MQCT 2024 User Guide

by Bikramaditya Mandal, Dulat Bostan, Carolin Joy and Dmitri Babikov[1]

Chemistry Department, Wehr Chemistry Building, Marquette University, Milwaukee, Wisconsin 53201-1881, USA

Content:

[1]Author to whom all correspondence should be addressed; electronic mail: dmitri.babikov@mu.edu

I. Molecular System Types and Rotation-Vibration Wavefunctions Included in MQCT

In MQCT program there are ten system types, summarized in Table 1, from the simplest rigid-diatom + atom, to the most general case of two asymmetric-top rotor molecules. For each system, MQCT calculations can be initiated by indicating the rotational and vibrational constants listed in the Table. Those are used by the code to set up and diagonalize Hamiltonian matrix for rotational motion (using the basis set of symmetric-top functions) in order to determine the rotational states of the system (energies and wave-functions).

Table 1: Ten types of systems handled by MQCT, with required and optional input data

SYS_TYPE	**Collision Partners**	**Required Constants**	**Channel Labels**	**Optional Input**
1	rigid diatom + atom	B_e, D_e	J	
2	vibrating diatom + atom	B_e, D_e, ω_e, x_e	j, v	vibrational functions, non-equidistant grid
3	symmetric top + atom	A, C	j, k, ε	
4	asymmetric top + atom	A, B, C	j, k_a, k_c	expansion over sym. top basis
5	rigid diatom + rigid diatom	B_{e1}, D_{e1}, B_{e2}, D_{e2}	j_1, j_2	
6	vibrating diatom + vibrating diatom	B_{e1}, D_{e1}, ω_{e1}, x_{e1}, B_{e2}, D_{e2}, ω_{e2}, x_{e2}	j_1, v_1, j_2, v_2	vibrational functions, non-equidistant grid
7	symmetric top + rigid diatom	A, C, B_e, D_e	j_1, k_1, ε_1, j_2	
8	asymmetric top + rigid diatom	A, B, C, B_e, D_e	j_1, k_{a1}, k_{c1}, j_2	expansion over sym. top basis
9	asymmetric top + symmetric top	A_1, B_1, C_1, A_2, C_2	j_1, k_{a1}, k_{c1}, j_2, k_2, ε_2	expansion over sym. top basis
0	asymmetric top + asymmetric top	A_1, B_1, C_1, A_2, B_2, C_2	j_1, k_{a1}, k_{c1}, j_2, k_{a2}, k_{c2}	expansion over sym. top basis

For a general asymmetric top molecule, the values of *A*, *B* and *C* should correspond to the *x*, *y*, and *z*-axes (not necessarily in decreasing order) and must be consistent with reference orientation of the potential energy surface for the system (*i.e.*, all Euler angles set to zero). If the molecule has a symmetry axis, user can take advantage of this, as usual, by choosing the reference orientation such that the symmetry axis is aligned with *z*, to enable independent calculations for para- and ortho-states of the molecule (and thus to reduce the rotational basis set size, accordingly). When symmetric tops are involved in collisions, the case of an oblate top is handled in a standard way, with rotational constants indicated such that *A* > C, while for a prolate top the input should be in the form *A* < *C*, opposite to the standard notation. In both cases it is assumed that *B* = *A*. Although the case of spherical top is not explicitly included, it can also be handled, by entering equal values for *A* and *C*.

The option of invoking the externally computed user-supplied rotational-vibrational states (*e.g.*, vibrational wave functions defined on a non-equidistant grid of points, or vibrationally distorted rotational states, such as those of Kyro model Hamiltonian, expanded over basis set of symmetric-top functions) is indicated in Table 1, where available, and is described in more detail in Sec. VI below (see Table 7).

Explicit form of rotational wavefunction constructed by MQCT is given in Table 2 for all system types. The ranges of quantum numbers are: $|j_1 - j_2| \leq j \leq j_1 + j_2$, $0 \leq k_1 \leq j_1$ (positive only), $\varepsilon_1 = \pm$ (parity),

$-j \leq m \leq +j$, $0 \leq k_2 \leq j_2$ (positive only), $\varepsilon_2 = \pm$ (parity).

For symmetric top molecules (`SYS_TYPE=3,7,9`) the input values of *inversion parity* are $\varepsilon = 0$ for positive parity $(+)$ and $\varepsilon = 1$ for negative parity $(-)$, since in the code the parity is implemented as a phase factor $(-1)^{\varepsilon}$. For $k = 0$ only positive parity is allowed ($\varepsilon = 0$). Also note that for asymmetric top molecules (`SYS_TYPE=4,8,9,0`) there are $2j + 1$ states of the type $j_{k_a k_c}$ with specific sets of coefficients b_k, for each molecule (expansion over the basis set of symmetric-top eigenstates). These are automatically computed by the code or, alternatively, users can supply their own sets of coefficients b_k pre-computed externally (*e.g.*, for Kyro-type Hamiltonian) for one (`SYS_TYPE=4,8,9`) or both (`SYS_TYPE=0`) asymmetric top collision partners (see Sec. VI, Table 7 and example input files).

Vibrational wavefunctions of diatomics (`SYS_TYPE=2,6`) are defined on a grid of points, equidistant by default but, a user-supplied non-equidistant grid can also be used (see Sec. VI, Table 7). In both cases the integrals of vibrational wavefunctions include the weights of grid points. MQCT code will first try to search for the file `USER_DEFINED_BASIS.DAT` which contains energies, vibrational wavefunctions, along with the weights of grid points, for one (`SYS_TYPE=2`) or both (`SYS_TYPE=6`) collision partners, see Sec. VI. If not provided, MQCT would use Morse parameters from the input file to construct the vibrational wavefunctions of non-rotating Morse oscillator. If those are not provided, it would compute the values of Morse parameters from spectroscopic coefficients indicated in the input file. Vibrational energies are computed using a standard Dunham's formula. Vibrational wave functions of Morse oscillator ($j = 0$, neglecting ro-vibrational distortion) are computed using recurrent relations and are used to compute matrix elements for the corresponding state-to-state transitions.

Table 2: Wavefunctions of rotational-vibrational eigenstates in MQCT calculations of different system types

System type	Representation of Wavefunctions
1	$\|\, jm\rangle = \mathrm{Y}^{j}_{m}(\beta, 0)$
2	$\|\, jmv\rangle = \psi_v(r) \times \mathrm{Y}^{j}_{m}(\beta, 0)$
3	$\|\, jmk\varepsilon\rangle = \sqrt{\frac{2j+1}{8\pi^2}} \sqrt{\frac{1}{2\left(1+\delta_{k,0}\right)}} \left[\mathrm{D}^{j*}_{m,+k}(0,\beta,\gamma) + \varepsilon \mathrm{D}^{j*}_{m,-k}(0,\beta,\gamma)\right]$
4	$\|\, jm\rangle = \sqrt{\frac{2j+1}{8\pi^2}} \left(\sum_{k=-j}^{+j_1} b_k \mathrm{D}^{j*}_{m,+k}(0,\beta,\gamma) \right)$
5	$\|\, jmj_1j_2\rangle = \sum_{m_1=-j_1}^{+j_1} \mathrm{C}^{j,m}_{j_1,m_1,j_2,m-m_1} \mathrm{Y}^{j_1}_{m_1}(\beta_1, 0) \times \mathrm{Y}^{j_2}_{m-m_1}(\beta_2, \alpha_2)$
6	$\|\, jmj_1j_2v_1v_2\rangle = \psi_{v_1}(r_1) \times \psi_{v_2}(r_2) \times \sum_{m_1=-j_1}^{+j_1} \mathrm{C}^{j,m}_{j_1,m_1,j_2,m-m_1} \mathrm{Y}^{j_1}_{m_1}(\beta_1, 0) \times \mathrm{Y}^{j_2}_{m-m_1}(\beta_2, \alpha_2)$
7	$\|\, jmj_1k_1\varepsilon_1j_2\rangle = \sqrt{\frac{2j_1+1}{8\pi^2}} \sqrt{\frac{1}{2\left(1+\delta_{k_1,0}\right)}} \sum_{m_1=-j_1}^{+j_1} \mathrm{C}^{j,m}_{j_1,m_1,j_2,m-m_1} \left[\mathrm{D}^{j_1*}_{m_1,+k_1}(0,\beta_1,\gamma_1) + \varepsilon_1 \mathrm{D}^{j_1*}_{m_1,-k_1}(0,\beta_1,\gamma_1)\right] \times \mathrm{Y}^{j_2}_{m-m_1}(\beta_2, \alpha_2)$
8	$\|\, jmj_1j_2\rangle = \sqrt{\frac{2j_1+1}{8\pi^2}} \sum_{m_1=-j_1}^{+j_1} \mathrm{C}^{j,m}_{j_1,m_1,j_2,m-m_1} \left(\sum_{k_1=-j_1}^{+j_1} b_{k_1} \mathrm{D}^{j_1*}_{m_1,k_1}(0,\beta_1,\gamma_1) \right) \times \mathrm{Y}^{j_2}_{m-m_1}(\beta_2, \alpha_2)$
9	$\|\, jmj_1j_2k_2\varepsilon_2\rangle = \sqrt{\frac{2j_1+1}{8\pi^2}} \sqrt{\frac{2j_2+1}{8\pi^2}} \sqrt{\frac{1}{2\left(1+\delta_{k_2,0}\right)}} \sum_{m_1=-j_1}^{+j_1} \mathrm{C}^{j,m}_{j_1,m_1,j_2,m-m_1} \left(\sum_{k_1=-j_1}^{+j_1} b_{k_1} \mathrm{D}^{j_1*}_{m_1,k_1}(0,\beta_1,\gamma_1) \right) \times \left[\mathrm{D}^{j_2*}_{m-m_1,+k_2}(\alpha_2,\beta_2,\gamma_2) + \varepsilon_2 \mathrm{D}^{j_2*}_{m-m_1,-k_2}(\alpha_2,\beta_2,\gamma_2)\right]$
0	$\|\, jmj_1j_2\rangle = \sqrt{\frac{2j_1+1}{8\pi^2}} \sqrt{\frac{2j_2+1}{8\pi^2}} \sum_{m_1=-j_1}^{+j_1} \mathrm{C}^{j,m}_{j_1,m_1,j_2,m-m_1} \left(\sum_{k_1=-j_1}^{+j_1} b_{k_1} \mathrm{D}^{j_1*}_{m_1,k_1}(0,\beta_1,\gamma_1) \right) \times \left(\sum_{k_2=-j_2}^{+j_2} b_{k_2} \mathrm{D}^{j_2*}_{m-m_1,k_2}(\alpha_2,\beta_2,\gamma_2) \right)$

In the code we use Euler angles that correspond to active rotations of each molecule, according to the intrinsic $z-y'-z''$ convention. The expression for Wigner D-function is $\mathrm{D}^{j}_{mk}(\alpha,\beta,\gamma) = e^{-im\alpha}\mathrm{d}^{j}_{mk}(\beta)e^{-ik\gamma}$, where j is angular momentum, m is its projection onto z-axis (that connects centers of mass of two collision partners), and k is projection of j onto the molecule-fixed axis z' (such as symmetry axis, or one of the principal axis of inertia). Eigenfunction of symmetric top is D^{j*}_{mk}, with eigenvalues of the corresponding operators given by $\hat{\mathcal{J}}^2\mathrm{D}^{j*}_{mk} = j(j+1)\,\mathrm{D}^{j*}_{mk}$, $\hat{\mathcal{J}}_{z'}\mathrm{D}^{j*}_{mk} = k\,\mathrm{D}^{j*}_{mk}$, and $\hat{\mathcal{J}}_{z}\mathrm{D}^{j*}_{mk} = m\,\mathrm{D}^{j*}_{mk}$. Eigenfunction of a linear top is $\mathrm{Y}^{j}_{m}=\sqrt{(2j+1)/(4\pi)}\,\mathrm{D}^{j*}_{m0}$.

II. Structure of the Input File, Required and Optional Keywords, Default Values

Initial conditions for classical and quantum degrees of freedom in the system are defined separately in two blocks of the input file, called `$SYSTEM` and `$BASIS`. Potential energy surface describes interaction between quantum and classical parts of the system and is defined in the third block of the input file, called `$POTENTIAL`. This block should be the last in the input file, while the order of the first two blocks is interchangeable since they are independent. Example below sets up the input for calculations of H_2O + He rotational excitation (a complete list of required and optional keywords is given in Tables 3, 4 and 5 below):

```
$BASIS
     SYS_TYPE=4, A=27.877, B=9.285, C=14.512,
     NMB_CHNLS=6,
     CHNLS_LIST=0,0,0, 1,0,1, 1,1,0, 1,1,1, 2,0,2, 2,2,1,
     INIT_CHNL=0,0,0
$END

$SYSTEM
     LABEL="H2O+He", MASS_RED=3.2748, RMIN=4.5, RMAX=20.0,
     NMB_ENERGS=1, U_ENERGY=200., JTOTL=0, JTOTU=100,
     TIME_STEP=15.0, TIME_LIM=3.5E+6
$END

$POTENTIAL
     E_UNITS=A.U., R_UNITS=A.U.,
     GRD_R=50, GRD_ANG1=40, GRD_ANG2=20, GRD_ANG3=40
$END
```

Each block starts with its name and is finalized by the $END. All entries inside the block are separated by coma. You can have as many spaces, lines or tabs as you want between the keywords and their values, all is taken care of by the parser. First keyword of the block $BASIS indicates the type of the system, followed by three values of the rotational constants of the molecule (in the units of wavenumber). Here the rotational states of the asymmetric top rotor are labelled using the standard notation: j, k_a, k_c (where k_a and k_c are projections of j onto the principal axis with smallest and largest moments of inertia, respectively). Another useful optional keyword allows forming a basis set out of all states below given energy, for example: EMAX=135.4 commands to include in the basis set all states at energies below 135.4 cm^{-1} (which, again, would be the same six channels). Rotational quenching calculations can be initiated by choosing an excited state as initial, for example: INIT_CHNL=2,0,2.

