Documentation for `Hawaii Hybrid` v.0.1

A. Finenko and D. Chistikov

January 9, 2025

Contents

1 Hawaii Hybrid code organization

`enum MonomerType`

Values
`ATOM` = 0
`LINEAR_MOLECULE` = 4
`LINEAR_MOLECULE_REQUANTIZED_ROTATION` = `MODULO_BASE` + 4
`LINEAR_VIBRATING_MOLECULE` = `MODULO_BASE` + 6
`ROTOR` = 6
`ROTOR_REQUANTIZED_ROTATION` = 2*`MODULO_BASE` + 6

Description This enum is used to distinguish between systems of different types and store the size of the phase point: `size(phase_point) = MonomerType % MODULO_BASE`, where `MODULO_BASE` is `#defined` to 100 by default.

`enum PairState`

Values
`FREE_AND_METASTABLE` = 0
`BOUND` = 1

Description

`enum CalculationType`

Values
`PRMU` = 0
`CORRELATION_SINGLE` = 1
`CORRELATION_ARRAY` = 2

Description

`struct Monomer`

Fields
`MonomerType t`
`double I[3]` – values of tensor of inertia
`double *qp` – dynamic variables (currently, Euler angles and conjugated momenta) at the current step of simulation
`double *dVdq` – the derivatives of potential energy with respect to coordinates pertaining to this monomer (the order of coordinates is the same as for `qp`)
`bool apply_requantization`

Description The `apply_requantization` will be set to `true` in `rhs` to signal that the requantization of the monomer's angular momentum is required during trajectory propagation. The order of variables in the `qp` array is specified by the following indices:

```
#define IPHI 0
#define IPPHI 1
#define ITHETA 2
#define IPTHETA 3
#define IPSI 4
#define IPPSI 5
```

`struct MoleculeSystem`

Fields
`intermolecular_qp[6]` – Coordinates and conjugated momenta that correspond to the intermolecular motion: (Φ, p_Φ, Θ, p_Θ, R, p_R).
`Monomer m1`
`Monomer m2`
`double mu` – reduced mass of molecule pair
`size_t Q_SIZE` – total number of coordinates for molecule pair
`size_t QP_SIZE` – total number of coordinates and momenta for molecule pair (a.k.a. `size(phase_point)`)

`double *intermediate_q` – contiguous vector of coordinates
`double *dVdq` – contiguous vector of potential energy derivatives

Description: Keep in mind that angular variables and momenta are stored in the same order as for `qp` in `Monomer`. These variables' locations are `#defined` as follows:

```
#define IPHI 0
#define IPPHI 1
#define ITHETA 2
#define IPTHETA 3
#define IR 4
#define IPR 5
```

Keep in mind that intermolecular coordinates and monomer's coordinates are not stored contiguously. The contiguous vector of coordinates can be assembled by calling `extract_q_and_write_into_ms` function, which stores the coordinates in memory pointed at by `intermediate_q`. These coordinates are passed to external functions that compute the values of intermolecular energy, its derivatives with respect to coordinates and induced dipole (see section 1.1).There is no guarantee that coordinates stored in `Monomer`'s and coordinates in memory at `intermediate_q` are always in sync. The function `extract_q_and_write_into_ms` must be invoked if the contiguous vector of coordinates is desired at a certain point of the program execution.

Function `init_ms`

Call: `MoleculeSystem *ms = MoleculeSystem init_ms(mu, t1, t2, I1, I2, seed)`

Arguments:
`double mu` – the reduced mass of the molecule pair
`MonomerType t1` specifies the type of first monomer
`MonomerType t2` specifies the type of second monomer
`double* I1` contains inertia tensor values for first monomer. If the monomer is atom, no values will be read from the pointer, so `NULL` can be passed. Two and three values are expected for the rotor and linear molecule, respectively.
`double* I2` contains inertia tensor values for second monomer.
`size_t seed` is the seed for random number generator. A unique seed will be produced if `0` is passed.

Description: The function prepares the `MoleculeSystem` struct based on the specified monomer types, allocates the memory using `malloc` and initializes the random number generator.

Function `kinetic_energy`

Call: `double kinetic_energy(*ms)`

Arguments: `MoleculeSystem* ms`

Description: The kinetic energy function is calculated at the phase-point stored in `MoleculeSystem`. Currently, implemented for intermolecular degrees of freedom and linear molecules.

