Documentation for `Hawaii Hybrid` v.0.1

A. Finenko and D. Chistikov

January 21, 2025

Contents

1 Hawaii Hybrid code organization

1.1 Module hawaii

`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
`double 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 1.5).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.

`struct CFnc`

Fields `double *t`
`double *data`
`size_t len` – number of samples in `*t` and `*data`
`size_t capacity` – capacity of `*t` and `*data`
`size_t ntraj` – number of trajectories used for averaging
`double Temperature`

`struct SFnc`

Fields `double *nu`
`double *data`
`size_t len` – number of samples in `*nu` and `*data`
`size_t capacity` – capacity of `*nu` and `*data`
`size_t ntraj` – number of trajectories used for averaging
`double Temperature`

`struct Spectrum`

Fields `double *nu`
`double *data`
`size_t len` – number of samples in `*nu` and `*data`
`size_t capacity` – capacity of `*nu` and `*data`

`struct CalcParams`

Fields `PairState ps`
`/* sampling */`
`double sampler_Rmin`
`double sampler_Rmax`
`double pesmin`
`/* initial spectral moments check */`
`size_t initialM0_npoints`
`size_t initialM2_npoints`
`double partial_partition_function_ratio`

```
/* requantization */
size_t torque_cache_len
double torque_bound
/* trajectory */
double sampling_time
size_t MaxTrajectoryLength
double cvode_tolerance
/* applicable to both correlation function AND spectral function calculations */
size_t niterations
size_t total_trajectories
/*correlation function calculation ONLY */
const char *cf_filename
double Rcut
/* pr/mu calculation ONLY */
const char *sf_filename
double ApproximateFrequencyMax
double R0
/*correlation function array ONLY (NOT IMPLEMENTED)*/
double *temperatures
size_t ntemperatures
```

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, LINEAR_MOLECULE (tested) and ROTOR (tested).

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 1.6)
Description This function is passed to `CVode` library to propagate the trajectory (see section 1.6).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. Implemented for cases of LINEAR_MOLECULE (tested), LINEAR_MOLECULE_REQUANTIZED_ROTATION (not tested enough) and ROTOR (tested).

Function `compute_numerical_rhs`

Call `Array compute_numerical_rhs(*ms, order);`

Arguments `MoleculeSystem *ms`
`size_t order` – the order of the central finite-difference formula (implemented for 2, 4 or 6)
Description Computes numerically the right-hand side of the Hamiltonian equations of motion. The order corresponds to the order of variables employed in MoleculeSystem struct.

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 `invert_momenta`

Call `void invert_momenta(*ms);`
Arguments `MoleculeSystem *ms`
Description Inverts the momenta stored inside MoleculeSystem.

Function `analytic_full_partition_function_by_V`

Call `double analytic_full_partition_function_by_V(*ms, *params, Temperature)`
Arguments `MoleculeSystem *ms`
`double Temperature`
Description Returns analytically calculated value for the ratio of the partition function for the totality of states to the volume. Currently, implemented only for linear molecule-atom.

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. The average over the sampled phase-points is multiplied by the `params.partial_partition_function_ratio`, which is supposed to be a ratio of the partition function over the part of the phase space that is pertinent to the select pair state to the total partition function and multiplied by volume. This ratio can be calculated using `mpi_perform_integration` function that invokes hep library.

Function `compute_dHdp`

Call `void compute_dHdp(*ms, *dHdp);`
Arguments `MoleculeSystem *ms`
`gsl_matrix *dHdp`
Description Calls to `rhsMonomer` to fill in derivatives of Hamiltonian with respect to momenta pertaining to monomers.

Function `calculate_M2`

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

Arguments
`MoleculeSystem *ms`
`CalcParams *params`
`double Temperature`
`double *m` – the estimate of M_2
`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.initialM2_npoints` phase-points are produced to estimate M_2 and its error. The derivative of dipole with respect to coordinates is done using cental 2-point finite-difference formula. The energy is differentiated with respect to momenta using `compute_dHdp`. Here we use GSL's wrappers over BLAS to conduct matrix-by-vector multiplication. The average over the sampled phase-points is multiplied by the `params.partial_partition_function_ratio`, which is supposed to be a ratio of the partition function over the part of the phase space that is pertinent to the select pair state to the total partition function and multiplied by volume. This ratio can be calculated using `mpi_perform_integration` function that invokes hep library.