The block $SYSTEM, besides the text LABEL, contains reduced mass of the collision partners (MASS_RED, in atomic mass units), the minimum and maximum values of distance between collision partners to initialize and terminate trajectories (RMIN and RMAX in the units of Bohr), the number of collision energies to compute (NMB_ENERGS) and their effective values (U_ENERGY in wavenumbers), the range of total angular momentum quantum number *J* (from JTOTL to JTOTU), propagation time-step TIME_STEP and the time limit TIME_LIM to terminate trajectories (both in atomic units). One useful option is to set the maximum value of impact parameter, in the units of Bohr, for example: B_IMPCT=9.50. If this is specified, then the upper and lower limits of *J* are not required (JTOTL and JTOTU are ignored, even if indicated). This option is convenient for calculations in a broad energy range, since the maximum impact parameter is less sensitive to collision energy. At each collision energy, the upper limit of *J* is determined individually, based on the indicated value of maximum impact parameter. By default, 4th-order Runge-Kutta method is employed to propagate MQCT trajectories (together with quantum equations for state populations) using a constant step-size TIME_STEP. Trajectories in MQCT are integrated trough the interaction region and are terminated when the molecule-molecule distance exceeds RMAX. If optional time-limit is indicated (by TIME_LIM, as in the example above) the trajectory is terminated as soon as either condition is fulfilled. This is recommended at low collision energies, when orbiting trajectories are possible (analogous to quantum scattering resonances, see below).

In the block $POTENTIAL, the units of energy and distance for the potential energy surface subroutine (supplied by the user, see below) are indicated first, followed by the number of quadrature points for integration of matrix elements for state-to-state transitions. For the units of distance (keyword R_UNITS) Bohr and Angstrom are available, as defined by A.U. and ANGS, respectively. For energy units (keyword E_UNITS) possible values are A.U., CM-1, KLVN and KCAL that correspond to Hartree, wavenumber, Kelvin and kilocalorie per mole, respectively. Note that angles are always assumed to be in radians. By default, state-to-state transition matrix elements are computed by direct numerical integration over all internal degrees of freedom. The number of quadrature points should be indicated for each angular coordinate, as in the example above (three Euler angles for an asymmetric top rotor, such as H_2O). Integration over angles is carried out at each point of the molecule-molecule distance grid (GRD_R points in the range between RMIN and RMAX). In the collision dynamics calculations, each matrix element is splined over the distance between the

grid points, using 3rd-order one-dimensional spline. Optionally, computed matrix elements can be saved to a file, using keyword SAVE_MTRX=YES. By default, this would be in the binary form (unformatted). Formatted matrix output can be requested by the keyword UNFORMAT=NO, *e.g.*, for the inspection by the user.

The three tables below give a comprehensive list of all required and optional input parameters for three parts of the input file, the blocks $BASIS, $SYSTEM, and $POTENTIAL. Default values are indicated, where applicable. A keyword must be specified only if the value different from the default is desired. Datatype "real" corresponds to double precision. The values "YES" and "NO" correspond to logical datatype.

Table 3: Description of keywords for the block $BASIS

Keyword	**Type, Range, Units**	**Description**	**Relevant** SYS_TYPE
SYS_TYPE	integer, 0 to 9	1 -- rigid diatom + atom 2 -- vibrating diatom + atom 3 -- symmetric top + atom 4 -- asymmetric top + atom 5 -- diatom + diatom (both rigid) 6 -- vibrating diatom + vibrating diatom 7 -- symmetric top + diatom (rigid) 8 -- asymmetric top + diatom (rigid) 9 -- asymmetric top + symmetric top 0 -- asymmetric top + asymmetric top	
NMB_CHNLS	Integer	Number of channels	all
CHNLS_LIST	Integers	Quantum numbers to specify channels	all
INIT_CHNL	Integers	Quantum numbers to specify the initial channel	all
EXCLUDE_STATES	default is "NO"	User can exclude specified states from the basis set, e.g., the weekly coupled states or the states with certain *j* and *m*.	all
BE,DE	real, positive, cm^{-1}	Rotational constants B_e, D_e (exp. format for D_e)	1,2,7,8
A,B,C	real, positive, cm^{-1}	Rotational constants *A*, *B*, *C* (for *x*, *y* and *z*, as in PES)	3,4,7,8
BE1,DE1	real, positive, cm^{-1}	Rotational constants B_e, D_e for molecule #1	5,6
BE2,DE2	real, positive, cm^{-1}	Rotational constants B_e, D_e for molecule #2	5,6,7,8
A1,B1,C1	real, positive, cm^{-1}	Rotational constants *A*, *B*, *C* for molecule #1 (for *x*, *y* and *z*)	9,0
A2,B2,C2	real, positive, cm^{-1}	Rotational constants *A*, *B*, *C* for molecule #2 (for *x*, *y* and *z*)	9,0
WE,XE	real, positive, cm^{-1}	Vibrational constants ω_e, x_e	2

WE1,XE1	real, positive, cm^{-1}	Vibrational constants ω_e, x_e for molecule #1	6
WE2,XE2	real, positive, cm^{-1}	Vibrational constants ω_e, x_e for molecule #2	6
JMIN,JMAX	Integer	The range of rotational number *j* included in the basis set; optional, used to avoid listing all levels individually	1,2
VMIN,VMAX	Integer	The range of vibrational number *v* included in basis set; optional, used to avoid listing all levels individually	2
JMIN1,JMAX1, JMIN2,JMAX2	Integer	The range of rotational numbers included in basis sets for molecules #1, 2; option, used to avoid listing the levels	5,6
VMIN1,VMAX1, VMIN2,VMAX2	Integer	The range of vibrational numbers included in basis sets for molecules #1, 2; option, used to avoid listing the levels	6
NCHL1,NCHL2	Integer	Number of lower energy channels included for molecules #1, 2; optional, used to avoid listing the levels individually	5,6,7,8,9,0
EMAX	real, cm^{-1}	Channel energy cutoff. Only the states below it are included in calculations; option, to avoid listing all levels	all
EMAX1,EMAX2	real, cm^{-1}	Channel energy cutoffs for molecules #1 and #2; only the states below this energy are included in calculations	5,6,7,8,9,0
SYMMETRY	default is "NO"	If "YES", only the states coupled to the initial state are retained in the basis (ortho vs para states).	3,4,7,8,9,0
ATOMIC_MASSES	real, positive, amu	Masses of atoms in the diatomic; to determine COM	2,6
MORSE_DEPTH	real, positive, cm^{-1}	Depth parameter of Morse oscillator (dissociation energy)	2,6
MORSE_WIDTH	real, positive, Bohr	Width parameter of Morse oscillator	2,6
MORSE_POSITN	real, positive, Bohr	Equilibrium Distance of Morse oscillator	2,6
RMIN_VIBGRID	real, positive, Bohr	Minimum diatomic bond length	2,6
RMAX_VIBGRID	real, positive, Bohr	Maximum diatomic bond length	2,6
WGHT_POSPAR	real, positive, ≤1.d0	Weight of positive total parity wave functions in the case of identical particles collision. Default is 1.d0	5,6,0
CHNL_ENERGS	real, cm^{-1}	User can list energies of states (e.g., computed externally)	all
LEVELS_FILE	default is "NO"	If "YES", user provides energies, wave functions and assignments of externally-computed states, in a file	all
CS_APPROX	default is "NO"	If "YES", the coupled-states approximation will be used	all
IDENTICAL	default is "NO"	If "YES", collision partners are treated as indistinguishable	5,6,0
PRINT_STATES	default is "NO"	If "YES", prints out states and structure of coupling matrix	all
SAVE_TRAJECT	default is "NO"	If "YES", trajectory info is saved to files for AT-MQCT	all
AT_APPROX	default is "NO"	If "YES", AT-MQCT is done using saved trajectory info	all

Table 4: Description of keywords for the block `$SYSTEM`

Keyword	Type, Range, Units	Description
LABEL	Text	Name of your job
MASS_RED	real, positive, amu	Reduced mass of two scattering partners
RMIN,RMAX	real, positive, Bohr	Minimum and maximum values of distance between partners
B_IMPCT	real, positive, Bohr	Maximum value of collision impact parameter
JTOTL,JTOTU	Integer	Lower and upper limits of total angular momentum J
NMB_ENERGS	Integer	Number of collision energy values U to propagate
U_ENERGY	real, positive, cm^{-1}	List of collision energies U to propagate
UMIN,UMAX	real, positive, cm^{-1}	Minimum and maximum collision energies U
DU	real, positive, cm^{-1}	Step size for setting collision energies U
TIME_STEP	real, positive, au	Propagation time step for RK4, or maximum time step allowed for ODEINT
MIN_TMSTP	real, positive, au	Minimum time step allowed in ODEINT
SINGLE_STEP	default is "NO"	Propagates each trajectory in one step using ODEINT
TIME_LIM	real, positive, au	Time limit for propagation
EPS_ODEINT	real, positive, <1.d0	Relative error for step-size control in ODEINT (exp. format)
EPS_MONCAR	real, positive, %	Desirable error in Monte Carlo sampling of initial conditions
PROPAGATOR	Text	RK4 is default, ODEINT is optional
NMB_LOOPS	integer, default is 1	Number of full loops (360 deg.) to propagate for orbiting trajectories
NMB_OSCIL	integer, default is 1	Number of outer turning points to propagate for oscillating trajectories
NO_RESONANCE	default is "NO"	If "YES", orbiting trajectories are removed from analysis
DIFF_CROSS	default is "NO"	If "YES", differential cross section is computed (elastic only)
ANG_RES	integer, default is 1000	Number of points for angular resolution of the differential cross section
MONTE_CARLO	default is "NO"	If "YES", initial conditions are sampled randomly
NMB_TRAJ	integer, default is 100	Number of trajectories to sample using Monte Carlo (total number of trajectories)
CHECK_POINT	Integer	Wall clock time (minutes after the start) to start writing a checkpoint file
RESTART	default is "NO"	If "YES", program will start from a check point file
PRN_TRJCT	Integer	Indicates the value of ℓ for which all the trajectory data will be printed
PRN_JM	two integers	Indicates the desired values of j and m for the option above
MPI_PERTRAJ	integer, default is 1	Number of CPUs (MPI tasks) to use for calculations of each trajectory
ENERGY_ERROR	default is "YES"	"NO" disables calculations of energy conservation error (for speedup, CS or AT-MQCT)
AT_ADAPTOL	real, positive	If indicated, RK4 with adaptive step size is used for AT-MQCT, with this tolerance
POLAR_ANGLE	default is "NO"	If "Yes", right hand side is computed and the equation of motion for Θ is propagated
DL	Integer	Step size for orbital angular momentum ℓ, default is 1
PROB_SPLINE	default is "NO"	If "YES", transition probabilities are interpolated to obtain the values skipped by DL
DL_LR	integer, default is 1	DL for the long-range part of the interaction potential (normally DL_LR > DL)
B_SWITCH	real, positive	Impact parameter (in Bohr) where the sampling switches from DL to DL_LR

Table 5: Description of keywords for the block $POTENTIAL

Keyword	**Type, Range, Units**	**Description**	**Relevant** SYS_TYPE
READ_MTRX	default is "NO"	If "YES", read the potential coupling matrix from file	all
SAVE_MTRX	default is "NO"	If "YES", write the potential coupling matrix to file	all
UNFORMAT	default is "YES"	Saves matrix in binary form; set "NO" to save it as text	all
PROG_RUN	default is "YES"	Propagates trajectories; set "NO" to compute matrix only	all
E_UNITS	Text	Energy units of PES: "A.U.", "CM-1", "KCAL" or "KLVN"	all
R_UNITS	Text	Distance units for supplied PES: "A.U." or "ANGS"	all
GRD_R	Integer	Number of points for *R*-grid	all
GRD_VIB	integer / two integers	Number of points for vibrational grid / grids	2 / 6
GRD_ANG1	integer / two integers	Number of points for α-grid / grids	1,2,3,4 / 5,6,7,8,9,0
GRD_ANG2	integer / two integers	Number of points for β-grid / grids	1,2,3,4 / 5,6,7,8,9,0
GRD_ANG3	integer / two integers	Number of points for γ-grid / grids	3,4 / 5,6,7,8,9,0
VGRID_FILE	default is "NO"	If "YES", PES values at grid points are stored to/read from file	6,7,8,9,0
EXPANSION	default is "NO"	If "YES", the PES is represented by expansion over basis	except 2, 6
NMB_TERMS	Integer	Number of PES expansion terms	except 2, 6
TERMS	sets of integers	List of the expansion terms (labeled appropriately)	except 2, 6
TERMS_FILE	default is "NO"	If "YES", reads externally-computed PES expansion terms	except 2, 6
CALC_EXPANSION	default is "NO"	If "YES", the expansion coefficients are computed.	except 2, 6
IR_BGN,IR_FIN	Integer	The range of *R*-grid points used, defaults are 1 to GRD_R	all
RGRID_EQDS	default is "YES"	If "NO", non-equidistant *R*-grid is generated by the code	all
RGRID_FILE	default is "NO"	If "YES", user-defined *R*-grid is read from file	all
L_MAX	Integer	Maximum value of index λ in the expansion of the PES	all
M_MAX	Integer	Maximum value of index μ in the expansion of the PES	3,4
L1_MIN,L1_MAX	Integer	Min. and max. values of index λ_1 in the expansion of the PES	5,7,8,9,0
L2_MIN,L2_MAX	Integer	Min. and max. values of index λ_2 in the expansion of the PES	5,7,8,9,0
M1_MAX	Integer	Maximum value of index μ_1 in the expansion of the PES	5,7,8
M2_MAX	Integer	Maximum value of index μ_2 in the expansion of the PES	9,0
ODD_L1L2L	default is "NO"	Terms with odd values of $\lambda_1+\lambda_2+\lambda$ would be included	5,7,8,9,0
AXL_SYM1	integer, default is 1	Only μ_1 that are multiples of AXL_SYM1 are included	3,4,7,8,9,0
AXL_SYM2	integer, default is 1	Only μ_2 that are multiples of AXL_SYM2 are included	7,8,9,0
EQU_SYM1	integer, default is 1	Only λ_1 that are multiples of EQU_SYM1 are included	1,3,4,5,7,8,9,0
EQU_SYM2	integer, default is 1	Only λ_2 that are multiples of EQU_SYM2 are included	5,7,8,9,0