Function `Hamiltonian`

Call: `double Hamiltonian(*ms)`

Arguments: `MoleculeSystem* ms`

Description: Calls to `kinetic energy`, assembles a contiguous vector of coordinates via `extract_q_and_write_into_ms` and passes it to external `pes`.

Function `q_generator`

Call: `void q_generator(*ms, *params)`

Arguments:
`MoleculeSystem* ms`
`CalcParams *params`

Description: Generates R with density $\rho \sim R^2$ in the range [`params.sampler_Rmin`, `params.sampler_Rmax`]. The distributions of φ, ψ are $\varphi, \psi \sim U[0, 2\pi]$ and for θ is $\cos\theta \sim U[0, 1]$. Currently implemented for intermolecular degrees of freedom and linear molecules.

Function `p_generator`

Call	`void p_generator(*ms, T)`
Arguments	`MoleculeSystem* ms` `double T`
Description	Samples momenta p from distribution $\rho \sim e^{-K/kT}$ at given temperature. Calls to `p_generator_linear_molecule` and `p_generator_rotor` to sample momenta for monomers.

Function `reject`

Call	`bool reject(*ms, Temperature, pesmin)`

Arguments	`MoleculeSystem* ms`
	`double Temperature`
	`double pesmin` – the minimum value of PES

Description	Applies the rejection step to the phase-point that is stored in the `MoleculeSystem`. It presupposes that the provided phase-point is sampled from $\rho \sim e^{-K/kT}$ using `q_generator` and `p_generator` functions. The random variable $u \sim U[0,1]$ is chosen, to determine whether the current phase-point is to be accepted with probability $\rho \sim \exp(-H/kT)$.

Function `rhs`

Call	`void rhs(t, y, ydot, *user_data);`
Arguments	`UNUSED(realtype t)` `N_Vector y` stores coordinates and conjugated momenta `N_Vector ydot` is filled by function with numerical values of right-hand side of Hamilton's equations of motion at provided phase point `void *user_data` is employed to pass `MoleculeSystem*` inside the function (see section 1.2)
Description	This function is passed to `CVode` library to propagate the trajectory (see section 1.2).First, the phase-point coordinates are stored into `MoleculeSystem` struct. A contiguous vector of coordinates is assembled via `extract_q_and_write_into_ms`. Next, by calling the external function `dpes`, the derivatives of potential energy are computed and stored into `MoleculeSystem.dVdq`. The components of derivative vector are then copied into the field `Monomer.dVdq` of the corresponding monomer via the call to `extract_dVdq_and_write_into_monomers`. The right-hand side of Hamilton's equations with respect to intermolecular degrees of freedom are readily obtained and filled into `ydot`, while the derivatives with respect to monomer's coordinates are handled by `rhsMonomer` function.

Function `rhsMonomer`

Call	`void rhsMonomer(*m, *deriv);`
Arguments	`Monomer *m` `double *deriv` stores the right-hand side of Hamilton's equations of motion with respect to coordinates and momenta that correspond to the passed-in monomer
Description	In addition to differentiating the kinetic energy, the derivatives of potential energy, which are taken from `Monomer.dVdq`, are also added to compute the right-hand side. When `apply_requantization` flag is set, then the momenta in `qp` are rescaled so that angular momentum is brought to the closest half-integer. Currently, implemented only for linear molecules.

Function `j_monomer`

Call	`double j_monomer(m);`
Arguments	`Monomer m`
Description	Computes the magnitude of angular momentum of passed-in monomer. Currently, implemented only for linear molecules.

Function `torque_monomer`

Call	`double torque_monomer(m);`
Arguments	`Monomer m`
Description	Computes the magnitude of torque of passed-in monomer. Currently, implemented only for linear molecules.

Function `calculate_M0`

Call	`void calculate_M0(*ms, *params, Temperature, *m, *q);`

Arguments	`MoleculeSystem *ms`
	`CalcParams *params`
	`double Temperature`
	`double *m` – the estimate of M_0
	`double *q` – the error of the estimate

Description	By sampling from $\rho \sim e^{-K/kT}$ and rejecting some of the points using the `reject` function, `params.initialM0_npoints` phase-points are produced to estimate M_0 and its error.

MPI Function `mpi_calculate_M0`

Call	`void calculate_M0(ctx, *ms, *params, Temperature, *m, *q);`

Arguments	`MPI_Context ctx`
	`MoleculeSystem *ms`
	`CalcParams *params`
	`double Temperature`
	`double *m`
	`double *q`

Description	The task of iterating `params.initialM0_npoints` points is split equally between processes of communicator.

