MidasCpp

Molecular Interactions Dynamics And Simulation
C++ package
Chemistry Program Package
Completely Pathetic Programming

User's Guide

Contents

Preface

This is the documentation for MidasCpp version v2019.4.0

0.1 Distribution and License

MidasCpp[1] is distributed under the GNU Lesser General Public License, version 2.1. A description of MidasCpp is available on https://midascpp.gitlab.io/ and the source code is hosted on GitLab, which is *the* source of distribution including updates and communication.

> **Consider to cite:**
>
> 1. O. Christiansen, D. Artiukhin, I. H. Godtliebsen, E. M. Gras, W. Győrffy, M. B. Hansen, M. B. Hansen, E. L. Klinting, J. Kongsted, C. König, D. Madsen, N. K. Madsen, K. Monrad, G. Schmitz, P. Seidler, K. Sneskov, M. Sparta, B. Thomsen, D. Toffoli, A. Zoccante, *MidasCpp*, version 2019.04.0

0.2 Standard Disclaimer

MidasCpp is an experimental code under constant development. There are no guarantees concerning the performance of the code or the correctness of the results. The authors can not accept responsibility for any damage done or caused by the use of this program.

0.3 Citation

Any use of the program that results in published material should cite the program itself,

> "O. Christiansen, D. Artiukhin, I. H. Godtliebsen, E. M. Gras, W. Győrffy, M. B. Hansen, M. B. Hansen, E. L. Klinting, J. Kongsted , C. König, D. Madsen, N. K. Madsen, K. Monrad, G. Schmitz, P. Seidler, K. Sneskov, M. Sparta , B. Thomsen, D. Toffoli, and A. Zoccante, *MidasCpp*, version 2019.04.0"

as well as the original literature describing the essential MidasCpp functionalities used. The theoretical background chapters of the manual describes the background and list appropriate references. The essential `MidasCpp` references will be highlighted separately via a red box. Please cite these articles if you use the described functionality. An example for such a reference list you fill find at the end of this chapter for the `MidasCpp` program itself.

0.4 Authors

The development and distribution of MidasCpp is led by the initiator **Ove Christiansen**. The list of authors contributing over time includes:

Contributing authors (listed alphabetically by surname):

- Denis Artiukhin
- **Ove Christiansen**
- Ian Heide Godtliebsen
- Eduard Matito Gras
- Werner Győrffy
- Mikkel Bo Hansen
- Mads Bøttger Hansen
- Emil Lund Klinting
- Jacob Kongsted
- Carolin König
- Diana Madsen
- Niels Kristian Madsen
- Kasper Monrad
- Gunnar Schmitz
- Peter Seidler
- Kristian Sneskov
- Manuel Sparta
- Bo Thomsen
- Daniele Toffoli
- Alberto Zoccante

References

Midas Related Publications

[1] O. Christiansen, D. Artiukhin, I. H. Godtliebsen, E. M. Gras, W. Győrffy, M. B. Hansen, M. B. Hansen, E. L. Klinting, J. Kongsted, C. König, D. Madsen, N. K. Madsen, K. Monrad, G. Schmitz, P. Seidler, K. Sneskov, M. Sparta, B. Thomsen, D. Toffoli, A. Zoccante, *MidasCpp*, version 2019.04.0.

Part I

MidasCpp User's Guide

Part II

Theory

We outline in the following general aspects of the theory behind the MidasCpp options. In each section we describe the overall framework about specific aspects of quantum computations on the nuclear motion of molecules. We describe the key aspects of the implementation with issues to consider. We define the key references to read and cite for each topic. We give references the overall program input sections, and refer to the reference manual III for a detailed list of all input options.

Chapter 1

Coordinates

Coordinates is a important part of performing dynamical computations on molecules. For electronic wave functions Cartesian coordinates are widely used. For nuclei, the set of Cartesian coordinates of all the nuclei would be strongly coupled, and we would end with the problem of computing potentials and wave functions for a set of highly correlated variables. Often we can get a much better starting point by using the right set of coordinates. In some simple cases finding the right coordinates correspond essentially to solving the problem.

1.1 Normal-Coordinates and harmonic oscillator treatment of Molecular vibrations

1.1.1 Classical coordinate transformation approach

Often, the molecule can to a good approximation be described as vibrating with small amplitudes around some equilibrium configuration. Thus, it seems natural, at least in those cases, to describe the nuclear positions in terms of displacements from this configuration. We shall first consider this issue classically and then quantize.

The reference configuration is described by the nuclear coordinates $\mathbf{R}_P^0$ and the nuclear displacements are denoted $\mathbf{D}_P$. They are related by

$$\mathbf{R}_P = \mathbf{R}_P^0 + \mathbf{D}_P \tag{1.1}$$

Since $\mathbf{R}_P^0$ is time-independent we obtain that the time-derivative of $\mathbf{R}_P$ is equal to the time-derivative of $\mathbf{D}_P$

$$\dot{\mathbf{R}}_P = \dot{\mathbf{D}}_P \tag{1.2}$$

The center of mass coordinates defines three constraints on the nuclear displacements. The approximate separation of rotation from internal motion also gives rise to some criteria. We are here aiming at describing the internal molecular motion. Thus, there are 6(5) constraints between the $\mathbf{R}_P$ with 3 from translation and 3(2) from rotation.