IDENTICAL_PES	default is "NO"	If "Yes", the number of expansion terms is reduced by symmetry	5,0
PRINT_DIAGONAL	default is "NO"	If "YES", prints diagonal elements of transition matrix	all
PARALLEL_IO	default is "NO"	If "YES", read and write matrix elements in parallel	all
EQ_ANG_GRID	default is "YES"	Equidistant grids are used for α and γ (if "No", Gauss-Legendre)	3,4,5,6,7,8,9,0
MTRX_PATH	default is "NO"	If "YES", the matrix file/s is/are read from location indicated in the file named "Matrix_Path.DAT"	all
W3J_RECURSIVE	default is "NO"	If "YES" a recursive algorithm is used to compute Wigner 3j-sysbols (instead of a simple algorithm). Works better for larger *j*.	all
MIJ_CUTOFF	real, default is 1.d-12 cm^{-1}	Neglects matrix elements that are below this value	all
RETRUNCATE	default is "NO"	If "YES", the matrix is re-truncated/rewritten using MIJ_CUTOFF	all
CUTOFF_R1	real, default is 6.d0 Bohr	Indicates first value of *R* for matrix analysis and truncation	all
CUTOFF_R2	real, default is 8.d0 Bohr	Indicates second value of *R* for matrix analysis and truncation	all
MIJ_SHIFT	default is "YES"	Shifts each matrix elements by its value at the last point of *R*-grid	all
CALC_RMS	default is "NO"	If "YES", RMS of PES expansion is computed for one value of *R*	1,3,4,5,7,8,9,0
RMS_GRD1	integer, default is 20	Sparse grid for RMS calculations, in α	5,7,8,9,0
RMS_GRD2	integer, default is 10	Sparse grid for RMS calculations, in β	1,3,4,5,7,8,9,0
RMS_GRD3	integer, default is 20	Sparse grid for RMS calculations, in γ	3,4,5,7,8,9,0
RMS_R	real, default is 8.d0 Bohr	Indicates the value of *R* for RMS calculations	1,3,4,5,7,8,9,0
REBALANCE	default is "NO"	If "YES", to equalize the load, the matrix elements are computed for two values of *R* given by CUTOFF_R1 and CUTOFF_R2.	except 2, 6
BALANCED_MIJ	default is "NO"	If "YES", the matrix elements are computed for all values of *R*-grid using information from the REBALANCE step.	except 2, 6

III. Different Types of MQCT Calculations

By default, the most general full-coupled version of MQCT calculations is carried out by the code, in which the transitions due to the Coriolis coupling are included. Such "coupled-channel" calculations are referred to as CC-MQCT. One important option, initiated by the keyword `CS_APPROX=YES`, is to run the so-called *coupled-states* calculations, CS-MQCT [Semenov 2014], where the Coriolis-driven transitions are neglected. Within each j channel, the Coriolis term couples $2j+1$ degenerate states labelled by m. In CS-MQCT, these Coriolis couplings and corresponding transitions are neglected, so, the calculations are done *independently* for various fixed values of m. Computational speed up by an order of magnitude is typical, due to simpler equations of motion and smaller number of coupled states.

New option in MQCT 2023 is adiabatic trajectory approximation, named AT-MQCT [Mandal 2020]. In this case, two runs of the code are required: During the first "adiabatic" run user can include only one state in the basis, the initial state, and indicate the keyword `SAVE_TRAJECT=YES`. The code propagates MQCT trajectories in this trivial basis (including the Coriolis driven transitions between m states of the initial channel) and saves all trajectory information to files. During the second run, with full basis set specified and with keyword `AT_APPROX=YES` indicated, the code propagates only the quantum equations of motion (to determine inelastic transition probabilities) reading pre-computed trajectory information from files. In this way, the coupling between classical and quantum parts of the system is neglected. Moreover, the evolution of potential energy along the trajectory recorded during the first run, can be used to automatically adjust the propagation time step for the second run, using a tolerance parameter, e.g., `AT_ADAPTOL=0.001` (with numerical value determined by convergence studies). Computational speed up by two orders of magnitude is typical (RK4 integrator is used in this case). One must also remember that in this version of the code, the CS-MQCT approximation cannot be used simultaneously with AT-MQCT approximation, so, only one of these keywords can be indicated at a time.

For AT-MQCT calculations the directory named `AT_APPROX_TRAJS` is created, with three more levels of sub-directories inside, that correspond to different collision energies and different degenerate states j and m. Information on each individual trajectory is saved to an unformatted file `AT_j_m_l_p.DAT`, where l is the orbital angular quantum number and p is exchange parity of two identical collision partners, zero or one (for non-identical partners this value is set to 0). If the Monte-Carlo sampling of initial conditions is used, the trajectories with different values of j and m are saved to the same directory, since these values are sampled randomly.

IV. Calculation of Potential Coupling Matrix for State-to-State Transitions

Within MQCT 2023 code there are two strategies/options for computing the potential coupling matrix, different by the method of PES representation and by data handling. Description of the format for these two options can be found in the file `user_suppl_pot.f` in the directory `PES_USER`, and the file `pes_sys_type.f` in the main code directory `MQCT_2023`

Option 1: is the default, equivalent to the keyword `EXPANSION=NO`. In this case the user should provide a subroutine `USER_DEFINED_PES` that generates the value of potential energy as a function of the molecule-molecule distance *R* and the internal coordinates (Euler angles in the body-fixed reference frame, and bond lengths). MQCT code will use this subroutine to compute elements of the state-to-state transition matrix directly, by numerical integration over the internal molecular degrees of freedom. Such calculations are done for every grid point of the molecule-molecule distance *R* and the data are stored in the memory. For calculations of the collision process, when the values of matrix elements and their derivatives (for classical equations of motion) are needed at certain values of *R* along the trajectory, one-dimensional cubic spline of each matrix element is computed.

Option 2: keywords `READ_EXPANSION=YES`. In this case, the user is required to supply a data file `PES_EXPAN_TERMS.DAT` that contains pre-computed coefficients for analytical expansion of the PES (listed in Table 8 for different system types) at every grid point of the molecule-molecule distance *R*. The format of the data file is described in the subroutine `EXPAND_PES` (see Sec. III). Using these coefficients, MQCT will calculate elements of the state-to-state transition matrix analytically (using equations of Table 9) at every grid point of the molecule-molecule distance *R*, store them in the memory and finally spline (during the dynamics calculations, just like in the Option 1). This is convenient when these data are already available, say from literature. Importantly, the user can pre-run the MQCT code with the optional keyword `CALC_EXPANSION=YES` to generate the file `PES_EXPAN_TERMS.DAT`, as shown in several example files provided with the code, and is discussed in Sec. V.

Both the direct calculation of transition matrix (default, Option 1) and the projection of PES onto the basis set of analytic functions (Option 2, keyword `CALC_EXPANSION=YES`) deal with numerical integration over the internal degrees of freedom using multi-dimensional numerical quadrature. For both cases the number of quadrature points should be indicated by the user in the input file using keywords `GRD_VIB`, `GRD_ANG1`, `GRD_ANG2`, and `GRD_ANG3`. Integration over vibrational coordinate uses an equidistant grid quadrature for automatically generated Morse oscillator functions, or optionally a user-specified non-equidistant grid with the weights indicated for each point in the file `USER_DEFINED_BASIS.DAT`. Gauss-Legendre quadrature is used for angle β only (`GRD_ANG2`). Integrals over α and γ are computed using an equidistant grid by default, but Gauss-Legendre quadrature can also be requested using a keyword `EQ_ANG_GRID=NO`. For all molecule + atom systems (`SYS_TYPE=1` to `4`) only one integer value should be assigned to each of `GRD_VIB`, `GRD_ANG1`, `GRD_ANG2`, and `GRD_ANG3`. But for all molecule + molecule systems (`SYS_TYPE=5` to `0`) two integer numbers should be given sequentially, separated by coma. Note, however, that depending on the system type, some of these numbers are dummy (not used) and can be arbitrary. Details are given below:

SYS_TYPE=1: for diatomic + atom integration is carried out along β-angle only, with the number of points GRD_ANG2. The values of GRD_ANG1 and GRD_ANG3 are dummy. Vibrational degree of freedom is dummy for all SYS_TYPE but 2 and 6.

SYS_TYPE=2: for vibrating diatomic + atom, besides β-angle described above, the integral includes GRD_VIB points for vibration. The values of GRD_ANG1 and GRD_ANG3 are dummy, just as in SYS_TYPE=1.

SYS_TYPE=3 and 4: for any top + atom the number of points for angles β and γ is indicated by GRD_ANG2 and GRD_ANG3, respectively. The value of GRD_ANG1 is dummy.

SYS_TYPE=5: for diatom + diatom the number of points along two β-angles is given by two entries of the keyword GRD_ANG2, while the number of points for α-angle is indicated by the *second* entry of the keyword GRD_ANG1. The first entry of the keyword GRD_ANG1, and both entries of the keyword GRD_ANG3 are dummy.

SYS_TYPE=6: for vibrating diatom + diatom the number of points along each bond length is indicated by two entries of the keyword GRD_VIB, in addition to the angular coordinates of SYS_TYPE=5.

SYS_TYPE=7 and 8: for any top + diatom the number of points along two β-angles is given by two entries of the keyword GRD_ANG2, the number of points along α is given by the second entry of GRD_ANG1 (the first entry is dummy), while the number of points along γ is given by the first entry of the keyword GRD_ANG3 (the second entry is dummy).

SYS_TYPE=9 and 0: for top + top systems the number of points along α is given by the second entry of GRD_ANG1 (only the first entry is dummy), the number of points along two β-angles is given by two entries of the keyword GRD_ANG2, the number of points along two γ-angles is given by two entries of the keyword GRD_ANG3.

V. Compiling and Running the Code, and Choosing Efficient Parallelization Strategies

Compilation of the PES function and of MQCT code is done by using a Makefile. Our version of Makefile is provided as an example in the main directory. Users are expected to change variables in the Makefile as needed for their computer architecture, directory structure and PES function. The example Makefile contains a brief description at the top of the file to describe how it should work. Users would make changes in the section that follows. First, one needs to provide a path to PES subroutines as variable named pes_dir. Next, in the variable named pes_files, one would write the name of the Fortran file (or files separated by a blank space) that contains the PES subroutine, with the extension *.o (i.e., all object files of the PES routine). After that, users should provide their choice of name for the executable file as variable exe_name. Finally, if the MQCT codes are not present in the same directory (as shown in the example Makefile), then one would have to provide a path to the MQCT code files as variable src_dir.

Next step in modifying the `Makefile` is to provide appropriate names for the compiler and linear algebra libraries. For that, the user should indicate the name of the Fortran compiler with MPI option in the variable `FC`. Then any flags for the Fortran compiler should be mentioned in the variable `FFLAGS`, such as optimization level, debugging or traceback. At the end, one should indicate the name of the linear algebra libraries in the variable `LDFLAGS`, such as `LAPACK, BLAS` and `MKL`.

After making and saving all changes in the `Makefile`, and before proceeding to the compilation stage, users should make sure that all the necessary modules for Fortran compiler and math libraries are loaded. It is also recommended that prior to compilation users clean their old files by using `$make clean` command. After that, simply use `$make` to compile everything. If the compilation is successful, the directory `./bin` will contain an executable file and it is recommended to use this file in further calculations.

In the example directory, our `Makefile` is prepared for $H_2O + H_2O$ system. The PES of the molecular system is located in the directory `./PES_DIRECTORIES/PES_H2OH2O` which is assigned to the variable `pes_dir`. For this system, all subroutines to evaluate the PES are combined in one FORTRAN file `PES_H2OH2O.f`. Therefore, the variable `pes_files` is assigned only one filename `PES_H2OH2O.o`. The name of the executable as indicated in the variable `exe_name` is `mqct_H2OH2O`. And finally, the `MQCT` codes are located in the directory named `MQCT_SOURCE_CODES` in the root, which is assigned to `src_dir`.

The input file for MQCT should have the extension `*.inp`, and its name should be placed in the file `INPUT_NAME.inp`. This permits the user to store multiple input files (*e.g.*, for different molecules) in the program directory, but run actual calculations with one specific input file.

There are two levels of parallelization in the code. At the first level, propagation of each trajectory can be done by multiple processors used as a group (*e.g.*, all processors of a node) to compute right-hand sides of the classical and quantum equations of motion. This requires some minimal message passing. At the second level, propagation of different trajectories can be assigned to different groups, which requires virtually no message passing. The program attempts to evenly split all requested trajectories between the groups. For example, if the code is submitted for execution using 60 nodes of the machine with 32 cores per node (1920 processors total) the following option, indicated in the `$SYSTEM` block:

```
MPI_PERTRAJ=32, NMB_TRAJ=300
```

will result in formation of 60 groups, and assignment of 5 trajectories per group. Typically, trajectories with larger impact parameters are shorter and faster to propagate. Thus, for an equalized load, and optimal use of resources, it is recommended to assign several trajectories per group, not just one. Note that `NMB_TRAJ` is an optional keyword used only in conjunction with Monte-Carlo sampling of the initial states (see Sec. VIII below). In regular (not a Monte-Carlo) calculations, the code automatically determines the required number of MQCT trajectories and attempts to split them evenly between the groups.

Several practical aspects of MQCT calculations should be outlined. In principle, the calculations of potential coupling matrix

and the subsequent calculations of collision trajectories can be done in a single run of MQCT, but this is practical for simple systems only, when the CPU effort and the memory requirement are relatively low. Namely, if the keyword PROG_RUN=YES is indicated, while the matrix is not yet available, the code will compute the matrix by direct integration (Option 1, the default) and will proceed to the propagation of trajectories, all in one run. Calculation of different matrix elements is split about equally between different processors and there is not any constraint on the number of processors at the matrix stage. When the code proceeds to the propagation of the equations of motion, it allocates a group of processors (MPI_PERTRAJ) for calculation of the right-hand-side of each equation of motion. Within a group, each processor holds in its memory only a part of the matrix, and the overall cost of trajectory propagation is split about equally. If the total number of processors in the run is N_PROC, the code will start propagating a set of N_PROC/MPI_PERTRAJ trajectories, which should be an integer. When a group of processors finishes one trajectory, it takes the next, and so on. Therefore, the number of processors is limited by N_PROC ≤ MPI_PERTRAJ*NMB_TRAJ. If the number of processors is larger than this, the code will stop with an error message. If SAVE_MTRX=YES is indicated, then, prior to the propagation of trajectories, the code will save truncated matrix to a single file, unformatted by default (MTRX_UF.DAT) or formatted (MTRX.DAT) if the keyword UNFORMAT=NO is specified. This procedure works but has a bottleneck: Since all pieces of the matrix computed by different processors are sent to one processor, a very large memory per processor is required (in order to store the entire matrix). Writing the entire matrix to one file by one processor is also slow and inefficient (since other processors are idling). Therefore, for calculations with a large matrix, a multistep procedure with parallel-IO is recommended, as explained below.

