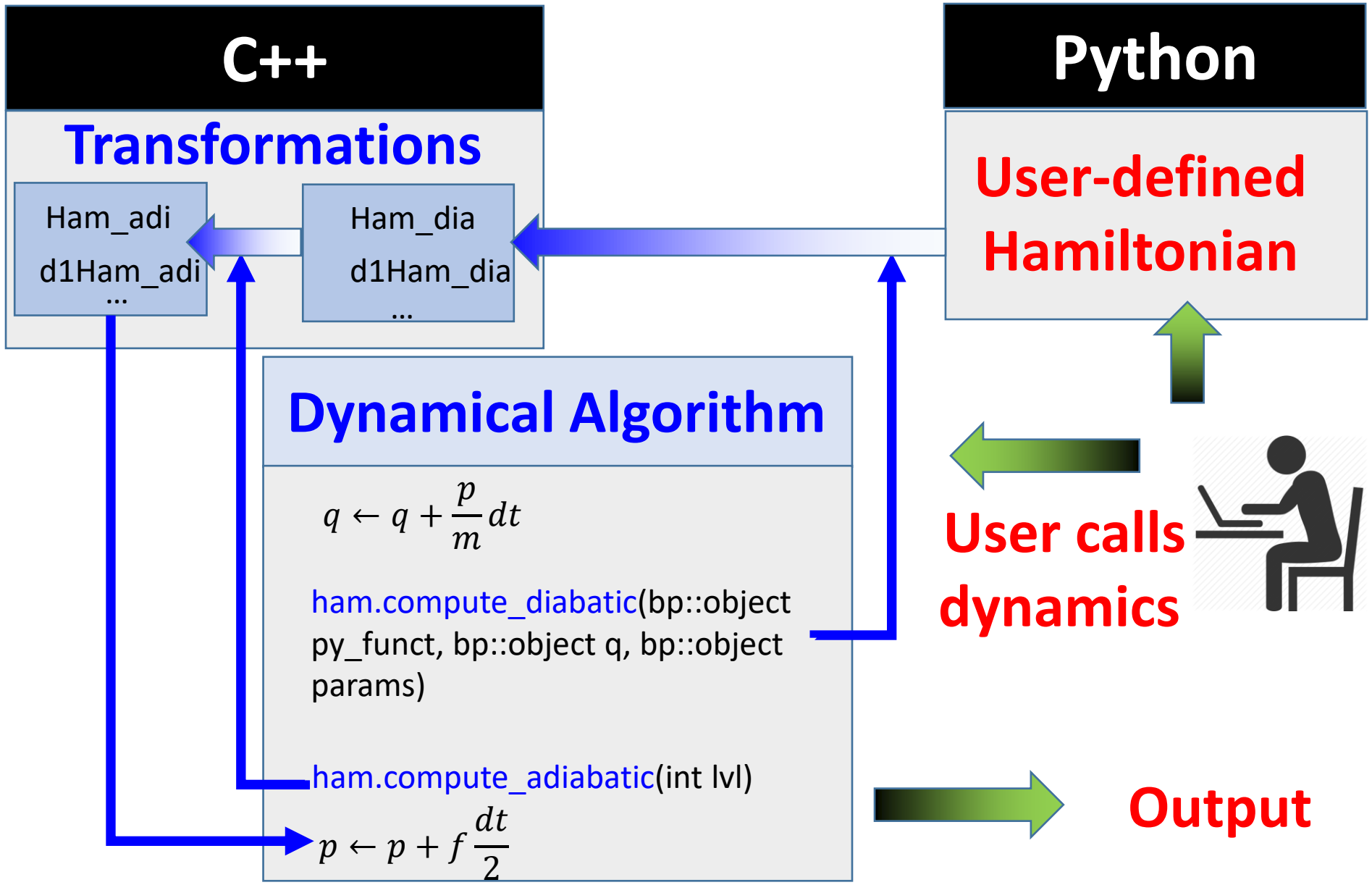
$$F_{adi,n}^{HF} = -\langle\boldsymbol{\psi}_{adi}|\nabla_n H|\boldsymbol{\psi}_{adi}\rangle = \left[-\nabla_n H_{adi} + D_{adi,n}^{+}H_{adi} + H_{adi}D_{adi,n}\right]$$

$$F_{dia,n}^{MF} = -\langle\boldsymbol{\psi}_{dia}|\nabla_{\mathrm{n}} H|\boldsymbol{\psi}_{dia}\rangle = \left[-\nabla_n H_{dia} + D_{dia,n}^{+}S^{-1}H_{dia} + H_{dia}S^{-1}D_{dia,n}\right]$$

More on the nHamiltonian class.