We first introduce the $3N$ dimensional column vector $\mathbf{D}$ containing the Cartesian displacement coordinates of the atoms

$$\mathbf{D}^T = (d_{1X}, d_{1Y}, d_{1Z}, d_{2X}, d_{2Y}, d_{2Z}, ..., d_{NX}, d_{NY}, d_{NZ}) \tag{1.3}$$

We next introduce the diagonal $3N \times 3N$ matrix $\mathbf{G}$ containing the inverse masses on the diagonal

$$\mathbf{G} = \begin{pmatrix} M_1^{-1} & 0 & . & . & . & . & . & . & & 0 \\ 0 & M_1^{-1} & & & & & & & & . \\ . & & M_1^{-1} & & & & & & & . \\ . & & & M_2^{-1} & & & & & & \\ . & & & & M_2^{-1} & & & & & \\ . & & & & & M_2^{-1} & & & & \\ & & & & & & & & . & \\ & & & & & & & & . & \\ . & & & & & & & & . & \\ 0 & . & . & & & & & . & M_N^{-1} \end{pmatrix} \tag{1.4}$$

The classical nuclear kinetic energy term T_n now can be written as

$$T_n = \frac{1}{2} \sum_P (\dot{\mathbf{R}}_P)^T M_P \dot{\mathbf{R}}_P = \frac{1}{2} (\dot{\mathbf{D}})^T \mathbf{G}^{-1} \dot{\mathbf{D}} \tag{1.5}$$

As a next step we introduce nuclear coordinates, $\mathbf{D}_M$ that are scaled with the square-root of the masses

$$\mathbf{D}_M = \mathbf{G}^{-\frac{1}{2}} \mathbf{D} \tag{1.6}$$

In terms of the $\mathbf{D}_M$ coordinates we have the following very simple form for T_n

$$T_n = \frac{1}{2} (\dot{\mathbf{D}}_M)^T \dot{\mathbf{D}}_M \tag{1.7}$$

In the next step we shall introduce a transformation of these coordinates to a set of coordinates $\mathbf{Q}$

$$\mathbf{Q} = \mathbf{L}^T \mathbf{D}_M \tag{1.8}$$

where $\mathbf{L}^T$ is a $3N \times 3N$ transformation matrix. We shall use the transformation from $\mathbf{D}_M$ to $\mathbf{Q}$ to simplify matters.

Now, the main complication lies in the potential energy term. The kinetic energy term can hardly be made simpler than the one in Eq.(1.7). We would like at this stage to maintain the kinetic energy T_n term in the simple form it has. This can be achieved by requiring the transformation matrix $\mathbf{L}$ to be orthogonal. Thus we require

$$\mathbf{L}^T \mathbf{L} = \mathbf{1} \tag{1.9}$$

$$\mathbf{L}\mathbf{L}^T = \mathbf{1} \tag{1.10}$$

With this first condition on $\mathbf{L}$ we easily obtain the back transformation

$$\mathbf{D}_M = \mathbf{L}\mathbf{Q} \tag{1.11}$$

The kinetic energy maintain the simple form in Eq.(1.7) now in terms of the $\mathbf{Q}$ coordinates

$$T_n = \frac{1}{2} (\dot{\mathbf{Q}})^T \dot{\mathbf{Q}} \tag{1.12}$$

Up to now we have considered only the kinetic energy for vibration, while the main complexity actually comes from the potential energy. The choice of $\mathbf{L}$ can be used to give the potential energy in a convenient set of coordinates that simplify the treatment. That is of course impossible for the most general choices for the potential. However, if we consider the reference condition to be a minimum for the potential energy a low-order Taylor expansion in the nuclear displacement coordinates gives (we write the potential as function of $\mathbf{D}$ - it is understood that if the potential

is given in terms of the $\mathbf{R}_P$ it is trivial to define a new function with $\mathbf{D}$ as argument when the reference configuration has been defined)

$$V(\mathbf{D}) = V(\mathbf{0}) + \sum_{P=1}^{N_n} \sum_\alpha \frac{dV}{dD_{P\alpha}}|_{\mathbf{0}} D_{P\alpha} + \frac{1}{2} \sum_{P,Q=1}^{N_n} \sum_{\alpha,\beta} \frac{d^2V}{dD_{P\alpha}dD_{Q\beta}}|_{\mathbf{0}} D_{P\alpha} D_{Q\beta} + ... \tag{1.13}$$

Since the reference configuration is chosen to be an equilibrium configuration the linear term vanishes. Defining the symmetric force-constant matrix $\mathbf{F}$ by

$$F_{P\alpha Q\beta} = \frac{\partial^2 V}{\partial D_{P\alpha} \partial D_{Q\beta}}|_{\mathbf{0}} \tag{1.14}$$

and neglecting terms higher than second order we obtain

$$V(\mathbf{D}) - V(\mathbf{0}) \approx \frac{1}{2}(\mathbf{D})^T \mathbf{F} \mathbf{D} = \frac{1}{2}(\mathbf{D}_M)^T (\mathbf{G}^T)^{\frac{1}{2}} \mathbf{F} \mathbf{G}^{\frac{1}{2}} \mathbf{D}_M = \frac{1}{2}(\mathbf{Q})^T \mathbf{L}^T \mathbf{G}^{\frac{1}{2}} \mathbf{F} \mathbf{G}^{\frac{1}{2}} \mathbf{L} \mathbf{Q} = \frac{1}{2}(\mathbf{Q})^T \mathbf{\Lambda} \mathbf{Q} \tag{1.15}$$