For complicated molecular systems with many states and large state-to-state transition matrix it is recommended that the matrix is pre-computed in a separate run of MQCT following one parallelization strategy, and only after that the trajectories are propagated using a different number of processors and different parallelization strategy. Namely, for the "matrix" run two keywords SAVE_MTRX=YES, PROG_RUN=NO should be indicated, and the number of processors should be as large as possible. Overhead of parallelization is negligible for matrix calculations and this approach permits to reduce the wall-clock time. The subsequent "propagation" run is initiated by two keywords: READ_MTRX=YES, PROG_RUN=YES. In order to reduce message passing between the nodes, one may want to fit a whole number of processor groups in each node (for example, ten groups per node), by choosing an appropriate group size MPI_PERTRAJ. However, for very large systems the memory requirement may become more important. Remember that each group of processors will have to hold the entire matrix in the memory. Therefore, expanding the group onto several nodes may be unavoidable for machine architectures with small memory.

While the matrix calculations *per se* do not require large memory, the matrix collection and redistribution by one processor, as well as saving to (and reading from) single file represent a bottleneck. To avoid this problem, users can indicate the keyword PARALLEL_IO=YES during both "matrix" and "propagation" runs. At the "matrix" stage, this option tells individual processors to save their pieces of computed (and truncated) matrix into individual output files without any message passing. The number of files

will be equal to the number of processors used for this run. During the subsequent "propagation" run, with PARALLEL_IO=YES indicated, the elements of truncated matrix will be distributed evenly between processors of each group, but each processor will read and use only its assigned chunk of the matrix. This procedure avoids reading the entire matrix by one processor or holding the entire matrix in the memory of one processor.

While two separate runs are optional when the matrix is calculated by direct integration (Option 1), they are mandatory in the case when the PES expansion is computed (prior to using Option 2). Namely, if Option 2 is used, then for the first run, initiated by the keyword CALC_EXPANSION=YES to generate the PES expansion coefficients, the number of processors must be equal to the number of requested expansion terms, since each processor will be responsible for computing one expansion coefficient. The code will compute the expansion coefficients and will stop, without proceeding to the calculations of matrix elements or trajectories. Then, user should replace CALC_EXPANSION in the input file by READ_EXPANSION=YES and run the code again with the number of processors appropriate for calculations of matrix elements (keywords SAVE_MTRX=YES, PROG_RUN=NO). Finally, the code should be run a third time for the propagation of trajectories (keywords READ_MTRX=YES, PROG_RUN=YES), with an appropriate value of MPI_PERTRAJ set up. The keyword PARALLEL_IO=YES can be used during the "matrix" and "trajectories" calculations, as explained above.

In this procedure, one should be careful about the units of distance and energy. The file of expansion terms PES_EXPAN_TERMS.DAT, printed by the code, will always contain distance in Bohr and energy in the same units that are used by the potential energy surface routine (employed for the calculations of expansion). If the expansion is used further (to compute the state-to-state transition matrix and/or the collision dynamics), the units of distance should be set as Bohr and the units of energy should stay the same as in the potential energy surface routine. However, the matrix elements computed by the code will always be saved in atomic units (Bohr for distance and Hartree for energy).

If the angular grid is very large, and, in particular, when the number of *R*-grid points is large, it may be convenient to split calculations of the PES expansion into several runs. To do that, user can specify the range of *R*-grid to cover in one run, using keywords IR_BGN and IR_FIN. Results of successive runs are combined automatically into a single file for the expansion terms and coefficients.

When computing the state-to-state transition matrix by direct integration (Option 1), or computing the PES expansion by projection (Option 2), it may be advantageous, in terms of CPU time, to keep in the memory the values of potential at the grid points, rather than calling the PES subroutine each time when the value for new point is needed. However, a very large grid (for larger molecules and complicated PES) may not fit as a whole into the memory of one CPU. For this case the option VGRID_FILE=YES is recommended. The code first generates the PES at the points of the grid and saves these data to the unformatted file VGRID_UF.DAT. The number of processors should be at least equal to the number of points of *R*-grid, or larger. Then the code loads this information

into the memory of processors to compute matrix elements (or the expansion coefficients) but does it by slices, sequentially for each value of *R* on the grid, since the calculation at each value of *R* is independent. Different processors will be responsible for computing different elements of the matrix, or different expansion terms. Note that if the expansion is computed, the code will save the data file `VGRID_UF.DAT` and will normally stop (except a rare special case when the number of *R*-grid points is equal to the number of the PES expansion terms). It should be run again with the number of CPUs equal to the number of the expansion terms. The code will read the data file `VGRID_UF.DAT` and proceed to the calculations of the expansion coefficients, one term per processor. Calculations of matrix elements and the propagation of trajectories should be done in the following independent runs, as explained above.

State-to-state transition matrix elements computed and used by the code are labeled by one index that runs through all matrix elements (all transitions). Information about two states coupled by given matrix element is contained in a separate array and is stored in the same matrix file. If the keyword `PARALLEL_IO=YES` is used and the matrix is saved to multiple files, the range of matrix elements saved to each file is embedded in the file name using the indexes of first and last matrix elements in this file, `Mij_xxx_yyy.DAT`. Inside the file, we save sequentially the index of transition, the value or *R*-grid, and the value of matrix element in Hartree, running the inner cycle over *R*-grid and the outer cycle over the index of transition. Information about two states coupled by given matrix element is contained in a separate index file, `Ind_xxx_yyy.DAT`. Directory `MATRIX_TRUNCATED` holds matrix files created by different processors. In order to take full advantage of symmetry (when the symmetry is not obvious) the code automatically disregards transitions described by matrix elements with absolute values smaller than `MIJ_CUTOFF=1.d-12` (in the units of cm^{-1}). These matrix elements are neglected, and the corresponding transitions are removed from the list. If needed, an alternative value of the cut-off can be specified, for example, to make calculations more affordable by disregarding transitions between the weakly coupled states. This truncation scheme represents a simple but very efficient approximation. A suitable value of `MIJ_CUTOFF` should be determined by convergence studies. Two values of *R* distance (where the matrix elements are analyzed for this truncation) can be indicated using keywords `CUTOFF_R1` and `CUTOFF_R2` (the default values are 6 and 8 Bohr). For each transition, the code first computes the values of matrix element at two grid points closest to `CUTOFF_R1` and `CUTOFF_R2`. If at least one of them is larger than `MIJ_CUTOFF`, the code computes this matrix element through the rest of *R*-grid. If both are smaller, the corresponding matrix element is not computed, and this transition is disregarded to create a truncated (reduced) matrix. One can easily re-truncate the existing matrix, by moving it to the directory `MATRIX_FILES`, indicating the value of desired `MIJ_CUTOFF` and running the code one more time (with the same number of processors that was used for calculations of the original matrix) using keywords `READ_MTRX=YES, RETRUNCATE=YES, PROG_RUN=NO`. New directory `MATRIX_TRUNCATED` will be created with new files that will hold the re-truncated matrix. The new value of `MIJ_CUTOFF` should be larger than the old one. If one wants a matrix with smaller cut-off value (a larger matrix), it has to be recomputed from scratch.

In complicated molecular systems the computation of matrix elements for some transitions (e.g., with large values of j_1 and j_2 of the initial and final states) may require much more numerical effort than that for other matrix elements (with small values of j_1 and j_2). This is particularly so for SYS_TYPE = 5,6,7,8,9 and 0, characterized by large values of λ_1, λ_2 and λ in the PES expansion, since in this case the multiple summations required for computation of matrix elements (see Table 8 below) become numerically demanding. In this case some matrix elements will be computed fast, and the corresponding processors will idle, while some other processors will continue running. Moreover, many matrix elements will be disregarded based on the value of MIJ_CUTOFF, and the corresponding processors will not do any useful work. To overcome this imbalance, we developed a three-step procedure that permits users to equalize the load between all processors at the stage of matrix computation. Note that it should be combined with parallel output. In the first preliminary step, initiated by keywords PARALLEL_IO = YES, REBALANCE = YES, the matrix elements are computed for two values of the molecule-molecule distance R given by CUTOFF_R1 and CUTOFF_R2. During this step, a counter variable is set to determine the numerical effort needed for each matrix element. As usual, MQCT code compares the values of computed matrix elements with given cut-off value MIJ_CUTOFF to decide whether the transition is strong enough and must be retained, or it can be disregarded. Based on this run, the code will create a directory named REBALANCE_FILES which contains information that will be used during the next step to distribute the load equally among processors. In the next intermediate step, the user needs to copy a utility program analysis_load.f90 from the directory TOOLBOX_FILES into the directory REBALANCE_FILES, where it should be compiled and run (sequentially). The code will ask to enter the number of processors to be used during the actual calculation of matrix elements and will store information in new files created in the same directory. Finally, to compute matrix elements for all values of R-grid using information generated in the previous steps, user should indicate PARALLEL_IO = YES, BALANCED_MIJ = YES. The number of processors should be the same as indicated during the run of analysis_load.f90 (which can be different from the first run). The same R-grid must be used during the entire procedure.

VI. PES Interface and Other User-Supplied Subroutines and Data Files

Formally, the potential energy surface subroutine USER_DEFINED_PES operates with the same coordinates for all system types, but some of these coordinates are dummy variables. Namely, the input for PES subroutine requires, besides the molecule-molecule distance R, one vibrational coordinate and three Euler angles for each collision partner (see the file user_suppl_pot.f in the directory PES_USER, or the file pes_sys_type.f in the code directory MQCT_2023). However, the vibrational coordinate is used only for SYS_TYPE=2,6 where the vibrational motion of the diatomic is explicitly treated, while it is a dummy variable for all other system types. In the future, our plan is to add one vibrational degree of freedom (such as bending motion in triatomic molecules) for other system types, but this is not yet implemented in the present release of the code. The subroutine USER_DEFINED_PES can also be used as an "layer" between the external PES subroutine and MQCT, for example, to convert units

of energy not supported by MQCT (such as eV), to convert degrees into radians, or to transform spherical angles into Euler angles. Note that a subroutine `MFTOBF_CONV` (in the file `user_suppl_pot.f`) is available for conversion of the PES from a popular space-fixed reference frame to the body-fixed reference frame used by MQCT for `SYS_TYPE=7,8`.

Concerning the angular coordinates, some of them are dummy as described in Sec. II above, and some of them are arbitrary (fixed at zero value here). For completeness, we summarized these properties in Table 6, where each dummy variable is indicated by a dash. We want to emphasize one more time that our reference frame, called the body-fixed reference frame tied to the molecule-molecule vector, is different from coordinates used in some other codes (such as MOLSCAT) where the reference frame is tied to one of the molecules. The words "azimuthal" and "polar" are given in the cases of diatomics, to underline the meaning of those rotations.

Table 6: Degrees of freedom in the user-supplied PES subroutine

SYS_TYPE	R	r_1	r_2	α_1	β_1	γ_1	α_2	β_2	γ_2
1	distance	--	--	--	Polar	zero	--	--	--
2	distance	vibration	--	--	Polar	zero	--	--	--
3	distance	--	--	zero	Euler	Euler	--	--	--
4	distance	--	--	zero	Euler	Euler	--	--	--
5	distance	--	--	--	Polar	zero	Azimuthal	polar	--
6	distance	vibration	vibration	--	Polar	zero	Azimuthal	polar	--
7	distance	--	--	zero	Euler	Euler	Azimuthal	polar	--
8	distance	--	--	zero	Euler	Euler	Azimuthal	polar	--
9	distance	--	--	zero	Euler	Euler	Euler	Euler	Euler
0	distance	--	--	zero	Euler	Euler	Euler	Euler	Euler

For several special cases listed below, some of the input data must be generated externally by users and supplied in separate files. The following table gives a brief description of these data files (extension `*.DAT`). Examples of these files can be found in the directory `EXAMPLES_IO`. In order to create properly formatted files, users can employ our subroutines supplied with the code, all located in the `TOOLBOX_CODES`. They are listed in the table below. Note that these are not ready-to-use utility programs to generate these data, but merely the examples of data formats required by MQCT code.

Table 7: Description of the input data files for MQCT calculations

Data File Name	Generating Subroutine	Brief Description
PES_EXPAN_TERMS.DAT	EXPAND_PES	Contains *R*-dependence of the expansion coefficients for analytic representation of the PES (as in Table 8 below). Works for all SYS_TYPE, except 2 and 6 where the vibrational motion is involved.
USER_DEFINED_BASIS.DAT	DEFINE_BASIS See example file	Contains channel labels (quantum numbers), energies, and wave functions of the externally computed ro-vibrational states. For SYS_TYPE=2 and 6 the vibrational wave functions should be pre-computed on a grid, with indicated grid points and weights. For SYS_TYPE=4,8,9,0 the rotational states (*e.g.*, of Kyro Hamiltonian) should be represented by expansion over the basis set of symmetric-top eigenstates. Depending on system type, this includes one or both collision partners.
USER_DEFINED_RGRID.DAT	See example file	Contains user defined (*e.g.*, non-equidistant) grid for the molecule-molecule distance *R*. Can be useful for large amplitude vibrational motion states. Must be in the units of Bohr.
STATES_TO_EXCLUDE.DAT	See example file	Optional. Undesired states can be excluded by listing in this file the state numbers (as they appear in the file STATES.out), and setting the keyword EXCLUDE_STATES=YES.

Note that the default method of describing ro-vibrational states is expected to be reasonably accurate only for low to moderate rotational and vibrational excitations, since it neglects ro-vibrational interaction. A more reliable approach, accurate up to dissociation limit, is to compute numerically accurate ro-vibrational states using an external code (not provided) and feed them as input for the MQCT code for one (SYS_TYPE=2) or both (SYS_TYPE=6) collision partners. This is achieved by the keyword LEVELS_FILE=YES. If specified, the file named USER_DEFINED_BASIS.DAT is also required (copied or linked to the code directory), that should contain energies and wave functions of pre-computed states. The number of states in the files should be equal to NMB_CHNLS (indicated in the block $BASIS). The values and weights of grid points for numerical integration (can be non-equidistant) should also be specified. The number of points should be equal to GRD_VIB (indicated in the block $POTENTIAL). Examples of such files are distributed with the code.