MPI Function `mpi_calculate_M0`

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

Arguments
`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. The behavior is the same as in `calculate_M0`.

MPI Function `mpi_calculate_M2`

Call `void calculate_M2(*ms, *params, Temperature, *m, *q);`

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

Description The task of iterating `params.initialM2_npoints` points is split equally between processes of communicator. The behavior is the same as in `calculate_M2`.

Function `correlation_eval`

Call `int correlation_eval(*ms, *traj, *params, *crln, *tps);`

Arguments
`MoleculeSystem *ms`
`Trajectory *traj`
`CalcParams *params`
`double *crln`
`int *tps`

Description Evaluates the mean between forward and backward correlation functions from the initial condition set in the `*ms`. It also tracks the number of turning points and returns it using the output parameter `*tps`.

MPI Function `calculate_correlation_and_save`

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

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

Description First, estimates of the zeroth and second specral moments are obtained using `params.initialM0_npoints` and `params.initialM2_npoints` points, respectively. The accumulation of `params.total_trajectories` individual correlation functions is divided into `params.niterations` iterations. The individual correlation functions are obtained using `correlation_eval` function. The current aggregate estimate of the correlation function is saved to `params.cf_filename` at the end of each iteration. The zeroth moment based on the current estimate of the correlation function is made and compared to the value obtained during static phase-space sampling. NEED to correct the calculation of the estimate of the second moment based on the current correlation function. If `FREE_AND_METASTABLE` pair state is requested, then we SHOULD divide the contributions depending on the number of turning points provided by `correlation_eval`. The tracking of the number of turning points is already implemented. The communication between processes is realized using `MPI_Allreduce` function.

MPI Function `calculate_spectral_function_using_prmu_representation`

Call `SFnc calculate_spectral_function_using_prmu_representation_and_save(*ms, *params, Temperature)`

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

Description First, we check that `params.ApproximateFrequencyMax` is less than Nyquist frequency of the signal that will be sampled with requested `params.sampling_time`. Then, based on the calculated frequency step, a length of the frequency array is calculated. Since the frequency step depends on the `params.sampling_time` and `params.MaxTrajectoryLength`, the maximum frequency of the calculated spectral function will be somewhat close to the requested `params.ApproximateFrequencyMax` (as close as possible using the frequency step). The estimates of the zeroth and second spectral moments are obtained using `params.initialM0_npoints` and `params.initialM2_npoints`, respectively. The accumulation of `params.total_trajectories` is divided into `params.niterations`. For each trajectory the intermolecular distance is set to `params.R0`. The trajectory is cut at the same distance `params.R0`. Connes apodization is applied to time dependencies of Cartesian components of dipole throughout collisional trajectory before applying Fourier transform to them. The length of the dipole array which is equal to `params.MaxTrajectoryLength` needs to be a power of 2. The current estimate of the spectral function is saved to `params.sf_filename` at the end of each iteration. The zeroth and second spectral moments are obtained from the current estimate of the spectral function and compared to the values obtained through static phase-space sampling.

Function `connes_apodization`

Call `void connes_apodization(a, sampling_time);`

Arguments
```
Array a
double sampling_time
```

Description Multiplies the provided array by Connes apodization with time-normalization factor (a) set to `sampling_time`: $A(t) = (1 - t^2/a^2)^2$. See link.

1.2 Processing results

Function `save_correlation_function`

Call `void save_correlation_function(*fp, cf, *params);`

Arguments
```
FILE *fp
CFnc cf
CalcParams *params
```

Description The correlation function values `cf.data` assumed to be unnormalized by the number of trajectories `cf.ntraj`. The file stream `fp` is truncated using `ftruncate` before writing into it.