In Eq.(1.15) we have made use of the fact that the first derivatives vanishes at equilibrium. We have also used the coordinate transformations in Eqs.(1.6,1.8) and that since $\mathbf{G}$ is diagonal $(\mathbf{G}^T)^{\frac{1}{2}} = \mathbf{G}^{\frac{1}{2}}$. We have in Eq.(1.15) introduced the matrix

$$\mathbf{\Lambda} = \mathbf{L}^T \mathbf{G}^{\frac{1}{2}} \mathbf{F} \mathbf{G}^{\frac{1}{2}} \mathbf{L} \tag{1.16}$$

Using the orthogonality of $\mathbf{L}$ we obtain

$$\mathbf{G}^{\frac{1}{2}} \mathbf{F} \mathbf{G}^{\frac{1}{2}} \mathbf{L} = \mathbf{L} \mathbf{\Lambda} \tag{1.17}$$

Since $\mathbf{G}^{\frac{1}{2}} \mathbf{F} \mathbf{G}^{\frac{1}{2}}$ is a symmetric matrix it can always be diagonalized by an orthogonal matrix giving real eigenvalues. It is thus in agreement with our previous criteria for the $\mathbf{L}$ matrix if we require Eq.(1.17) to be an eigenvalue problem. The $\mathbf{\Lambda}$ matrix is thereby required to be a real diagonal matrix with eigenvalues $\lambda_k = \Lambda_{kk}$. The requirement that Eq. (1.17) is an eigenvalue problem determines only the eigenvectors up to a constant factor. A normalization factor is determined from the orthogonality constraint earlier discussed leaving only an undetermined phase. From imposing this eigenvalue criterion we obtain that the second order expanded potential energy achieves an even simpler form - it becomes a quadratic form without terms bilinear in the Q_k

$$V(\mathbf{D}) - V(\mathbf{0}) \approx \frac{1}{2} \sum_k \lambda_k Q_k^2 \tag{1.18}$$

Often, the eigenvalue problem is formulated as

$$\mathbf{G} \mathbf{F} (\mathbf{G}^{\frac{1}{2}} \mathbf{L}) = (\mathbf{G}^{\frac{1}{2}} \mathbf{L}) \mathbf{\Lambda} \tag{1.19}$$

which is easily obtained from Eq.(1.17) by multiplication with $\mathbf{G}^{\frac{1}{2}}$. Diagonalization of $\mathbf{G}\mathbf{F}$ thus determines the eigenvalues λ_k and from the eigenvectors the $\mathbf{L}$ matrix is obtained by multiplication with $\mathbf{G}^{-\frac{1}{2}}$. It is usually more advantageous to work with symmetric matrices, and the previous formulation is from that perspective advantageous.

Since the PES is invariant to translation and rotation, there are 6(5) coordinates that can be obtained from the cartesian by orthogonal transformation that leaves the V and its second order expansion unaltered. Accordingly, it can be shown that contributions from translation and rotation can be separated out and they can be identified as those giving zero eigenvalues in the above eigenvalue equation.

The total classical vibrational energy of the system described by the second order expanded potential becomes accordingly

$$E_{HO} = \frac{1}{2} \sum_{k=1}^{3N_n-6} (\dot{Q}_k^2 + \lambda_k Q_k^2) \tag{1.20}$$

Through a set of coordinate transformations we have obtained the vibrational energy as a sum of uncoupled harmonic oscillators. This makes an exact treatment possible in both classical and quantum mechanics which is the motivation for this formulation. The coordinates bringing about the separation into a set of independent oscillators are denoted normal coordinates.

We should remember that this simple form of the energy applies only for the vibrational part of the energy in the particular approximation where terms that are higher than second order in displacement from the reference configuration are neglected. We shall return to the full potential later. Furthermore, rotational motion and the interaction between vibrational and rotational motion is not taken into account. These are also important for some spectroscopic studies.

1.1.2 Quantization of the vibrational energy

We can now write the classical equations in terms of general coordinates Q and momenta P

$$E_{HO} = \frac{1}{2} \sum_{k=1}^{M} (P_k^2 + \lambda_k Q_k^2) \tag{1.21}$$

where $P_k = \dot{Q}$ and $M = 3N - 6(5)$. Making the transition to Quantum Mechanics we have

$$H = \sum_{k=1}^{M} H_k \tag{1.22}$$

where

$$H_k = -\frac{1}{2} \hbar^2 \frac{\partial^2}{\partial Q_k^2} + \frac{1}{2} \lambda_k Q_k^2 \tag{1.23}$$

Each H_k is a one-dimensional harmonic oscillator problem that can be solved by

$$H_k \psi_{n_k}^k(Q_k) = \hbar \omega_k (n_k + \frac{1}{2}) \psi_{n_k}^k(Q_k) \tag{1.24}$$

with

$$\omega_k = \sqrt{\lambda_k} \tag{1.25}$$

and $\psi_{n_k}^k(Q_k)$ is standard Harmonic oscillator eigenfunctions. Thus, with the total Hamiltonian as a sum of independent contributions, the total wave function becomes a simple product wave function

$$\Psi(Q_1, , , , , Q_M) = \prod_{k=1}^{M} \psi^k(Q_k) \tag{1.26}$$

We have, with some approximations on the way, gone all the way from the basic molecular Schrödinger to the actual quantum states describing dynamics. In this subsection we found bound state dynamics in the harmonic approximation.