VII. Expansion of PES Over the Basis Set of Analytic Functions

In MQCT the PES can be expanded over the basis set of analytic functions specific to the system type (by indicating keyword `CALC_EXPANSION=YES`). Those are listed in Table 8, along with the labels of such expansion terms. Note that all our expansions use three angles of active Euler rotations for orientation of each molecule relative to the molecule-molecule axis *z*, which is called body-fixed reference frame (which is different from a popular set-up where one of the molecules is fixed in the lab, while the other is moved relative to it and is rotated relative to the lab axes). To define the expansion terms by the keyword `TERMS`, user will specify a set of "quantum numbers" for each term, separated by coma. The order of indicated expansion terms defines the order in which they will be handled (computed by subroutines `USER_DEFINED_TERMS` or `CALC_EXPANSION=YES`, written and read from file `PES_EXPAN_TERMS.DAT`, summed into the matrix element, *etc*.). The total number of terms to be employed must be specified by the keyword `NMB_TERMS`. If user does not wish to list all terms, the code can automatically assign them based on the keywords `L1_MAX, M1_MAX, L2_MAX, M2_MAX` and `L_MAX`, as appropriate to each system type (see Table 8). In this case, the code will generate all the terms and save into a file named `EXPANSION_TERMS_LIST.DAT` and will stop. The user will then need to incorporate these terms within the input (copy/paste). One can add/eliminate some terms by hand at this point if one wants to do that.

The ranges of "quantum numbers" for expansion terms are:

$$\lambda_1, \lambda_2, \lambda \geq 0; \quad |\lambda_1 - \lambda_2| \leq \lambda \leq \lambda_1 + \lambda_2; \quad 0 \leq \mu_1 \leq \lambda_1 \text{ (positive only)}; \quad -\lambda_2 \leq \mu_2 \leq \lambda_2 \text{ (positive if } \mu_1 = 0).$$

Note that in the expressions of Table 8 two terms in each square bracket account for the invariance of potential with respect to reflection through origin, using phase factor $(-1)^{\lambda_1+\mu_1+\lambda_2+\mu_2+\lambda}$. Moreover, for the PESs of two identical molecules (`SYS_TYPE=0,5`), indicated by optional keyword `IDENTICAL_PES=YES`, the expansion functions are further symmetrized with respect to exchange of two molecules, by combining the expansion terms as indicated in Table 8 with phase factor $(-1)^{\lambda_1+\lambda_2}$, and, simultaneously the number of terms is reduced by two additional conditions: $\lambda_2 \leq \lambda_1$, and, if $\lambda_2 = \lambda_1$ then $|\mu_2| \leq \mu_1$.

Also note that for linear molecules only the terms with even values of $\lambda_1 + \lambda_2 + \lambda$ are required. In the case of non-liner molecules, we found that the terms with odd values of $\lambda_1 + \lambda_2 + \lambda$ are small too, and therefore can often be neglected (although they are not strictly zero and may play role in some cases). The keyword `ODD_L1L2L=YES` may be used to generate the terms with odd values of $\lambda_1 + \lambda_2 + \lambda$.

The expansion functions τ in Table 8 are mutually orthogonal, are complex in general, but are normalized only in those cases when the square brackets are not present (although, each individual term in the square bracket is also normalized). In the code, for simplicity and efficiency, the expansion coefficients are obtained by projection of potential only onto the first term in each square bracket: $v(R) = \langle \text{first term of } \tau(\Lambda)|V(R,\Lambda)\rangle$, and then this $v(R)$ is used for every term in the bracket). These expansion coefficients are always real. They are used to construct elements of state-to-state transition matrix using analytical expressions summarized in Table 9**.**

Table 8: PES expansion functions in MQCT calculations

System Type	System Description	Labels of Expansion Terms	Expansion Functions
1	rigid diatom + atom	λ	$\tau_{\lambda}(\beta) = \mathrm{Y}_0^{\lambda}(\beta, 0)$
2	vibr.diatom + atom		
3	sym. top + atom	λ, μ	$\tau_{\lambda\mu}(\beta,\gamma) = \left[(-1)^{\mu}\mathrm{Y}_{+\mu}^{\lambda}(\beta,\gamma) \; + \; \mathrm{Y}_{-\mu}^{\lambda}(\beta,\gamma)\right]$
4	asym. top + atom		
5	rigid diatom + rigid diatom	$\lambda_1, \lambda_2, \lambda$	$\tau_{\lambda_1\lambda_2\lambda}(\beta_1,\alpha_2,\beta_2) = \sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)} \mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta}\mathrm{Y}^{\lambda_1}_{+\eta}(\beta_1,0)\mathrm{Y}^{\lambda_2}_{-\eta}(\beta_2,\alpha_2)$ If `IDENTICAL_PES=YES`, combine with: $+\mathrm{C}^{\lambda,0}_{\lambda_2,\eta,\lambda_1,-\eta}\mathrm{Y}^{\lambda_2}_{+\eta}(\beta_1,0)\mathrm{Y}^{\lambda_1}_{-\eta}(\beta_2,\alpha_2)$
6	vibr. diatom + vibr. diatom		
7	sym. top + rigid diatom	$\lambda_1, \mu_1, \lambda_2, \lambda$	$\tau_{\lambda_1\mu_1\lambda_2\lambda}(\beta_1,\gamma_1,\alpha_2,\beta_2) = \sqrt{\frac{2\lambda_1+1}{4\pi}} \sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)} \mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta} \times \left[\mathrm{D}^{\lambda_1*}_{\eta,+\mu_1}(0,\beta_1,\gamma_1) + (-1)^{\lambda_1+\mu_1+\lambda_2+\lambda}\,\mathrm{D}^{\lambda_1*}_{\eta,-\mu_1}(0,\beta_1,\gamma_1)\right] \times \mathrm{Y}^{\lambda_2}_{-\eta}(\beta_2,\alpha_2)$
8	asym. top + rigid diatom		
9	sym. top + asym. top	$\lambda_1, \mu_1\lambda_2, \mu_2, \lambda$	$\tau_{\lambda_1\mu_1\lambda_2\mu_2\lambda}(\beta_1,\gamma_1,\alpha_2,\beta_2,\gamma_2) = \sqrt{\frac{2\lambda_1+1}{8\pi^2}}\sqrt{\frac{2\lambda_2+1}{8\pi^2}} \sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)} \mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta}\left[\mathrm{D}^{\lambda_1*}_{+\eta,+\mu_1}(0,\beta_1,\gamma_1)\mathrm{D}^{\lambda_2*}_{-\eta,+\mu_2}(\alpha_2,\beta_2,\gamma_2) + (-1)^{\lambda_1+\mu_1+\lambda_2+\mu_2+\lambda}\,\mathrm{D}^{\lambda_1*}_{+\eta,-\mu_1}(0,\beta_1,\gamma_1)\mathrm{D}^{\lambda_2*}_{-\eta,-\mu_2}(\alpha_2,\beta_2,\gamma_2)\right]$ If `IDENTICAL_PES=YES`, combine with: $+\mathrm{C}^{\lambda,0}_{\lambda_2,\eta,\lambda_1,-\eta}\left[(-1)^{\lambda_1+\lambda_2}\,\mathrm{D}^{\lambda_2*}_{+\eta,+\mu_2}(0,\beta_1,\gamma_1)\mathrm{D}^{\lambda_1*}_{-\eta,+\mu_1}(\alpha_2,\beta_2,\gamma_2) + (-1)^{\mu_1+\mu_2+\lambda}\,\mathrm{D}^{\lambda_2*}_{+\eta,-\mu_2}(0,\beta_1,\gamma_1)\mathrm{D}^{\lambda_1*}_{-\eta,-\mu_1}(\alpha_2,\beta_2,\gamma_2)\right]$
0	asym. top + asym. top		

Note: The relationship $(-1)^m\mathrm{Y}^j_m = \sqrt{(2j+1)/(4\pi)}\,\mathrm{D}^{j*}_{0m}$ is used for system types 3 and 4. The relationship $\mathrm{Y}^j_m = \sqrt{(2j+1)/(4\pi)}\,\mathrm{D}^{j*}_{m0}$ is used for system types 5 to 8.

Table 9: Computation of state-to-state transition matrix elements using PES expansion

System Type	System Description	State-to-state transition matrix equations
1	rigid diatom + atom	$\langle j''m\|V(\Lambda)\|j'm\rangle = \sqrt{\frac{2j'+1}{2j''+1}}\sum_{\lambda} v_{\lambda}(R)\sqrt{\frac{2\lambda+1}{4\pi}}\mathrm{C}^{j'',m}_{j',m,\lambda,0}\mathrm{C}^{j'',0}_{j',0,\lambda,0}$
2	vibr. diatom + atom	$\langle j''mv''\|V(\Lambda)\|j'mv'\rangle = \sqrt{\frac{2j_1'+1}{2j_1''+1}}\sum_{i} w_i\psi_{v_1'}(r_i)\psi_{v_1''}(r_i)\sum_{\lambda} v^{i}_{\lambda}(R)\sqrt{\frac{2\lambda+1}{4\pi}}\mathrm{C}^{j'',m}_{j',m,\lambda,0}\mathrm{C}^{j'',0}_{j',0,\lambda,0}$
3	sym. top + atom	$\langle j''mk''\varepsilon''\|V(\Lambda)\|j'mk'\varepsilon'\rangle = \sqrt{\frac{2j'+1}{2j''+1}}\frac{1}{\sqrt{2(1+\delta_{k',0})2(1+\delta_{k'',0})}}\sum_{\lambda\mu} v_{\lambda\mu}(R)\sqrt{\frac{2\lambda+1}{4\pi}}\mathrm{C}^{j'',m}_{j',m,\lambda,0}$ $\left[\left(\mathrm{C}^{j'',k''}_{j',k',\lambda,\mu}+\varepsilon'\mathrm{C}^{j'',k''}_{j',-k',\lambda,\mu}+\varepsilon''\mathrm{C}^{j'',-k''}_{j',k',\lambda,\mu}+\varepsilon'\varepsilon''\mathrm{C}^{j'',-k''}_{j',-k',\lambda,\mu}\right)+(-1)^{\mu}\left(\mathrm{C}^{j'',k''}_{j',k',\lambda,-\mu}+\varepsilon'\mathrm{C}^{j'',k''}_{j',-k',\lambda,-\mu}+\varepsilon''\mathrm{C}^{j'',-k''}_{j',k',\lambda,-\mu}+\varepsilon'\varepsilon''\mathrm{C}^{j'',-k''}_{j',-k',\lambda,-\mu}\right)\right]$
4	asym. top + atom	$\langle j''m\|V(\Lambda)\|j'm\rangle = \sqrt{\frac{2j'+1}{2j''+1}}\sum_{\lambda\mu} v_{\lambda\mu}(R)\sqrt{\frac{2\lambda+1}{4\pi}}\mathrm{C}^{j'',m}_{j',m,\lambda,0}\sum_{k''=-j''}^{+j''}\sum_{k'=-j'}^{+j'} b_{k''}b_{k'}\left(\mathrm{C}^{j'',k''}_{j',k',\lambda,\mu}+(-1)^{\mu}\mathrm{C}^{j'',k''}_{j',k',\lambda,-\mu}\right)$
5	rigid diatom + rigid diatom	$\langle j''mj_1''j_2''\|V(\Lambda)\|j'mj_1'j_2'\rangle = \sqrt{\frac{2j_1''+1}{2j_1'+1}}\sqrt{\frac{2j_2''+1}{2j_2'+1}}\sum_{\lambda_1\lambda_2\lambda} v_{\lambda_1\lambda_2\lambda}(R)\sqrt{\frac{2\lambda_1+1}{4\pi}}\sqrt{\frac{2\lambda_2+1}{4\pi}}\sum_{m_1=-j_1'}^{+j_1'}\mathrm{C}^{j',m}_{j_1',m_1,j_2',m-m_1}\sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)}\mathrm{C}^{j'',m}_{j_1'',m_1-\eta,j_2'',m-(m_1-\eta)}$ $\times\mathrm{C}^{j_1',0}_{j_1'',0,\lambda_1,0}\mathrm{C}^{j_2',0}_{j_2'',0,\lambda_2,0}\mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta}\mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_1,\eta}\mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_2,-\eta}$ If `IDENTICAL_PES=YES`, combine with: $+\mathrm{C}^{j_1',0}_{j_1'',0,\lambda_2,0}\mathrm{C}^{j_2',0}_{j_2'',0,\lambda_1,0}\mathrm{C}^{\lambda,0}_{\lambda_2,\eta,\lambda_1,-\eta}\mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_2,\eta}\mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_1,-\eta}$
6	vibr. diatom + vibr. diatom	$\langle j''mj_1''j_2''v_1''v_2''\|V(\Lambda)\|j'mj_1'j_2'v_1'v_2'\rangle = \sqrt{\frac{2j_1''+1}{2j_1'+1}}\sqrt{\frac{2j_2''+1}{2j_2'+1}}\sum_{i_1} w_{i_1}\psi_{v_1'}(r_{i_1})\psi_{v_1''}(r_{i_1})\sum_{i_2} w_{i_2}\psi_{v_2'}(r_{i_2})\psi_{v_2''}(r_{i_2})\sum_{\lambda_1\lambda_2\lambda} v^{i_1i_2}_{\lambda_1\lambda_2\lambda}(R)\sqrt{\frac{2\lambda_1+1}{4\pi}}\sqrt{\frac{2\lambda_2+1}{4\pi}}$ $\sum_{m_1=-j_1'}^{+j_1'}\mathrm{C}^{j',m}_{j_1',m_1,j_2',m-m_1}\sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)}\mathrm{C}^{j'',m}_{j_1'',m_1-\eta,j_2'',m-(m_1-\eta)}\times\mathrm{C}^{j_1',0}_{j_1'',0,\lambda_1,0}\mathrm{C}^{j_2',0}_{j_2'',0,\lambda_2,0}\mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta}\mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_1,\eta}\mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_2,-\eta}$ If `IDENTICAL_PES=YES`, combine with: $+\mathrm{C}^{j_1',0}_{j_1'',0,\lambda_2,0}\mathrm{C}^{j_2',0}_{j_2'',0,\lambda_1,0}\mathrm{C}^{\lambda,0}_{\lambda_2,\eta,\lambda_1,-\eta}\mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_2,\eta}\mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_1,-\eta}$