Making Interfaces

How `compute_dynamics` works



Different ways of computing matrix elements. Example of H_{dia}^{vib}

University at Buffalo
The State University of New York

Blue = Required Input
Green = Output
Green with D = Can be set up directly via Python function call

Function	Q	P	H_{dia}	D_{dia}	d_{dia}	H_{dia}^{vib}
nHamiltonian::**compute_diabatic**(bp::object py_funct …)			D	D	D	D
nHamiltonian::**compute_nac_dia**(…)						
nHamiltonian::**compute_hvib_dia**(…)						

Different ways of computing matrix elements. Example of H_{adi}^{vib}

University at Buffalo
The State University of New York

Function	Q	P	S	H_{dia}	∇H_{dia}	D_{dia}	U	H_{adi}	∇H_{adi}	D_{adi}	d_{adi}	H_{adi}^{vib}
nHamiltonian::**compute_diabatic**(bp::object py_funct …)			D	D	D	D						
nHamiltonian::**compute_adiabatic**(…)												
nHamiltonian::**compute_adiabatic**(bp::object py_funct …)							D	D	D	D	D	D
nHamiltonian::**compute_nac_adi**(…)												
nHamiltonian::**compute_hvib_adi**(…)												

nHamiltonian class as a hierarchical data type to handle multiple trajectories

University at Buffalo
The State University of New York

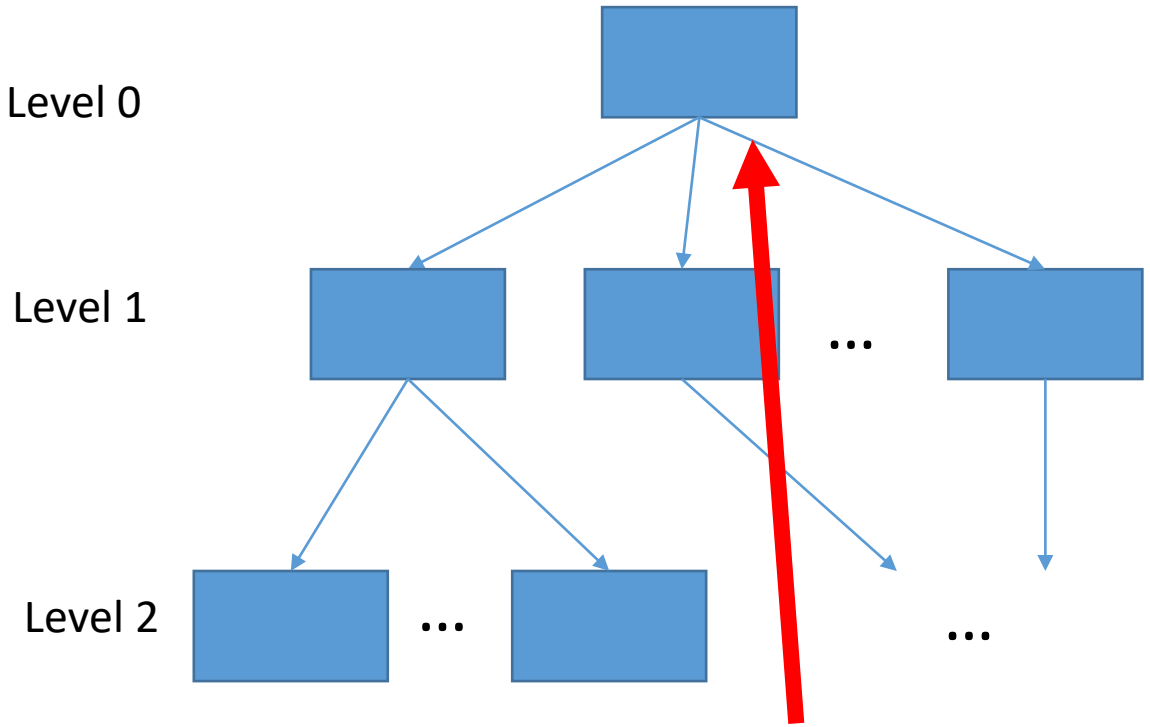
nHamiltonian

- level
- id
- nHamiltonian* parent
- vector<nHamiltonian*> children

- nnucl, nadi, ndia

- CMATRIX* ham_dia, nac_dia, hvib_dia
- CMATRIX* ham_adi, nac_adi, hvib_adi
- CMATRIX* ovlp_dia, time_overlap_dia
- CMATRIX* ovlp_adi, time_overlap_adi
- CMATRIX* basis_transform
- vector<CMATRIX*> dc1_adi, dc1_dia
- vector<CMATRIX*> d1ham_adi, d1ham_dia

- ampl_dia2adi
- ampl_adi2dia



int entanglement_opt

A selector of a method to couple the trajectories in this ensemble.
- 0: no coupling [default]
- 1: ETHD
- 2: ETHD3 (experimental)
- 22: another flavor of ETHD3 (experimental)