1.2 Frequency scaled normal coordinates

One may wish to use frequency-scaled coordinates for increased numerical stability. In MidasCpp this is done in the fitting routine for polynomials and general fit functions. When this is used for the construction of a PES, this will give a different scaling of the operators in the output, and the frequency scalingfactor is denoted in the output. The relation between the normal coordinates Q and the frequency scaled coordinates $\tilde{Q}$ for each mode k is described in the following:

$$\tilde{Q}_k = \sqrt{\frac{\omega_k}{\hbar}} Q_k \tag{1.27}$$

Some basic results:

$$\frac{\partial^2}{\partial Q_k^2} = \frac{\omega_k}{\hbar} \frac{\partial^2}{\partial \tilde{Q}_k^2} \tag{1.28}$$

$$H_k = -\frac{1}{2}\hbar^2 \frac{\omega_k}{\hbar} \frac{\partial^2}{\partial \tilde{Q}_k^2} + \frac{1}{2}\omega_k^2 \frac{\hbar}{\omega_k} \tilde{Q}_k^2 \tag{1.29}$$

$$H_k = \hbar\omega_k \left(-\frac{1}{2} \frac{\partial^2}{\partial \tilde{Q}_k^2} + \frac{1}{2} \tilde{Q}_k^2 \right) \tag{1.30}$$

In the standard output MIDAS operator file (see section B.2.2) the resulting fitting coefficients correspond to coordinates scaled with both frequency and mass. In the following is an example of how to compare the obtained fit-parameters of a Morse function to the physical parameters for a diatomic molecule.

A Morse function can be written as:

$$V(r - r_e) = D_e \left(1 - \exp\left(\alpha(r - r_e)\right)\right)^2 \tag{1.31}$$

The displacement $r - r_e$ is the same as the previously introduced displacement vector $\mathbf{D} = \mathbf{M}^{-\frac{1}{2}}\mathbf{LQ}$. The potential expressed in frequency-scaled normal coordinates becomes:

$$V(\tilde{Q}_k) = D_e \left(1 - \exp\left(\tilde{\alpha}\tilde{Q}_k\right)\right)^2 \tag{1.32}$$

where

$$\alpha = \tilde{\alpha}\sqrt{\omega} \tag{1.33}$$

The relevant MidasCpp input for the fit-function may look like:

```
#4 OptFunc
F(Q,alpha,De) De*(1-EXP(-alpha*(Q)))*(1-EXP(-alpha*(Q))) (0,0.1,0.1)
...// a second Morse-function for a non-standard ADGA calculation:
#4 OptFunc
F(Q,alpha2,De2,disp) De2*(1-EXP(-alpha2*(Q-disp)))*(1-EXP(-alpha2*(Q-disp))) (0,0.1,0.1,0.0)
```

The electronic dissociation energy D_e is given directly in a.u.

The $\tilde{\alpha}$ parameter from the .mop file should be scaled with $\cdot\sqrt{\omega_k m}$, where m is the reduced mass of the molecule and both the frequency and mass is in atomic units. This also assumes that the vibrational coordinate only changes for one of the two atoms, as defined under "VIBCOORD" in the input file.

1.3 Semi-numerical gradient-based frequency analysis

MidasCpp is able to calculate and diagonalize the full (or a reduced) Hessian matrix for finding the normal coordinates (or some of them) of the molecular system at an equilibrium geometry.

At the heart of this lies the ability to calculate the result of a *Hessian transformation*, i.e. the result of multiplying the Hessian matrix $\mathbf{H}$ and some arbitrary coordinate vector, here denoted $\mathbf{b}_j$, to get $\boldsymbol{\sigma}_j = \mathbf{H}\mathbf{b}_j$. (In this section subscripts p, q, etc., shall denote a nucleus, α, β, etc., shall denote the Cartesian directions x, y or z, while subscripts i, j, etc., simply denotes a certain element (e.g. a basis vector) of a set. Furthermore $\mathbf{H}$ refers to the Hessian of the potential energy V, i.e. it is equivalent to the force-constant matrix $\mathbf{F}$.)

Writing out the multiplication explicitly, we find that the $p\alpha$'th element of the j'th column vector of $\boldsymbol{\Sigma}$ may be calculated (approximately) as

$$\sigma_{p\alpha,j} = \sum_{k\sigma} h_{p\alpha,k\sigma} b_{k\sigma,j} \tag{1.34}$$

$$= \sum_{k\sigma} b_{k\sigma,j} \frac{\partial}{\partial d_{k\sigma}} \frac{\partial V}{\partial d_{p\alpha}} \tag{1.35}$$

$$= \nabla_{\mathbf{b}_j} \left(\frac{\partial V}{\partial d_{p\alpha}} \right) \tag{1.36}$$

$$\approx \frac{\left(\frac{\partial V}{\partial d_{p\alpha}} \right)_{\mathbf{r}_0 + \delta_{\text{eff}}\mathbf{b}_j} - \left(\frac{\partial V}{\partial d_{p\alpha}} \right)_{\mathbf{r}_0 - \delta_{\text{eff}}\mathbf{b}_j}}{2\delta_{\text{eff}}}, \tag{1.37}$$

where $\nabla_{\mathbf{v}}(f)$ is the derivative of a function f in the direction of $\mathbf{v}$, and δ_{eff} is an effective displacement factor $\delta_{\text{eff}} = \delta / \|\mathbf{b}_j\|$ that accounts for the possibility of $\mathbf{b}_j$ not being normalized. (δ is set by #4 DispFactor.)

What (1.37) gives us is a formula for calculating directly a transformed basis vector $\boldsymbol{\sigma}_j$ *without* having to calculate explicitly the entire Hessian matrix, which would otherwise be necessary if using (1.34) directly.