7	sym. top + rigid diatom	$\langle j'' m j_1'' k'' \varepsilon'' j_2'' \vert V(\Lambda) \vert j' m j_1' k' \varepsilon' j_2' \rangle = \sqrt{\frac{2j_1''+1}{2j_1'+1}} \sqrt{\frac{2j_2''+1}{2j_2'+1}} \frac{1}{\sqrt{2\left(1+\delta_{k',0}\right)2\left(1+\delta_{k'',0}\right)}} \sum_{\lambda_1 \mu_1 \lambda_2 \lambda} v_{\lambda_1 \mu_1 \lambda_2 \lambda}(R) \sqrt{\frac{2\lambda_1+1}{4\pi}} \sqrt{\frac{2\lambda_2+1}{4\pi}} \mathrm{C}^{j_2',0}_{j_2'',0,\lambda_2,0}$ $\sum_{m_1=-j_1'}^{+j_1'} \mathrm{C}^{j',m}_{j_1',m_1,j_2',m-m_1} \sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)} \mathrm{C}^{j'',m}_{j_1'',m_1-\eta,j_2'',m-(m_1-\eta)} \mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta} \mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_1,\eta} \mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_2,-\eta}$ $\left[\left(\mathrm{C}^{j_1',k'}_{j_1'',k'',\lambda_1,\mu_1} + \varepsilon' \mathrm{C}^{j_1',-k'}_{j_1'',k'',\lambda_1,\mu_1} + \varepsilon'' \mathrm{C}^{j_1',k'}_{j_1'',-k'',\lambda_1,\mu_1} + \varepsilon'\varepsilon'' \mathrm{C}^{j_1',-k'}_{j_1'',-k'',\lambda_1,\mu_1}\right)\right.$ $\left. + (-1)^{\lambda_1+\mu_1+\lambda_2+\lambda} \left(\mathrm{C}^{j_1',k'}_{j_1'',k'',\lambda_1,-\mu_1} + \varepsilon' \mathrm{C}^{j_1',-k'}_{j_1'',k'',\lambda_1,-\mu_1} + \varepsilon'' \mathrm{C}^{j_1',k'}_{j_1'',-k'',\lambda_1,-\mu_1} + \varepsilon'\varepsilon'' \mathrm{C}^{j_1',-k'}_{j_1'',-k'',\lambda_1,-\mu_1}\right)\right]$
8	asym. top + rigid diatom	$\langle j'' m j_1'' j_2'' \vert V(\Lambda) \vert j' m j_1' j_2' \rangle = \sqrt{\frac{2j_1''+1}{2j_1'+1}} \sqrt{\frac{2j_2''+1}{2j_2'+1}} \sum_{\lambda_1 \mu_1 \lambda_2 \lambda} v_{\lambda_1 \mu_1 \lambda_2 \lambda}(R) \sqrt{\frac{2\lambda_1+1}{4\pi}} \sqrt{\frac{2\lambda_2+1}{4\pi}} \mathrm{C}^{j_2',0}_{j_2'',0,\lambda_2,0}$ $\sum_{m_1=-j_1'}^{+j_1'} \mathrm{C}^{j',m}_{j_1',m_1,j_2',m-m_1} \sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)} \mathrm{C}^{j'',m}_{j_1'',m_1-\eta,j_2'',m-(m_1-\eta)} \mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta} \mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_1,\eta} \mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_2,-\eta}$ $\sum_{k_1''=-j_1''}^{+j_1''} \sum_{k_1'=-j_1'}^{+j_1'} b_{k_1''} b_{k_1'} \left[\mathrm{C}^{j_1',k_1'}_{j_1'',k_1'',\lambda_1,\mu_1} + (-1)^{\lambda_1+\mu_1+\lambda_2+\lambda} \mathrm{C}^{j_1',k_1'}_{j_1'',k_1'',\lambda_1,-\mu_1}\right]$
9	sym. top + asym. top	$\langle j'' m j_1'' j_2'' k_2'' \varepsilon'' \vert V(\Lambda) \vert j' m j_1' j_2' k_2' \varepsilon' \rangle = \sqrt{\frac{2j_1''+1}{2j_1'+1}} \sqrt{\frac{2j_2''+1}{2j_2'+1}} \sqrt{\frac{1}{2\left(1+\delta_{k_2',0}\right)2\left(1+\delta_{k_2'',0}\right)}} \sum_{\lambda_1 \mu_1 \lambda_2 \mu_2 \lambda} v_{\lambda_1 \mu_1 \lambda_2 \mu_2 \lambda}(R) \sqrt{\frac{2\lambda_1+1}{8\pi^2}} \sqrt{\frac{2\lambda_2+1}{8\pi^2}}$ $\sum_{m_1=-j_1'}^{+j_1'} \mathrm{C}^{j',m}_{j_1',m_1,j_2',m-m_1} \sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)} \mathrm{C}^{j'',m}_{j_1'',m_1-\eta,j_2'',m-(m_1-\eta)} \mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta} \mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_1,\eta} \mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_2,-\eta}$ $\sum_{k_1''=-j_1''}^{+j_1''} \sum_{k_1'=-j_1'}^{+j_1'} b_{k_1''} b_{k_1'} \left[\left(\mathrm{C}^{j_1',k_1'}_{j_1'',k_1'',\lambda_1,\mu_1} \left(\mathrm{C}^{j_2',k_2'}_{j_2'',k_2'',\lambda_2,\mu_2} + \varepsilon' \mathrm{C}^{j_2',-k_2'}_{j_2'',k_2'',\lambda_2,\mu_2} + \varepsilon'' \mathrm{C}^{j_2',k_2'}_{j_2'',-k_2'',\lambda_2,\mu_2} + \varepsilon'\varepsilon'' \mathrm{C}^{j_2',-k_2'}_{j_2'',-k_2'',\lambda_2,\mu_2}\right)\right)\right.$ $\left. + (-1)^{\lambda_1+\mu_1+\lambda_2+\mu_2+\lambda} \left(\mathrm{C}^{j_1',k_1'}_{j_1'',k_1'',\lambda_1,-\mu_1} \left(\mathrm{C}^{j_2',k_2'}_{j_2'',k_2'',\lambda_2,-\mu_2} + \varepsilon' \mathrm{C}^{j_2',-k_2'}_{j_2'',k_2'',\lambda_2,-\mu_2} + \varepsilon'' \mathrm{C}^{j_2',k_2'}_{j_2'',-k_2'',\lambda_2,-\mu_2} + \varepsilon'\varepsilon'' \mathrm{C}^{j_2',-k_2'}_{j_2'',-k_2'',\lambda_2,-\mu_2}\right)\right)\right]$

0	asym. top + asym. top	$\langle j'' m j_1'' j_2'' \vert V(\Lambda) \vert j' m j_1' j_2' \rangle = \sqrt{\frac{2j_1''+1}{2j_1'+1}} \sqrt{\frac{2j_2''+1}{2j_2'+1}} \sum_{\lambda_1 \mu_1 \lambda_2 \mu_2 \lambda} v_{\lambda_1 \mu_1 \lambda_2 \mu_2 \lambda}(R) \sqrt{\frac{2\lambda_1+1}{8\pi^2}} \sqrt{\frac{2\lambda_2+1}{8\pi^2}}$ $\sum_{m_1=-j_1'}^{+j_1'} \mathrm{C}^{j',m}_{j_1',m_1,j_2',m-m_1} \sum_{\eta=-\min(\lambda_1,\lambda_2)}^{+\min(\lambda_1,\lambda_2)} \mathrm{C}^{j'',m}_{j_1'',m_1-\eta,j_2'',m-(m_1-\eta)}$ $\times \mathrm{C}^{\lambda,0}_{\lambda_1,\eta,\lambda_2,-\eta} \mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_1,\eta} \mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_2,-\eta} \Bigg[\sum_{k_1'=-j_1'}^{+j_1'} b_{k_1'} b_{k_1'-\mu_1} \sum_{k_2'=-j_2'}^{+j_2'} b_{k_2'} b_{k_2'-\mu_2} \mathrm{C}^{j_1',k_1'}_{j_1'',k_1'-\mu_1,\lambda_1,\mu_1} \mathrm{C}^{j_2',k_2'}_{j_2'',k_2'-\mu_2,\lambda_2,\mu_2}$ $+(-1)^{\lambda_1+\mu_1+\lambda_2+\mu_2+\lambda} \sum_{k_1'=-j_1'}^{+j_1'} b_{k_1'} b_{k_1'+\mu_1} \sum_{k_2'=-j_2'}^{+j_2'} b_{k_2'} b_{k_2'+\mu_2} \mathrm{C}^{j_1',k_1'}_{j_1'',k_1'+\mu_1,\lambda_1,-\mu_1} \mathrm{C}^{j_2',k_2'}_{j_2'',k_2'+\mu_2,\lambda_2,-\mu_2} \Bigg]$ If `IDENTICAL_PES=YES`, combine with: $+\mathrm{C}^{\lambda,0}_{\lambda_2,\eta,\lambda_1,-\eta} \mathrm{C}^{j_1',m_1}_{j_1'',m_1-\eta,\lambda_2,\eta} \mathrm{C}^{j_2',m-m_1}_{j_2'',m-(m_1-\eta),\lambda_1,-\eta} \Bigg[(-1)^{\lambda_1+\lambda_2} \sum_{k_1'=-j_1'}^{+j_1'} b_{k_1'} b_{k_1'-\mu_2} \sum_{k_2'=-j_2'}^{+j_2'} b_{k_2'} b_{k_2'-\mu_1} \mathrm{C}^{j_1',k_1'}_{j_1'',k_1'-\mu_2,\lambda_2,\mu_2} \mathrm{C}^{j_2',k_2'}_{j_2'',k_2'-\mu_1,\lambda_1,\mu_1}$ $+(-1)^{\mu_1+\mu_2+\lambda} \sum_{k_1'=-j_1'}^{+j_1'} b_{k_1'} b_{k_1'+\mu_2} \sum_{k_2'=-j_2'}^{+j_2'} b_{k_2'} b_{k_2'+\mu_1} \mathrm{C}^{j_1',k_1'}_{j_1'',k_1'+\mu_2,\lambda_2,-\mu_2} \mathrm{C}^{j_2',k_2'}_{j_2'',k_2'+\mu_1,\lambda_1,-\mu_1} \Bigg]$

Notes: In the code, for efficiency, the calculation of Clebsch-Gordon coefficients is precluded by a check of triangle rule, to avoid computing those coefficients that are zero by definition.

In the cases of asymmetric tops, the range of summation over k_1' is either $-j_1' + \mu_1 \leq k_1' \leq j_1'$ or $-j_1' \leq k_1' \leq j_1' - \mu_1$, depending on the sign in front of μ_1 (in the formula above this is not indicated explicitly, for transparency, but is implicit in the code). And the same for k_2'.

Basis set expansion for `SYS_TYPE=2,6` (that include vibrational motion) is not implemented in this release of the code but will probably be implemented in the future. Meanwhile, users can use Option 1 to compute matrix elements (by direct integration).

VIII. Sampling of Initial Conditions and Monte-Carlo Simulations

By default, the initial conditions for MQCT trajectories are set-up for all integer values of the orbital angular momentum quantum number ℓ in the range $0 \le \ell \le \ell_{\max}$, with $\ell_{\max}$ determined either by the maximum value of total angular momentum and the angular momentum of the initial state as $\ell_{\max} = J_{\max} - j$, or by the maximum impact parameter and collision energy as $\ell_{\max} = b_{\max}\sqrt{2\mu U}/\hbar$. The results of calculations are summarized in three files as follows: Individual state-to-state transition probabilities, ordered according to the channels list, are printed in the file `OPACITY_FUNCTIONS_L.out`, whereas the total elastic and inelastic probabilities are printed in the file `TOTAL_PROBABILITIES_L.out`, as dependencies on the impact parameter b and the orbital angular momentum l. Partial cross-sections for each channel (transition probabilities multiplied by $2J + 1$) are printed in the file `PARTIAL_CROSS_SECTIONS_J.out` as a function of total angular momentum J. First few lines of each file contain information about the initial state (state number in the list, j and m), the collision energy U and the header line, followed by the table of results. The values of deflection angle of MQCT trajectories and the phase shift obtained from them are printed in the file `DEFLECTION_FUNCTION.out`. These files are generated by default. We recommend that users inspect all these dependencies after each program's execution, to make sure that the overall behavior of the system is reasonable.