Packing variables for multiple trajectories

University at Buffalo
The State University of New York

Individual trajectory

Swarm of uncoupled trajectories

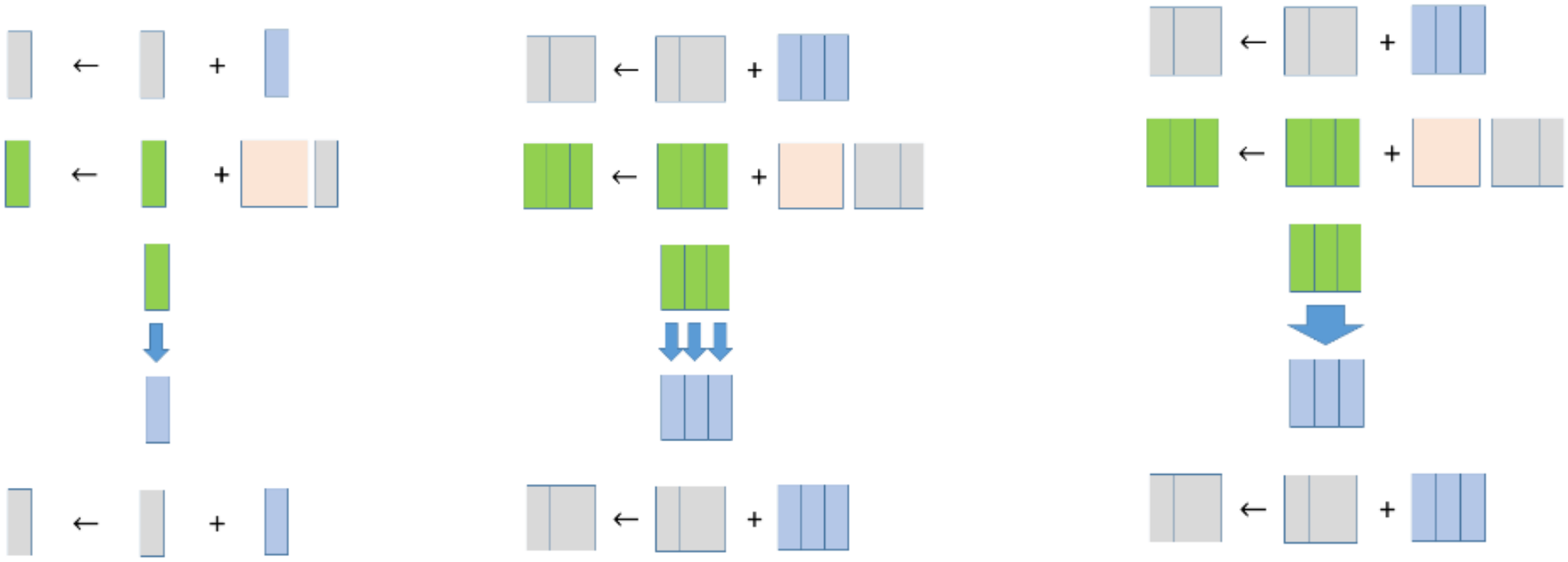
Swarm of coupled trajectories

$$p \leftarrow p + f\frac{dt}{2}$$

$$q \leftarrow q + M^{-1}pdt$$

Forces update

$$p \leftarrow p + f\frac{dt}{2}$$



Keep the Dynamical Workflow Fixed

University at Buffalo
The State University of New York

User defines how to run the dynamical simulation

```
for i in range(500):
    propagate_el(Cdia, Cadi, Hvib, Sdia, 0.5*dt, rep)
    p = p + 0.5*f*dt
    q = q + dt*p/m
    compute_model(model, Hdia, Sdia, d1ham_dia, dc1_dia, q, params)
    ham.compute_adiabatic(1);
    f = compute_frc(ham, Cdia, Cadi, rep)
    p = p + 0.5*f*dt
    Hvib = compute_Hvib(Hdia, Hadi, dc1_dia, dc1_adi, p, m, rep)
    propagate_el(Cdia, Cadi, Hvib, Sdia, 0.5*dt, rep)
Etot = compute_etot(ham, p, Cdia, Cadi, m, rep)
```

User defines what function to use to compute entries in the Hamiltonian object (diabatic/adiabatic Ham, overlap matrix, derivatives, etc.) - NEXT

Example: Model Calculations

University at Buffalo
The State University of New York

```
def model2(q, params):

    obj = tmp()
    obj.ham_dia = CMATRIX(2,2);   obj.ovlp_dia = CMATRIX(2,2);
    obj.d1ham_dia = CMATRIXList();  obj.d1ham_dia.append( CMATRIX(2,2))
    obj.dc1_dia = CMATRIXList();  obj.dc1_dia.append( CMATRIX(2,2))

    x = q.get(0)
    x0,k,D,V = params["x0"], params["k"], params["D"], params["V"]

    obj.ovlp_dia.set(0,0, 1.0+0.0j);  obj.ovlp_dia.set(0,1, 0.0+0.0j);
    obj.ovlp_dia.set(1,0, 0.0+0.0j);  obj.ovlp_dia.set(1,1, 1.0+0.0j);

    obj.ham_dia.set(0,0, k*x*x*(1.0+0.0j) );   obj.ham_dia.set(0,1, V*(1.0+0.0j));