Most electronic structure programs calculate the gradient of the potential energy surface at a point $\mathbf{r}$ at a rather modest computational cost compared to calculating the Hessian matrix. The Hessian transformation of a single vector using (1.37) only requires two *single-point* electronic structure calculations with gradients (at the points $\mathbf{r} = \mathbf{r}_0 \pm \delta_{\text{eff}}\mathbf{b}_j$) and is therefore cheaper than using (1.34). If calculating *all* the columns of $\boldsymbol{\Sigma}$ the cost will be the same though, but as we shall discuss below we may be able to do with transformations of only some of the basis vectors, if targeting only one or a few eigenvectors of $\mathbf{H}$.

Note also that (1.37) is a finite difference approximation. With the error scaling as $O(\delta_{\text{eff}}^2)$ we could expect an infinitesimally small displacement factor to give the most accurate result. Having said that, a too small value of δ_{eff} may in practical calculations, which only employ finite-precision arithmetics, give cause to large round-off errors if the value of the gradient is not substantially different in the two points evaluated. It can thus be expected that a compromise must be found between the displacement factor being too small and too large. The default value of #4 DispFactor has been found to generally work well for small and medium-sized molecules (tens of atoms).

Having established how to perform a Hessian transformation $\hat{H}(\mathbf{b})$ (without explicitly calculating the entire Hessian matrix), we can further move on to describe the #4 Targeting feature, which aims at finding the normal coordinate(s) that has/have the largest overlap with some predefined target coordinates. This is done in an iterative manner.

The scheme of an iterative eigenvalue equation solver is then to begin with a set of n_t initial (orthonormal) trial vectors $\{\mathbf{b}_j \mid j = 1, \ldots, k\}$ that are then transformed with $\hat{H}$ to yield $\{\boldsymbol{\sigma}_j \mid j = 1, \ldots, k\}$, where $k = n_t$ in this initial iteration, but in general will be used for denoting the dimension of the reduced space. We will use the notation

$$\mathbf{B}^{(k)} = (\mathbf{b}_1, \ldots, \mathbf{b}_k) \tag{1.38}$$

$$\boldsymbol{\Sigma}^{(k)} = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_k) \tag{1.39}$$

to denote the matrices with the respective vectors as columns. The *reduced k-dimensional Hessian* matrix is then the projection of $\boldsymbol{\Sigma}^{(k)}$ on the column space of $\mathbf{B}^{(k)}$,

$$\tilde{\mathbf{H}}^{(k)} = \mathbf{B}^{(k)\dagger} \boldsymbol{\Sigma}^{(k)}, \tag{1.40}$$

where the tilde is used to denote quantities pertaining to the reduced space. The reduced Hessian is then subsequently diagonalized;

$$\tilde{\mathbf{C}}^{(k)\dagger}\tilde{\mathbf{H}}^{(k)}\tilde{\mathbf{C}}^{(k)} = \tilde{\boldsymbol{\Lambda}}^{(k)} = \text{diag}(\tilde{\lambda}_1, \ldots, \tilde{\lambda}_k), \tag{1.41}$$

The above equation is for the right-hand eigenvectors. If the transformer/matrix is not symmetric then there is also an equivalent equation for the left-hand eigenvectors. This can be the case due to numerical inaccuracies in (1.37). By default the reduced Hessian is made symmetric, but this can be switched off by #4 NonSymmetricHessian.

When beginning the iterative solving procedure one has decided on some way to focus on a certain number of solutions to the eigenvalue problem. Various schemes exist, e.g. one could aim for the n_{eq} lowest-lying eigenvalues, or those closest to a certain predefined value. Here the criterion is to find those with the largest overlap with one or several predefined target vectors.

Regardless of the scheme the reduced eigenvectors are sorted in some well-defined way, and the n_{eq} most desirable ones are chosen for further study, $\{\tilde{\mathbf{c}}_1^{(k)}, \ldots, \tilde{\mathbf{c}}_{n_{\text{eq}}}^{(k)}\}$. If the columns of $\mathbf{B}^{(k)}$ span the full space, and $\tilde{\mathbf{c}}_j^{(k)}$ is an eigenvector of $\tilde{\mathbf{H}}^{(k)}$, then $\mathbf{c}_j^{(k)} = \mathbf{B}^{(k)}\tilde{\mathbf{c}}_j^{(k)}$ is an eigenvector of $\mathbf{H}$.

For knowing how close we are to actually having found a solution to the full space eigenvalue problem, the residual vector

$$\mathbf{r}_j^{(k)} = \mathbf{H}\mathbf{c}_j^{(k)} - \tilde{\lambda}_j\mathbf{c}_j^{(k)} = (\boldsymbol{\Sigma}^{(k)} - \tilde{\lambda}_j^{(k)}\mathbf{B}^{(k)})\tilde{\mathbf{c}}_j^{(k)}, \qquad j = 1, \ldots, n_{\text{eq}} \tag{1.42}$$

is formed for each of the chosen eigenvectors $\tilde{\mathbf{c}}_j^{(k)}$. The closer $\mathbf{r}_j^{(k)}$ is to being zero the closer $\mathbf{c}_j^{(k)}$ is to being an eigenvector of $\mathbf{H}$ in the full space. We can therefore use the norm of the residual as measure of how acceptable the solution is – the smaller the better. Another measure that is frequently used is the (relative) change in eigenvalue between successive iterations of the solving algorithm. If the (relative) eigenvalue change is small that also signifies that the algorithm has converged towards an actual eigenvalue. In short the following criteria must therefore be fulfilled in order for a solution to be accepted;

$$\epsilon_{\text{res}} \geq \left\|\mathbf{r}_j^{(k)}\right\|, \qquad \text{for } j = 1, \ldots, n_{\text{eq}} \tag{1.43}$$

$$\epsilon_{\text{energy}} \geq \left|\frac{\lambda_j^{(k)} - \lambda_j^{(k-n_{\text{eq}})}}{\lambda_j^{(k-n_{\text{eq}})}}\right|, \quad \text{for } j = 1, \ldots, n_{\text{eq}}, \tag{1.44}$$

where ϵ_{res} (#4 ItEqResThr) and ϵ_{energy} (#4 ItEqEnerThr) are predefined thresholds and $(k - n_{\text{eq}})$ signifies the values obtained in the previous iteration.