These dependencies are often smooth, so, the users have an option of skipping some values of ℓ using a keyword `DL` to make the overall calculations more affordable. For example, if `DL=3` is indicated, the code will only do calculations for $\ell = 1, 4, 7, 10, ...$ (one value of ℓ in the middle of each interval) and then will interpolate all results to fill the gaps. In those cases when the behavior of opacity function is different in the short-range and long-range interaction regions, users may want to use two different values of `DL`. For example, if one wants to use `DL=3` in the range $b \le 10$ Bohr and then switch to `DL=20` in the range $10 \le b \le 30$ Bohr, user should indicate `DL=3, DL_LR=20, B_SWITCH=10, B_MAX=30.`

For heavier molecules and higher collision energies (large $J_{\max}$ or $\ell_{\max}$) and/or for highly excited rotational states (large j_1 and j_2), the number of initial states covered may become prohibitively expensive (large j, many values of m). Running MQCT trajectories for all possible initial degenerate states, which is a default in our code, may become computationally expensive and, in fact, is unnecessary. In such cases, it is more efficient to sample the values of ℓ, j and m randomly and simultaneously, using an efficient multi-dimensional Monte-Carlo procedure. This option is initiated by the following keywords: `MONTE_CARLO=YES, NMB_TRAJ=200,` where `NMB_TRAJ` is the number of trajectories to compute. Note that in the case of Monte-Carlo sampling the elastic cross section is computed only approximately (it depends on the choice of $b_{\max}$ and should be ignored), while the differential cross section is not computed at all (due to the lack of accurate phase information). The random sampling procedure implemented in this version of MQCT is somewhat different from the previously published [Mandal 2021]. For all molecule + molecule system types it starts from sampling the initial value of j through the range $|j_1 - j_2| \le j \le j_1 + j_2$. This sampling is weighted, to give proportionally more weight to higher values of j that have more m states. For the molecule + atom system types, this step is not needed since the value of j is uniquely determined by the initial channel. Second step is sampling of the initial value of m through the

range $0 \leq m \leq j$. This sampling is again weighted, to give twice more weight to $m \geq 1$ states (relative to $m = 0$) in order to account for negative values of m. Finally, the value of ℓ is sampled uniformly through the range $0 \leq \ell \leq \ell_{\max}$. This procedure replaces a multi-dimensional sampling with one Monte-Carlo sampling, which is very efficient. All properties associated with individual trajectories in the Monte-Carlo sample are provided in the file `MC_InitialCond.DAT` which includes trajectory number, initial state index, corresponding values of j and m, parity of the initial state (relevant to the case of identical collision partners), the value of ℓ and a statistical weight of each trajectory. Here, the weight is the frequency of a particular trajectory being sampled. In the cases when `NMB_TRAJ` indicated by user is greater or equal to the total number of trajectories possible for given initial channel and $\ell_{\max}$, the weights are automatically adjusted to their asymptotic values ($w = 1$ for $m = 0$ and $w = 2$ for $m \geq 1$, which accounts for positive and negative values of m). By referring to this file users can obtain all details about the sampling process. The results of Monte-Carlo calculations are summarized in three output files as follows: Inelastic state-to-state transition probabilities for individual trajectories are printed in the file `MC_Probabilities.out`. Evolution of Monte-Carlo cross-sections for each inelastic channel, as more and more trajectories are added to the sample, is printed in the file `MC_Convergence.out`. The `MC_Statistics.out` file contains information regarding the evolution of statistical error estimates for all inelastic cross sections, as more and more trajectories are added to the sample. The maximum statistical error observed among all inelastic channels is written into the main output file `CROSS_SECTIONS.out`, together with the value of Monte Carlo coverage, which represents the percentage of sampled trajectories relative to the total number of trajectories possible for given initial channel and $\ell_{\max}$. If additional trajectories are needed (to reduce statistical error), users have to raise the value of `NMB_TRAJ` and run the calculations one more time. The code automatically adds new trajectories to the sample and updates all the output files. Often 100 trajectories are sufficient for a semi-quantitative prediction of cross sections. Monte-Carlo calculations with 200 trajectories give results within a few percent of the regular MQCT calculations.

Calculations of differential cross sections involves phase information and computation of a coherent sum over all partial scattering waves, and thus requires propagation of MQCT trajectories for *all* allowed integer values of ℓ, which is default in the code [Mandal 2018]. The keyword `DIFF_CROSS=YES` can be used to request construction of the differential cross section. Angular resolution of the differential cross section is defined by `ANG_RES`. Angular dependence of differential cross section is printed into the file `DIFF_CROSS.out`.

However, when the Monte-Carlo sampling of the initial conditions is requested (for numerical efficiency, see above) the values of ℓ will be chosen randomly and only a few of them may be available for each initial degenerate state jm. In this case the meaningful determination of the deflection function, scattering phase and the differential cross section is technically challenging, and is not implemented in the code. So, `MONTE_CARLO=YES` should not be used simultaneously with calculations of the differential cross section. Same considerations apply to the integral cross section for the elastic scattering channel, since it also requires the scattering phase. If the `MONTE_CARLO=YES` option is used, the value of elastic cross section should normally be ignored as explained above

(zero is printed in the output file). In many applications, the elastic and/or differential cross section are not needed. Then `MONTE_CARLO=YES` is the best option.

If the elastic and/or differential cross sections are needed for complex systems, the most efficient and robust approach is to use the optional keyword `DL`, in order to skip some values of ℓ and thus make the overall calculations more affordable. For calculation of the differential cross section the code still computes a coherent sum over all ℓ, but within the "boxes" of size `DL` the same values of scattering phase and probability amplitudes are used. The magnitude of `DL` becomes a convergence parameter in this case and should be carefully checked by convergence studies.

There are several options in the code that allow users to visualize MQCT trajectories, or their most important properties. Opacity functions and deflection functions are printed by default, which gives dependencies of transition probabilities and scattering angle on ℓ and b (but only for `MONTE_CARLO=NO`, which is default). Another useful option is to use keyword `PRN_TRJCT` to print out all information for a trajectory with indicated value of ℓ. By default, the information from trajectories with the initial $j = \max(j_1, j_2)$ and $m = 0$ is printed. Other values of the initial j and m can be specified using optional keyword, for example: `PRN_JM=3,3`. Plotting and inspecting the deflection function, opacity functions and trajectories is recommended, particularly at low scattering energies, when trajectories may be trapped at certain values of ℓ.

There are options in the code to deal automatically with trapped trajectories, if those occur. For such trajectories the number of loops (due to mutual rotation of collision partners around the origin), and the number of periods (due to mutual oscillations of collision partners along the intermolecular distance *R*) is determined. The option `NMB_LOOPS=2` commands to stop propagation of orbiting trajectories after two full loops (the default value is 1). The option `NMB_OSCIL=5` commands to stop propagation at the fifth outer turning point (the default value is 1). When such trapped trajectory is forced to stop, it is still analyzed in a standard way. However, the result of such analysis is somewhat arbitrary, since the termination point is also arbitrary. The option `NO_RESONANCE=YES` tells the code to remove looping and oscillating trajectories from analysis, which can be used for calculation of non-resonant contribution to the integral cross sections.

Note that differential cross sections and the elastic channel integral cross sections can't be computed rigorously at collision energies when at least one resonant trajectory is present, since in such cases the deflection function is undefined in the range of small values of ℓ. A good recipe for extracting resonance information from trapped trajectories is yet to be found, but AT-MQCT helps in this respect. In any case the code automatically detects trajectories that exhibit resonant behavior and prints some basic information about these trajectories in the file `RESONANCE_TRAJECT.out`, including the relevant values of ℓ. Then, if desired, the user can rerun the code with the option `PRN_TRJCT` employed, to obtain more detailed information for each resonant trajectory.

IX. Convergence Studies and Propagation of MQCT Equations

All output files have extension `*.out`. System setup is written into the file `USER_INPUT_CHECK.out` and should be checked by user for correctness. The file `STATES.out` (written if the option `PRINT_STATES=YES` is chosen) contains the list of all quantum states involved in calculations, including their quantum numbers and a state number in the overall list, assigned by MQCT. Cross sections for transitions from the initial state to the final states (all states of the basis, including the elastic channel) are listed in the output file `CROSS_SECTIONS.out`, for each effective collision energy specified in the input. For each transition, the actual collision energy `E_COLL` is also given in the output file, which depends on the effective collision energy `U_ENERGY` and the state-to-state energy difference. These are different for different transitions, particularly at lower collision energies, which is a property of the mixed quantum/classical approach, discussed in detail in several recent papers (the so-called Billing correction). Next in the output file goes an important information about the largest values of energy conservation error, and the probability conservation error (both given as % of the initial value) encountered during the propagation. Users should check these numbers to ensure that they are reasonable (say below `1.0E-03`). Excessive values indicate that modification of the propagation parameters in the input file is needed. The last in the output file is CPU-time statistics of the code execution. More detailed information about propagation accuracy can be found in the file `INTEGRATOR_ERRORS.out`, where the data collected for the less accurate (worst) trajectory in the batch are printed, for each collision energy. Note that the energy conservation error is computed and printed by default but can be disabled using a keyword `ENERGY_ERROR=NO` in order to reduce numerical effort, particularly in CS-MQCT and AT-MQCT calculations where the total energy is not conserved anyway.

Convergence studies with respect to several input parameters should be carried out by the user. For the quantum part of the system, sensitivity of results (cross section values) should be checked with respect to the basis set size `NMB_CHNLS`, the range of molecule-molecule distances used to compute transition matrix elements `RMIN` and `RMAX`, the corresponding number of points `GRD_R`, and the number of integration points for each internal degree of freedom (parameters `GRD_ANG1`, `GRD_ANG2` and `GRD_ANG3`). If the PES is represented by expansion over analytic functions, then the convergence parameter is the number of terms `NMB_TERMS` (and, of course, what terms are included). For trajectory propagation, sensitivity of energy and probability conservation errors should be checked with respect to step size `TIME_STEP`, and, if the adaptive step-size is used, the value of tolerance `EPS_ODEINT` and the minimum time step `MIN_TMSTP`. For sampling of the initial conditions, convergence parameter is the upper limit of total angular momentum `JTOTU` (or, optionally, the maximum value of collision impact parameter `B_IMPCT`). If optional Monte-Carlo sampling of the initial states is chosen (`MONTE_CARLO=YES`) then the number of trajectories `NMB_TRAJ` is also a convergence parameter. The value of `RMAX` may also affect MQCT trajectories, since it is used to set up the initial molecule-molecule separation. Note that during trajectory propagation the value of `RMAX` can be specified larger than the range covered by the grid of precomputed matrix elements. In this case, the code will automatically set to zero all matrix elements in the asymptotic range. If trajectory penetrates the range of distances smaller than `RMIN` (e.g., due to high collision energy) a linear function is used for

extrapolation of matrix element into the repulsive short-distance range. One relevant keyword is MIJ_SHIFT. By default, each matrix element is automatically shifted by its value at the last point of *R*-grid, to ensure that no transitions occur in the asymptotic region. This shift can be disabled by indicating MIJ_SHIFT=NO.

Often, user will want to increase the number of channels relative to the previously run calculations, for example, in order to check convergence with respect to the basis set size, or to do calculations at higher collision energy, where the number of channels is typically larger. In this case our code allows us to add new needed elements to the existing transition matrix, without re-computing the entire matrix. This is achieved, simply, by rerunning the code with new increased number of channels, using any method of channel specification available, for example, increasing the value of keyword NMB_CHNLS and adding new channels to the list CHNLS_LIST (or increasing the value of keyword EMAX). Users need to indicate READ_MTRX=YES so that the code will read the existing matrix from file, check whether all needed matrix elements are present, automatically compute the missing elements, update the matrix file and/or proceed with trajectory calculations, depending on what options are indicated by the user. One requirement is that *R*-grid remains identical to the one used in the previous calculations. However, the way of computing elements of the state-to-state transition matrix can be changed, for example, by increasing the number of integration points for the internal coordinates, or increasing the number of terms in the PES expansion, etc. The only requirement is that new channels are added at the end of the list, not at the beginning or stuffed in the middle (automatically taken care of if EMAX is used.

Also, the user can always run calculations of dynamics with the number of channels smaller than that in the saved matrix file MTRX_UF.dat. The code will read the file and choose only those matrix elements that are necessary for this run. Again, the requirement is that the active channels are listed continuously from the beginning of the list, and only the channels at the end of the list can be omitted. If the user wants to exclude some states from the beginning or from the middle of the list, the keyword EXCLUDE_STATES can be used as explained in Sec. VI (the corresponding data file should be provided).

Besides the default RK4 propagator, one can choose an adaptive step-size method by indicating:

```
PROPAGATOR=ODEINT, TIME_STEP=500.0, MIN_TMSTP=10.0, EPS_ODEINT=1.0E-3
```

This method adjusts time-steps along each trajectory trying to keep accuracy below EPS_ODEINT, and is a slightly modified version of the code from Numerical Recipes. Our version of this propagator enforces MIN_TMSTP to avoid an excessively long integration near the turning point. We recommend that users run their first calculations using the default RK4 propagator to determine a suitable value of TIME_STEP for their system and collision conditions (by monitoring the conservation of energy and norm in the output file). Then, one can try to switch to the ODEINT by simultaneously setting MIN_TMSTP equal to this found value and increasing the value of TIME_STEP by an order of magnitude or even more (which will play the role of the maximum time step allowed in ODEINT). We found that for the systems with deep molecule-molecule attraction potentials, such as dipole-dipole, this approach gives a considerable

computational advantage. Also, our ODEINT propagator has an option of SINGLE_STEP that commands to integrate the entire trajectory in one step. In this case the code automatically estimates a termination time for each trajectory (based on the impact parameter, the initial separation of collision partners, and the collision energy) and uses this number as maximum time step. This is the most efficient propagation option recommended for the production runs. Note that if SINGLE_STEP is chosen, then no propagation information is printed along the trajectory, no phases are computed, and thus no elastic or differential cross sections are calculated.

X. Molecular Symmetry and Identical Collision Partners

Rotational states of asymmetric-top molecules are split onto two groups, called *para*- and *ortho*-states. We define them based on what values of the quantum number *k* participate in expansion of wave function over the basis of symmetric-top eigenstates (expansion coefficients b_k). Namely, for each *j*, *even* values of *k* produce para-states, while *odd* values of *k* produce ortho-states. Even and odd values of *k* never mix in the same $j_{k_a k_c}$ state. For symmetric molecules, such as H_2O, transitions are allowed within each group only, and are exactly forbidden between the para- and ortho-states, due to the symmetry of potential of interaction of the molecule with a quencher (any quencher). Including all states would not cause a problem but would be meaningless since the efficiency of calculations would be reduced. (However, including ortho- and para-states in the same calculations can be done as a test of numerical convergence: if calculations are accurate, these forbidden transitions should have numerically negligible matrix elements and cross sections; if they are not, that something is wrong either with PES interface, of with numerical convergence of calculations.)

If the states are specified explicitly as a list, the user should take care of this issue manually. But, if the basis set is generated automatically (*e.g.*, using the keyword EMAX), user has an option to indicate SYMMETRY=YES, for reducing the basis set size to one symmetry only, depending on symmetry of the initial state (as indicated in the input file). In the molecule + molecule case, symmetry consideration will be applied to the states of each molecule individually, for example, in a system like $H_2O + H_2$ one can run *four* sets of independent calculations for *p*-H_2O + *p*-H_2, *p*-H_2O + *o*-H_2, *o*-H_2O + *p*-H_2 and *o*-H_2O + *o*-H_2. Note, however, that in asymmetric molecules, such as HDO, all transitions are allowed, and all states should be included in the basis. Thus, indicating the keyword SYMMETRY=YES, would lead to unphysical results in this and other cases with no symmetry, and should not be done.