    obj.ham_dia.set(1,0, V*(1.0+0.0j));          obj.ham_dia.set(1,1, (k*(x-x0)**2 + D)*(1.0

    for i in [0]:
        obj.d1ham_dia[i].set(0,0, 2.0*k*x*(1.0+0.0j) );   obj.d1ham_dia[i].set(0,1, 0.0+0.0j);
        obj.d1ham_dia[i].set(1,0, 0.0+0.0j);                obj.d1ham_dia[i].set(1,1,2.0*k*(x-x0)*(1.0+0.0j));

        obj.dc1_dia[i].set(0,0, 0.0+0.0j);   obj.dc1_dia[i].set(0,1,-0.1+0.0j);
        obj.dc1_dia[i].set(1,0, 0.1+0.0j);   obj.dc1_dia[i].set(1,1, 0.0+0.0j);

    return obj
```

Initialize Python objects

Set matrix elements according to your model

Example: Atomistic Calculations

University at Buffalo
The State University of New York

```
def model_atomistic(q, params, indx):

    natoms = params["natoms"];  ndof = q.num_of_rows;  ndia = params[ "ndia" ]
    params[ "output_filename" ] = "detailed.out“

    obj = tmp()
    obj.ham_dia = CMATRIX(1,1);
    obj.ovlp_dia = CMATRIX(1,1);          obj.ovlp_dia.set(0,0, 1.0+0.0j)
    obj.d1ham_dia = CMATRIXList();
    for i in xrange(ndof):
        obj.d1ham_dia.append( CMATRIX(1,1) )

    os.system("mkdir wd/job_"+str(indx))
    os.system("cp dftb_in.hsd wd/job_"+str(indx)) #+"/dftb_in.hsd")
    os.chdir("wd/job_"+str(indx))

    create_input.update_coordinates(q, params)
    os.system("srun %s < dftb_in.hsd > out" % (exe_name) )  # DFTB calculations are run here!
    dftb_forces = parse_output.get_forces(params)
    os.chdir("../../")

    for i in xrange(ndof):
        obj.d1ham_dia[i].set(0,0, dftb_forces[i]*(-1.0+0.0j) )
        obj.dc1_dia[i].set(0, 0, 0.0+0.0j)

return obj
```

Initialize Python objects

Prepare and Run external program

Set matrix elements according to your model