If all equations in (1.43) and (1.44) are fulfilled then solution is accepted and the iterative algorithm is terminated. If not, then the current subspace was apparently not sufficiently large for describing correctly the sought eigenvectors/-values. Thus, it must be expanded.

For this the residual vectors are chosen, since they already by definition point in the direction in the full space in which there is a discrepancy between the right- and left-hand side of the eigenvalue equation $\mathbf{H}\mathbf{c}_j^{(k)} = \tilde{\lambda}_j^{(k)}\mathbf{c}_j^{(k)}$. Mode targeting/tracking consists of specifying in advance a set of target vectors $\{\boldsymbol{\tau}_i \mid i = 1, \ldots, n_t\}$ for which the eigenvectors $\mathbf{c}_j$ with the greatest overlaps $\langle\mathbf{c}_j|\boldsymbol{\tau}_i\rangle$ are sought. The MidasCpp implementation more specifically proceeds with targeting as follows; in each iteration the overlap $s_{ij}^{(k)}$ between every target and full space eigenvector is calculated,

$$s_{ij}^{(k)} = \langle\boldsymbol{\tau}_i|\mathbf{c}_j^{(k)}\rangle = \langle\boldsymbol{\tau}_i|\mathbf{B}^{(k)}\tilde{\mathbf{c}}_j^{(k)}\rangle = \langle\tilde{\mathbf{o}}_i|\tilde{\mathbf{c}}_j^{(k)}\rangle, \tag{1.45}$$

where

$$\tilde{o}_{li} = \langle\mathbf{b}_l|\boldsymbol{\tau}_i\rangle \tag{1.46}$$

are the overlaps between the targets and the trial vectors. The overlap matrix $\tilde{\mathbf{O}}^{(k)}$ is of reduced dimensions $(k \times n_t)$, and since neither the targets or trial vectors change once they have been added, $\tilde{\mathbf{O}}^{(k)}$ can simply be stored and updated in each iteration. The calculation of the actual overlaps in the k'th iteration is then a simple manner of performing the matrix multiplication

$$\tilde{\mathbf{S}}^{(k)} = (\tilde{\mathbf{O}}^{(k)})^\dagger \tilde{\mathbf{C}}^{(k)}. \tag{1.47}$$

The overlap sum for each target in the k'th iteration, $S_i^{(k)}$, is defined as

$$S_i^{(k)} = \sum_{j=1}^{n_{\text{overlap},i}} \left(s_{ij}^{(k)} \right)^2. \tag{1.48}$$

If $n_{\text{overlap},i} = k$ and the target $\boldsymbol{\tau}_i$ is spanned by the set of trial vectors (which is usually the case, since the targets are used as the initial trial vectors, at least in this implementation), then $S_i^{(k)} = 1$. However, not all terms will be equally large, with some of them probably being substantially greater than others, and for expanding the subspace further we are only interested in those that highly contribute to the overlap. The idea of the "overlap sum" strategy is then the following;

- for each target $\boldsymbol{\tau}_i$, sort the the eigenvectors $\tilde{\mathbf{c}}_j^{(k)}$ so that $s_{i1}^{(k)}, s_{i2}^{(k)}, \ldots, s_{ik}^{(k)}$ is a decreasing sequence.

- set $n_{\text{overlap},i}$ to the be the lowest integer that makes the overlap sum larger than a predefined threshold, $S_i^{(k)} \geq \epsilon_{\text{overlap}}$ (#4 OverlapSumMin).

- Then use the residuals of those eigenvectors when expanding the subspace with new trial vectors.

The above procedure applies to all of the targets given, but a given eigenvector should of course only be included once in the set from which the new trial vectors are made. But any given eigenvector is still allowed to enter the overlap sum for several targets. A further modification that has been implemented in `MidasCpp` is that one can set an upper bound, $n_{\text{overlap,max}}$ (#4 OverlapNMax), to $n_{\text{overlap},i}$, thus allowing for greater control of how many trial vectors are added to the subspace in each iteration. Note that in the limiting case that one sets $n_{\text{overlap,max}} = 1$ and $\epsilon_{\text{overlap}} = 1$ the result will be the single eigenvector that overlaps best with each target.

1.4 Optimized coordinates

There can be significant gain in the use of optimized coordinates, that is coordinates what are optimized by some kind of variational procedure. With the right set of coordinates the PES construction may require fewer points, and the anharmonic wave function may be computed faster. Furthermore, for interpretation purposes optimized or "quasi-local" coordinates may be convenient for interpretation also.