MPI Function `calculate_correlation_and_save`

Call	`CFnc calculate_correlation_and_save(ctx, *ms, *params, Temperature);`

Arguments	`MPI_Context ctx`
	`MoleculeSystem *ms`
	`CalcParams *params`
	`double Temperature`

Description	First, an estimate of the zeroth moment is obtained over `params.initialM0_npoints` points. The accumulation of `params.total_trajectories` individual correlation functions is divided into `params.niterations` iterations. The current aggregate estimate of the correlation function is saved to `params.cf_filename` at the end of each iteration.

`struct CalcParams`

Fields

```
PairState ps
/* sampling */
double sampler_Rmin
double sampler_Rmax
double pesmin
/* initial spectral moments check */
size_t initialM0_npoints
double partial_partition_function_ratio
/* requantization */
size_t torque_cache_len
double torque_bound
/* trajectory */
double sampling_time
double R0
double Rcut
```

```
size_t MaxTrajectoryLength
size_t CF_Length
/* correlation function array */
double *temperatures
size_t ntemperatures
```

1.1 External functions

Signatures

Supplied routines:

1. spherical decomposition for *ab initio* PES for CO_2-Ar (Kalugina/Lokshtanov)
2. spherical decomposition for *ab initio* IDS for CO_2-Ar (Kalugina/Lokshtanov)
3. spherical decomposition for full-dimensional *ab initio* PES for N_2-Ar (Finenko)
4. PIP-NN representation for *ab initio* PES surface for N_2-Ar (Finenko)
5. PIP-NN representation for *ab initio* IDS surface for N_2-Ar (Finenko)
6. spherical decomposition for long-range IDS for N_2-Ar (Wang)
7. spherical decomposition for long-range $d\mu/dr$ surface for N_2-Ar (Wang)
8. spherical decomposition for *ab initio* PES for H_2-Ar (LeRoy/Chistikov)
9. spherical decomposition for long-range IDS for H_2-Ar (Kalugina)
10. spherical decomposition for *ab initio* induced dipole for H_2-Ar (Meyer)
11. PIP-NN representation for *ab initio* IDS for H_2-Ar (Meyer/Finenko)
12. spherical decomposition for *ab initio* PES for CO_2-CO_2 (Kalugina/Lokshtanov)
13. spherical decomposition for *ab initio* IDS for CO_2-CO_2 (Kalugina/Lokshtanov)
14. spherical decomposition for *ab initio* PES for N_2-N_2 (Karman/Chistikov)
15. spherical decomposition for *ab initio* IDS for N_2-N_2 (Karman/Chistikov)
16. spherical decomposition for *ab initio* PES for N_2-H_2 (Kalugina)
17. spherical decomposition for long-range IDS for N_2-H_2 (Kalugina)
18. spherical decomposition for *ab initio* PES for CH_4-N_2 (Finenko)
19. spherical decomposition for *ab initio* IDS for CH_4-N_2 (Finenko)
20. PIP-NN representation for full-dimensional *ab initio* PES for CH_4-N_2 (Finenko)
21. spherical decomposition for *ab initio* PES for CH_4-CO_2 (Finenko)
22. spherical decomposition for *ab initio* IDS for CH_4-CO_2 (Finenko)
23. spherical decomposition for *ab initio* PES for $CO-Ar$ (Pederson)
24. spherical decomposition for *ab initio* IDS for $CO-Ar$ (Rizzo)

1.2 Interfacing with `CVode` library

2 A skeleton of the user's program

For now, let us assume that the user is supposed to use `Hawaii Hybrid` as a library, not via the configuration file to a driver program (which could be arranged in the future). Then the following structure is expected:

1. **Initialize parallel environment** (or multi-threaded environment?)
   Call `MPI_Init` to initialize MPI if desired.
2. **Initialize** `MoleculeSystem`
   Call `init_ms` specifying types of monomers, their tensors of inertia, the reduced mass of the molecule pair and the generator seed.
3. ...

3 Examples

3.1 Propagating trajectory for CO_2-Ar

3.2 Propagating trajectory for H_2-Ar while requantizing the angular momentum of H_2

3.3 Calculating the zeroth and second spectral moments of CO_2-Ar as phase-space averages using rejection-based sampler

3.4 Calculating a single correlation function for CO_2-Ar

4 Changelog

24.12.2024 `rhsMonomer`: accepts pointer so that monomer's `qp` can be changed if `apply_requantization` flag is set.

06.01.2025 `Makefile`: switched to `Makefile` from build script.

5 Todo's

- arena allocator?