Function `save_spectral_function`

Call `void save_spectral_function(*fp, sf, *params);`
Arguments `FILE *fp`
`SFnc sf`
`CalcParams *params`
Description The spectral function values `sf.data` assumed to be unnormalized by the number of trajectories `sf.ntraj`. The file stream `fp` is truncated using `ftruncate` before writing into it.

Function `read_correlation_function`

Call `bool read_correlation_function(*filename, *sb, *cf);`
Arguments `const char *filename`
`String_Builder *sb`
`CFnc *cf`
Description Reads the correlation function from the file `filename` expecting a header containing metainformation (the lines should begin with `'#'`) and data presented in the two-column format. The header is read into `String_Builder` without parsing. The numerical values are parsed using `fscanf` function and checked to be non-NaN. Returns `false` when either the file could no be correctly opened to a parser error was encountered.

Function `read_spectral_function`

Call `bool read_spectral_function(*filename, *sb, *sf);`
Arguments `const char *filename`
`String_Builder *sb`
`SFnc *sf`
Description Reads the spectral function from the file `filename` expecting a header containing metainformation (the lines should begin with `'#'`) and data presented in the two-column format. The header is read into `String_Builder` without parsing. The numerical values are parsed using `fscanf` function and checked to be non-NaN. Returns `false` when either the file could no be correctly opened to a parser error was encountered.

Function `integrate_composite_simpson`

Call `double integrate_composite_simpson(*x, *y, len);`
Arguments `double *x`
`double *y`
`size_t len`
Description Performs numerical integration using composite Simpson's 3/8 rule. See link.

Function `compute_M0_from_sf`

Call `double compute_M0_from_sf(sf);`
Arguments `SFnc sf`
Description Computes the zeroth moment of the spectral function using `integrate_composite_simpson`. The dimensions of the frequency and spectral function values are expected to be cm^{-1} and $\mathrm{J}\cdot\mathrm{m}^6\cdot\mathrm{s}^{-1}$, respectively. The zeroth moment is returned in units of $\mathrm{cm}^{-1}\cdot\mathrm{Amagat}^{-2}$.

Function `compute_M2_from_sf`

Call `double compute_M2_from_sf(sf);`
Arguments `SFnc sf`
Description Computes the second moment of the spectral function using `integrate_composite_simpson`. The dimensions of the frequency and spectral function values are expected to be cm^{-1} and $\mathrm{J}\cdot\mathrm{m}^6\cdot\mathrm{s}^{-1}$, respectively. The zeroth moment is returned in units of $\mathrm{cm}^{-3}\cdot\mathrm{Amagat}^{-2}$.

`struct WingParams`

Fields `double A`
`double B`
`double C`

Description Stores the parameters for the Lorentzian function shifted vertically by constant value: $y = C + A/(1 + B^2x^2)$.

`struct WingData`

Fields `size_t n`
`double *t`
`double *y`

Description

Function `wingmodel`

Call `double wingmodel(*wp, t);`
Arguments `WingParams *wp`
`double t`
Description

Function `fit_baseline`

Call `WingParams fit_baseline(*cf, EXT_RANGE_MIN);`
Arguments `CFnc *cf`
`size_t EXT_RANGE_MIN`
Description

Function `idct`

Call `double* idct(*v, len);`
Arguments `double *v`
`size_t len`
Description

Function `dct_numeric_sf`

Call `SFnc dct_numeric_sf(cf, *wp);`
Arguments `CFnc cf`
`WingParams *wp`
Description

Function `desymmetrize_sch`

Call `SFnc desymmetrize_sch(sf);`
Arguments `SFnc sf`
Description

Function `compute_alpha`

Call `Spectrum compute_alpha(sf);`
Arguments `SFnc sf`
Description

1.3 Interfacing with hep

Function `transform_variables`

Call	`void transform_variables(x, *transformed, *Jac);`

Arguments	`hep::mc_point<double> const& x`
	`double *transformed`
	`double *Jac`

Description	Transforms a point x from n-dimensional hypercube $[0,1]^n$ to a point in phase-space of the `MoleculeSystem` (which is passed in using global pointer). The following transformations are implemented

$$\begin{aligned} R &\leftarrow 1/x, \\ \theta &\leftarrow \pi x, \\ \phi &\leftarrow 2\pi x, \\ p &\leftarrow \tan(\pi(x - 1/2)). \end{aligned} \tag{1}$$

The jacobian of the transformation is accumulated along the steps. Implemented only for linear molecule-atom pair.