1.4.1 Optimized Coordinates

The computation of optimized coordinates relies on the ability to efficiently compute the VSCF energy. Following past work optimized coordinates have been implemented in MidasCpp. [2, 3]

> **Consider to cite:**
>
> 1. B. Thomsen, K. Yagi, O. Christiansen, "Optimized coordinates in vibrational coupled cluster calculations", *Journal of Chemical Physics* **2014**, *140*, 154102–124101–15
>
> 2. B. Thomsen, K. Yagi, O. Christiansen, "A simple state-average procedure determining optimal coordinates for anharmonic vibrational calculations", *Chemical Physics Letters* **2014**, *610–611*, 288–297

1.4.2 On PES forms appropriate for Optimization of Coordinates

It should be noted, that the exact energy is invariant under the choice of coordinates. However, the quality of approximate energies and wave functions will depend on the choice of coordinates. Thus, the optimization is a way to improve approximate treatments by providing better convergence to the exact results, and to obtain wave functions that in more concise manners represent the important physical effects. It is important that the Hamiltonian is in a form that is invariant under the transformation. Obviously the exact Hamiltonian is invariant, but approximate potentials are not necessarily invariant. One example that is invariant is a Taylor expanded force field such as a quartic force field, motivating this choice in the particular examples. An example that is not invariant is a many-mode expansion of the potential truncated to an order less than total number of modes.

1.4.3 State-average Optimized Coordinates

The outcome of the coordinate optimization procedure depends on the energy quantity to be optimized, and thereby the state. Typically the VSCF ground state energy is used, making the full procedure variational. A generalization aiming at including more than one state in the optimization is described in Ref.[3]

> **Consider to cite:**
>
> 1. B. Thomsen, K. Yagi, O. Christiansen, "A simple state-average procedure determining optimal coordinates for anharmonic vibrational calculations", *Chemical Physics Letters* **2014**, *610–611*, 288–297

1.4.4 Hybrid Optimized and Localized Coordinates

Another possible set of coordinates are hybrid optimized and localized coordinates as described in Ref.[4]. This set of coordinates is as optimized coordinates obtained from normal coordinates through an extra orthogonal transformation. In this case it is a hybrid measure that is optimized, including both the energy and a measure of locality.

> **Consider to cite:**
>
> 1. E. L. Klinting, C. König, O. Christiansen, "Hybrid Optimized and Localized Vibrational Coordinates", *The Journal of Physical Chemistry A* **Nov. 2015**, *119*, 11007–11021

1.5 FALCON: Flexible Adaptation of Local Coordinates Of Nuclei

Another algorithm for construction of localized coordinates is Falcon. Its distinct feature lies within an iterative procedure, which allows to generate purely vibrational rectilinear coordinates with well-defined locality at desirable parts of the molecule. The approach starts from small groups of atoms and iteratively fuses them into larger groups projecting out translational and rotational degrees of freedom at each step. The procedure is controlled by coupling estimates between such fusion groups and can be run in a black-box manner. The resulting coordinates can be divided into two types: i) fully localized at parts of the molecule and ii) inter-connecting. The latter are constructed from local degrees of freedom and describe relative vibrations of the fused groups. The coordinates are then relaxed by diagonalizing the corresponding reduced mass-weighted Hessian matrices keeping a local character at the fusion groups. The use of these strictly local coordinates is especially beneficial with incremental-type expansions of PESs (for details, see Sec. 2.3.7).

> **Consider to cite:**
>
> 1. C. König, M. B. Hansen, I. H. Godtliebsen, O. Christiansen, "FALCON: A method for flexible adaptation of local coordinates of nuclei", *Journal of Chemical Physics* **2016**, *144*, 074108

Chapter 2

Potential energy surfaces

2.1 Potential energy surfaces - general aspects

Given the Born-Oppenheimer approximation the concept of a potential energy surface is clear. From solving the electronic Schrödinger equation using fixed nuclei that provides an attractive electrostatic field for the electrons, we obtain an electronic energy (including the nuclear-nuclear repulsion energy by convention) that depends on the relative position of the nuclei. This energy can in the Born-Oppenheimer approximation be considered as a potential in which the nuclei movies. Thus, the features of the relevant potential energy surfaces dictate the dynamics of the system. A theoretical understanding of the dynamics of a molecular system therefore relates to the potential energy surface .

The solution of the electronic Schrödinger equation to obtain the potential energy surface is a topic of electronic structure theory. Advanced quantum mechanical theories for the electronic structure of molecules in the gas phase have been developed over the years. They have been implemented in efficient computer programs that are widely distributed and used by specialists and non-specialists. Using such programs one can obtain reliable predictions of many different types of molecular properties including molecular structures. However, the actual calculation of dynamics is another issue as is the construction of potential energy surfaces.

The solution of the electronic Schrödinger equation is not discussed further here. The finer details of the electronic structure are unimportant for the discussion here, and, first of all, treated in other texts and courses. It is rather assumed that theories and computer programs are available that can provide an approximate (but not to approximate) solution to the electronic problem for a given structure. To construct and use such theories and programs is in no way trivial, and any error made in this step penetrates to the prediction of the dynamics. However, we focus on characteristics of potential energy surfaces, what are the problem in knowing them, and how we can actually extract some important information about the potential energy surface without knowing it!

The PES is needed for discussing motion on the PES. Some basic QM wave function models are assumed later. Also in this chapter a basic classical mechanic view can sometimes be very helpful. Anyhow, motion on a PES is discussed in a later chapter.

In figure 2.1 a model potential energy surface as function of two variables is given. Throughout we shall return to this figure and explain various features.