Ortho and para states of asymmetric top rotor are identified by the code (by analyzing expansion coefficients b_k) and are given assignments: $\kappa = 0$ for para states and $\kappa = 1$ for ortho states (here we use Greek symbol kappa). They can easily be distinguished by users too, based on $j_{k_a k_c}$ assignment of a state: for para states the sum $k_a + k_c$ is even (e.g., 0_{00}, 1_{11}, 2_{20}, etc.) while for ortho states the sum $k_a + k_c$ is odd (e.g., 1_{10}, 1_{01}, 2_{10}, etc.).

Another symmetry property of a molecular state is its *inversion parity*, determined by the signs of two expansion coefficients b_k that correspond to two symmetric-top basis functions with the same absolute value of $|k|$. For *positive* parity states, the two components have the *same* expansion coefficients, $b_{-k} = b_{+k}$ (which corresponds to constructive superposition, “in phase”), while for

negative parity they have *opposite* signs: $b_{-k} = -b_{+k}$ (destructive superposition, "out of phase"). This is equivalent to the inversion parity of symmetric-top functions defined in Sec. I above, so, we can write $b_{-k} = \varepsilon\, b_{+k}$ where $\varepsilon = \pm$ is parity. Our code determines the parity of each state (by analyzing the signs of b_k). In the file `USER_INPUT_CHECK.out` user can find the assignments of asymmetric-top rotor states in terms of inversion parity and ortho/para character.

When two identical molecules are collided (e.g., two diatomics in `SYS_TYPE=5,6` or two asymmetric-top rotors in `SYS_TYPE=0`, *e.g.*, H_2O + H_2O) one may want to treat them as *indistinguishable*, by indicating the keyword `IDENTICAL=YES` in the input file. In this case only one set of rotational constants, and only the unique channels should be specified. For example, if the state $(j_{1_{k_{a_1}k_{c_1}}}, j_{2_{k_{a_2}k_{c_2}}}) = (1_{11}, 0_{00})$ of H_2O + H_2O is already specified, one should not include the state $(0_{00}, 1_{11})$ since two molecules are treated as indistinguishable. However, for a system of two indistinguishable molecules we define two symmetrized wavefunctions of the system, those with positive and negative *exchange parities* (±):

$$|\, jmj_1j_2\rangle^{\pm} = \frac{|\, jmj_1j_2\rangle_{\Lambda_1,\Lambda_2} \;\pm\; |\, jmj_1j_2\rangle_{\widetilde{\Lambda}_2,\widetilde{\Lambda}_1}}{\sqrt{2(1+\delta_{j_1j_2})}}$$

The second term in this expression (subscript $\widetilde{\Lambda}_2, \widetilde{\Lambda}_1$) is obtained from the first one (subscript Λ_1, Λ_2) by swapping two collision partners (molecules 1 and 2). Such symmetrized states are energetically degenerate but transitions between them are forbidden. If the keyword `IDENTICAL=YES` is indicated in the input file, MQCT code will carry out two consecutive independent calculations of collision dynamics, one with the states of positive exchange parity $|\, jmj_1j_2\rangle^{+}$, and second with the states of negative exchange parity $|\, jmj_1j_2\rangle^{-}$ and will average the results of two calculations using the weighting factor indicated by keyword `WGHT_POSPAR` (as explained below).

It is possible to show that this expression can be rewritten in the following form:

$$|jmj_1j_2\rangle^{\pm} = \frac{|jmj_1j_2\rangle_{\Lambda_1,\Lambda_2} \;\pm p|\, jmj_2j_1\rangle_{\Lambda_1,\Lambda_2}}{\sqrt{2(1+\delta_{j_1j_2})}}$$

where the *overall parity* of the rotational state of two asymmetric-top rotors is introduced as $p = (-1)^j(-1)^{\kappa_1+\kappa_2}\varepsilon_1\varepsilon_2$, where κ_1 and κ_2 correspond to the para/ortho character (0 or 1) of the two states, while ε_1 and ε_2 correspond to their inversion parities (± 1). Note that j_1 and j_2 are swapped in the second term, which is a shortened notation that means that we use quantum state 2 for molecule 1, whereas quantum state 1 is used for molecule 2 (this includes all quantum numbers $j_{k_ak_c}$ and the corresponding set of expansion coefficients b_k). Similar, $\delta_{j_1j_2} = 1$ only if two molecules are in the same quantum state $j_{k_ak_c}$ with the same set of expansion coefficients b_k. For a system of two linear-top rotors the expression of overall parity is simpler $p = (-1)^j$.

Trivial cases occur if the states of two particles are the same, such as $(0_{00}, 0_{00})$ or $(1_{11}, 1_{11})$. Then only one exchange parity is possible (and is equivalent to the original state of distinguishable collision partners), because the other parity annihilates the overall

wavefunction. Since $\kappa_1 = \kappa_2$ and $\varepsilon_1 = \varepsilon_2$ the expression for parity simplifies into $p = (-1)^j$. Consider $(1_{11}, 1_{11})$ channel as example. It has three components, $j = 0, 1$ and 2. For even states $j = 0$ and 2 we have $p = +1$ and therefore only positive exchange parity is meaningful. In contrast, for $j = 1$ we have $p = -1$ and therefore only negative exchange parity is meaningful. The code recognizes these cases and takes care of them automatically.

If transition matrix is calculated by direct integration (Option 1), the code will symmetrize wavefunctions as explained above and will compute matrix elements for state-to-state transitions within each parity p, setting other matrix elements equal to zero. But, if transition matrix is calculated by PES expansion (Option 2), the code will compute four matrix elements just as in the case of distinguishable collision partners (without symmetrization of wavefunctions) and then will symmetrize the matrix, by combining matrix elements in two different ways ($\pm$ for calculations with positive and negative exchange parity):

$$\langle j''mj_1''j_2''|V(\Lambda_1,\Lambda_2)|j'mj_1'j_2'\rangle^{\pm} = \frac{1}{\sqrt{2(1+\delta_{j_1',j_2'})2(1+\delta_{j_1'',j_2''})}} \begin{bmatrix} & \langle j''mj_1''j_2''|V(\Lambda_1,\Lambda_2)|j'mj_1'j_2'\rangle \\ \pm\ p' & \times \langle j''mj_1''j_2''|V(\Lambda_1,\Lambda_2)|j'mj_2'j_1'\rangle \\ \pm\ p'' & \times \langle j''mj_2''j_1''|V(\Lambda_1,\Lambda_2)|j'mj_1'j_2'\rangle \\ +\ p'p'' & \times \langle j''mj_2''j_1''|V(\Lambda_1,\Lambda_2)|j'mj_2'j_1'\rangle \end{bmatrix}$$

Here p' and p'' represent the overall parity (defined above) of the initial and final states. Analysis of this expression shows that, in the case of indistinguishable collision partners, transitions are possible only between the states of the same overall parity, $p'' = p'$. It also follows that for the system of two indistinguishable molecules there are three (not four) manifolds of uncoupled rotational states: para + para, ortho + ortho, and para + ortho, for which the calculations can be done independently. In the first two cases (two para or two ortho states) the elements of symmetrized transition matrix are different for two exchange parities ($\pm$) and are different from those of distinguishable particles case (thus, the keyword `IDENTICAL=YES` is required for the code to do two consecutive runs and compute the average). However, in the case of para + ortho states the symmetrized matrix is the same for two exchange parities and is exactly equivalent to the case of non-identical collision partners (so, for this subset of states the calculations can and should be done without the keyword `IDENTICAL=YES`).

Note that since the definition of the overall parity p includes $(-1)^j$, and since the value of j is varied through the range $|j_1 - j_2| \leq j \leq j_1 + j_2$, different j-states within each channel indicated as input will correspond to different parities p. Therefore, in general, the states with both positive and negative values of p will be found within any initial channel, and therefore two rounds of calculations will be caried out automatically by the code (except the trivial case when, say, $j_1 = 0$, and therefore only one value of $j = j_2$ is possible).

If the basis set is specified explicitly as a list of states, the user should make efforts to exclude the uncoupled states. If the basis set is generated automatically (employing the keyword `EMAX`, or keywords `EMAX1` and `EMAX2`), user has option to minimize the number of channels by choosing `SYMMETRY=YES`. Again, only the states coupled to the initial state would be included, based on ortho- and para-character of the states. These symmetry considerations remain valid within each block of states of the overall parity p.

Each of the two calculations carried out by the code in the case of `IDENTICAL=YES` includes all vales of ℓ, both even and odd. The results are converted then into four cross sections: $\sigma_{\rm evn}^{(+)}$ and $\sigma_{\rm odd}^{(-)}$ from the first run (with positive exchange parity states) and $\sigma_{\rm evn}^{(-)}$ and $\sigma_{\rm odd}^{(+)}$ from the second run (with negative exchange parity states), respectively. These are used to compute $\sigma^{(+)} = \sigma_{\rm evn}^{(+)} + \sigma_{\rm odd}^{(+)}$ and $\sigma^{(-)} = \sigma_{\rm evn}^{(-)} + \sigma_{\rm odd}^{(-)}$, and finally their weighted average $\sigma = w\sigma^{(+)} + (1-w)\sigma^{(-)}$ using `WGHT_POSPAR`, that user should provide in the input file. The default value is one, which leads to the overall cross $\sigma = \sigma^{(+)}$ (appropriate for H_2O + H_2O system). The value of `WGHT_POSPAR=0.d0`, in contrast, will result in $\sigma = \sigma^{(-)}$. Note that although, quantum mechanically, only even values of ℓ should be included into $\sigma_{\rm evn}^{(\pm)}$ and only odd values of ℓ should be included into $\sigma_{\rm odd}^{(\mp)}$, this is likely to matter at very low collision energies only, when just a few values of ℓ contribute. The range of MQCT applicability is in the semiclassical regime when many values of ℓ contribute and all ℓ-dependencies are smooth (scattering angle, transition probabilities). Therefore, for the improved sampling, we use both even and odd values of ℓ in both $\sigma_{\rm evn}^{(\pm)}$ and $\sigma_{\rm odd}^{(\mp)}$, and merely divide the result by two.

XI. Examples of input files and potential energy surfaces

Table 10 lists input files provided with MQCT_2023 code as examples. They cover calculations for all system types, with different modes of input (including the external data, such as pre-computed wave functions), different ways of computing (direct integration, basis set expansion) and saving the matrix (single file, formatted/unformatted, parallel IO, re-truncation), different sampling schemes (regular, steps in ℓ, Monte-Carlo sampling of ℓ, j, m), different kinds of trajectory propagation (CC, CS, AT) and analysis of the final data (total and differential cross sections, printing individual trajectories, handling resonances, etc.).

Table 10: A list of potential energy surfaces and example input files for different system types provided with the MQCT_2023

SYS_TYPE	PES Included	Name of the input file	Descriptions of examples provided
1	N_2 + O	N2_O_Single_Run.inp	Compute matrix and propagate trajectories a single run, and differential cross sections
	CO + He	CO_He_mtrx_dir_int.inp CO_He_dynamics.inp	Compute matrix by direct integration in the first run, then propagate trajectories in the separate second run
2	CO + H	CO_H_Morse.inp	Use Morse parameters to construct vibrational wavefunctions of a non-rotating Morse oscillator in the ro-vibrational calculations (an approximate treatment).
		CO_H_external_vib_wfn.inp	Compute matrix using ro-vibrational energies and wavefunctions supplied externally as USER_DEFINED_BASIS.DAT
3	C_6H_6 + He	C6H6_He_exp_term_list.inp	Generate the list of expansion terms in the file EXPANSION_TERMS_LIST.DAT
		C6H6_He_AT_MQCT_1st_run.inp C6H6_He_AT_MQCT_2nd_run.inp	Use AT_MQCT decoupling approximation for propagation of trajectories in two runs with constant time-step RK4 propagator
4	H_2O + He	H2O_He_CalcExp.inp	Compute expansion coefficients and save to PES_EXPAN_TERMS.DAT
		H2O_He_external_rot_wfn.inp	Compute matrix elements using rotational states (energies and wave function expansions) supplied externally in the input file USER_DEFINED_BASIS.DAT
	$HCOOCH_3$ + He	MF_He_CS_MQCT.inp	Propagate trajectories using coupled states approximation, CS-MQCT
5	CO + CO	CO_CO_MTRX_using_exp.inp	Compute matrix elements for indistinguishable collision partners using PES expansion
		CO_CO_Resonance.inp	Identify and print resonance trajectories using NMB_LOOPS and NMB_OSCIL
6	CO + H_2	CO_H2_Parallel_IO.inp	Compute and save matrix elements using the PARALLEL_IO keyword
		CO_H2_Prob_spline.inp	Read matrix elements using PARALLEL_IO; use DL for orbital angular momentum ℓ to skip trajectories and then use PROB_SPLINE to interpolate probabilities
7	ND_3 + D_2	ND3_D2_VGridFile.inp	Compute VGRID_UF.DAT to store the PES at the points of the grid
		ND3_D2_IR_BGN_FIN.inp	Calculate PES_EXPAN_TERMS.DAT in several runs for different parts of *R*-grid
8	H_2O + H_2	H2O_H2_exp_term.inp	Generate the list of expansion terms in the file EXPANSION_TERMS_LIST.DAT
		H2O_H2_Parallel_IO.inp	Compute, save and read matrix elements using the PARALLEL_IO keyword
9	H_2O + NH_3 (pairwise LJ)	H2O_NH3_MTRX_Retruncation.inp	Use MIJ_CUTOFF keyword to re-truncate and save the matrix of smaller size
0	H_2O + H_2O	H2O_H2O_AT_MQCT_1st_run.inp H2O_H2O_AT_MQCT_2nd_run.inp	Use AT_MQCT decoupling approximation for propagation of trajectories in two runs using ADAPTOL
		H2O_H2O_Identical.inp	Calculate matrix elements for identical collisional partners using keyword IDENTICAL_PES symmetrized expansion of potential energy surface
		H2O_H2O_MonteCarlo.inp	Compute cross sections using random sampling of the initial conditions

Note: Examples for SYS_TYPE=6,8,9,0 include externally provided wavefunctions. Examples for SYS_TYPE=1 includes EMAX, and JMAX, and SYS_TYPE=2 includes NMB_ENRGS. Also, SYS_TYPE=3 includes AXL_SYM1 and EQU_SYM1, and SYS_TYPE=8 includes ODD_L1L2 optional keywords.