Function `integrand_pf`

Call	`double integrand_pf(x);`
Arguments	`hep::mc_point<double> const& x`
Description	Based on the point in hypercube returns the value of the following integrand:

$$\text{jac} \cdot \exp\left(-\frac{E}{kT}\right). \tag{2}$$

(The energy is computed for `MoleculeSystem` that is passed in using global pointer.)

Function `integrand_M0`

Call	`double integrand_M0(x);`
Arguments	`hep::mc_point<double> const& x`
Description	Based on the point in hypercube returns the value of the following integrand:

$$\text{jac} \cdot \mu^2 \cdot \exp\left(-\frac{E}{kT}\right). \tag{3}$$

(The energy is computed for `MoleculeSystem` that is passed in using global pointer. The pointer to dipole function is also assumed to be set globally.)

Function `integrand_M2`

Call	`double integrand_M2(x);`
Arguments	`hep::mc_point<double> const& x`
Description	Based on the point in hypercube returns the value of the following integrand:

$$\text{jac} \cdot \left[\mu, H\right] \cdot \exp\left(-\frac{E}{kT}\right). \tag{4}$$

The derivative of dipole with respect to coordinates is computed using 2-point central finite-difference formula. (The energy is computed for `MoleculeSystem` that is passed in using global pointer. The pointer to dipole function is also assumed to be set globally.)

Function `mpi_perform_integration`

Call `double mpi_perform_integration(*ms, integrand, *params, Temperature, niterations, npoints, *m, *q);`
Arguments `MoleculeSystem *ms`
`Integrand integrand` – `typedef double (*Integrand)(hep::mc_point<double> const&)`
`CalcParams *params`
`double Temperature`
`size_t niterations`
`size_t npoints`
`double *m`
`double *q`
Description Evaluates the integral of requested `integrand`

1.4 Utility

`struct Array`

Fields `double *data`
`size_t n`
Description Sized array in dynamic memory.

Function `create_array`

Call `Array create_array(n);`
Arguments `size_t n`
Description

Function `init_array`

Call `void init_array(n);`
Arguments `Array *a`
`double *data`
`size_t n`
Description

Function `free_array`

Call `void free_array(n);`
Arguments `Array *a`
Description

`struct String_Builder`

Fields `char *nu`
`size_t count`
`size_t capacity`

Function `sb_append`

Call `void sb_append(*sb, *line, n);`
Arguments `String_Builder *sb`
`const char *line`
`size_t n`
Description

Function `free_sb`

Call	`void free_sb(sb);`
Arguments	`String_Builder sb`
Description	

1.5 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 *ab initio* PES for CH_4-CO_2 (Finenko)
4. spherical decomposition for *ab initio* IDS for CH_4-CO_2 (Finenko)
5. spherical decomposition for *ab initio* PES for H_2-Ar (LeRoy/Chistikov)

Routines to be added in the future:

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

1.6 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**
   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

See the file `examples/trajectory_co2_ar.cpp`.

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

3.5 Calculating a spectral function using p_r/μ-representation for CO_2-Ar

h

3.6 Processing 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.

08.01.2025 added MPI library which is hidden by the guard macro, implemented `mpi_calculate_M0`. Now the MPI and non-MPI versions of `hawaii.c` are compiled into two object files.

09.01.2025 implemented `calculate_correlation_and_save`.

09.01.2025 Porting some of `hep`-functionality in `hep_hawaii.cpp`. Zeroth moment can be calculated using adaptive Monte Carlo integration over phase space.

10.01.2025 Tracking the number of turning points is added to `correlation_eval`.

10.01.2025 PES and its derivatives for CH_4-CO_2 are adapted.

11.01.2025 fixing error inverting momenta in `correlation_eval`. Calculating correlation function for CO_2-Ar seems to be working correctly. Need to run a calculation with larger number of trajectories.

12.01.2025 added higher-order finite-difference formulas for differentiating energy: `compute_numerical_rhs`

12.01.2025 trajectory for CH_4-CO_2 using kinetic energy only works correctly. Angles transformation and its jacobian needs to be adapted from FUNCHAL to account for different order of coordinates.

15.01.2025 Run calculation of correlation function for CO_2-Ar for 10.000.000 trajectories. The correlation function is in close agreement with the previous results. Now moving on to adapting code for processing correlation functions in `hawaii`.

17.01.2025 Adapted functions related to processing correlation function from FUNCHAL. Code implementing `loess` algorithm is left as-is because of the use of Eigen3.

19.01.2025 Added checks for NaN values of dipole in `correlation_eval` to avoid the corruption of the correlation function estimate. At least one corrupted value of correlation function for CO_2-Ar has occurred during the calculation of 10.000.000 trajectories. The reason for the occurence of the NaN is unknown. Should investigate when initial condition leading to NaN values is found. Could this NaN be the consequence of overflow in `generate_normal`?

19.01.2025 Implemented `calculate_spectral_function_using_prmu_representation_and_save` function and tested that it produces correct result for CO_2-Ar at 300 K.

20.01.2025 Enable `switch-enum` option for compiler to invoke warning in the switch-cases where one of the cases of the enum is not explicitly handled even though default case is present.

20.01.2025 caught a possible (but really rare) overflow error in the implementation of Box-Mueller algorithm for sampling normally distributed variable: `generate_normal`.

5 Todo's

- During the prmu-calculation for CO2-Ar on cluster, the file with temporary result is not written. Could it be the fprintf buffer not flushing?
- Maybe we need a check that processes in the communicator get different seeds each?
- How to organize and store the calculated results (cfs and sfs) for examples?
- Arena Allocator for storing some temporary strings and small arrays. At the moment, there are waaaay too many mallocs/frees.
- Calculating second moment using `hep`-functionality and using rejection sampling
- Allow to calculate an array of correlation functions
- Saving the contributions of free and metastable states in correlation function separately. How can we pass in the information about the desired filenames? How to organize the storage for these contributions?
- The information about the number of turning points could be saved into gsl_histogram, and then partially displayed in the output file and saved to file system
- We are using MPI_Allreduce mechanism for broadcasting. This should be changed to direct MPI_Send / MPI_Recv calls.
- Can the `hawaii.h` file be used without constructing the MoleculeSystem? I moved code for processing correlation function there, but it can't be used without creating stubs for PES/dPES functions. Should `hawaii` create stubs when certain macro is in place?
- In case when correlation function is calculated for bound states, we can separate the contributions for bound-bound and bound-free transitions (Fakhardji trick)
- In case when correlation function is calculated for bound states, we should use the Zimmermann trick (use any point of trajectory as a 'starting point' of the correlation function). This should be the default approach for this correlation.
- Adapt transforming angles between for CH4-containing systems
- How to organize Makefiles for use in different environments (several clusters)? If we push the changes to remote git repo on the remote machine, the local changes get overriden, thus removing the changes made to local Makefile. Annoying...

- I would like to make a graphical shell in which it will be possible to set parameters for calculation (for example, in the form of a list with available options for each option). The program is given paths to files (or drag-and-dropped) with potential and dipole functions and it assembles an executable file. Here I make an assumption that it is better to have the potential to be statically compiled with the rest of the code rather than dynamically loading it from library. It would probably be nice to be able to run the trajectory program from graphical shell locally and display the calculation result. If the correlation function is calculated, then there is no need to add its processing within the same "stage". Let the correlation function be calculated separately, then the result should be loaded and converted into a spectrum. In the case of calculating the spectral function in the pr-mu representation, smoothing is also required, it is also carried out in a separate "stage". This would enable us to have a starting point for a graphical shell that can demonstrate some of the features of the library without overcomplicating from the very beginning. Later on, we could think about establishing TCP connection between shell and the main program running remotely enabling the user to setup the calculation with the desired parameters and monitor the calculation.