Multi-dimensional potential energy surfaces
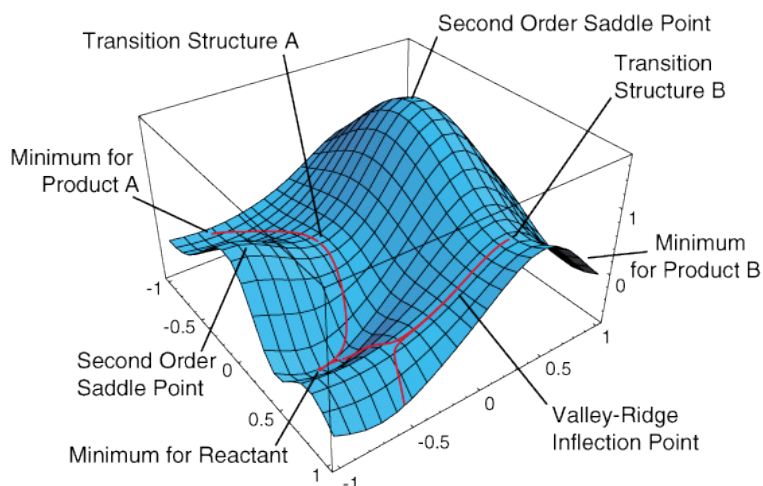
Figure 2.1: A model potential energy surface illustrating important stationary points and curves. (from Jack Simons:Introduction to theoretical chemistry)

2.2 The potential energy surface as a function of many variable

The potential energy surface gives meaning to several different concepts including molecular structure and vibrations. In this section, we consider some properties of a single PES. We shall not be concerned with electronic states that are degenerate because of high symmetry of the system, but focus on more general aspects.

Some basic features of a PES are

- The potential is determined from the time-independent electronic Schrödinger equation.

$$\hat{H}^{el}\Phi_I^{el}(\mathbf{r};\mathbf{R}) = E_I^{el}(\mathbf{R})\Phi_I^{el}(\mathbf{r};\mathbf{R}) \tag{2.1}$$

$$\hat{H}^{el} = \hat{T}_e + \hat{V}_{ee} + \hat{V}_{en} + \hat{V}_{nn} \tag{2.2}$$

- We can think of the potential as being a scalar function of the nuclear coordinates i.e., $V = E_I^{el}(\mathbf{R})$. For each value of the nuclear coordinates $E_I^{el}(\mathbf{R})$ has one value for each electronic state.

- Each electronic state has its own potential energy surface

- We can think of the potential energy as a function of Cartesian coordinates. However, it is often preferable to think of it as a function of other coordinates that describe the internal arrangement of the atoms. Thus, we can generally think of the PES as a function of some coordinates q_i that can be internal coordinates like bond-lengths, angles etc, normal

coordinates (see also later), or some other set of coordinates. If there are N atoms it is known that it takes three degrees of freedom to describe overall translation and three to describe overall rotation (two for a linear molecule). This leaves us with $M = 3N - 6(5)$ coordinates describing the internal degrees of freedom.

- $V(q_1, q_2, ...q_M)$ is called the potential energy surface. In reality it is never a surface since this correspond to $M = 2$. If $N = 2$ we have a linear molecule and $M = 1$ and we have that the potential energy "surface" is really a function of only one variable. Thus, it is rather a function or a curve. The electronic energy is a 3N-6(5) dimensional hyper-surface in a space of 3N-5 dimensions. We can only visualize a 2D-surface in 3D-space - but must remember that the full coordinate space is of dimension 3N-6(5). So what we can look at is only cuts, meaning that some coordinates have a prescribed value, while others are varying freely.

It is generally impossible to know the PES. First, we recall that we cannot solve the electronic Schrödinger equation exactly (except for Hydrogenic atoms and ions). While there are many approximate electronic structure methods it should be understood that they do not define an analytical expression for $V(q_1, q_2, ...q_M)$ but only a theory and a computational method for calculating individual points of the PES. Thus, we can know the values for a given set of structures, but not the function itself.

Having at our hand theories and computational methods for carrying out electronic structure calculation for a given structure, cant we just calculate the potential on a grid of points and thereby generate the potential numerically? For example, we introduce for each coordinate a discrete set of G points with values $q_i = q_{i,1}, q_{i,2}, ...q_{i,G}$. While calculating the potential for some points is our only way to knowing something about the potential, it is not a realistic approach in its simplest variant for larger molecules with more than four atoms. If we have M dimensions, then a simple grid with G points in each dimension will give G^M points, since the M variables must be varied independently. For ten points per degree of freedom corresponding to $G = 10$ (which is a realistic but modest number) we easily see that even for a five atomic nonlinear molecule with M=3*5-6=9 this gives us 10^9 points. A single electronic structure calculation takes of order a second to say months of CPU time depending on accuracy, molecular size, etc. Clearly, such a simple direct-product will generally not be a realistic approach.

Thus, our working condition in interpreting molecular structure and dynamics is that we have put a function, the PES, in a central role, even if we don't know it and, from the complexity point of view, it is a quantity that we cannot really expect to know fully for polyatomic molecules. We shall now demonstrate that one can in some automatic ways navigate potential energy surfaces and extract some essential aspects. More generally, one can define some ways of focusing the PES construction on the most important part.

2.3   Approximate representation of potential energy surfaces

In this section we consider various representations of the potential energy operator for internal motion. Out focus is in first hand on the vibrational motion, e.i. bound states. We describe restricted mode representations as well as Taylor-expansions in the normal coordinate representations and discuss pros and cons. Our focus is on the potential. The very same principles can be used for other properties, such as for example the dipole-moment surfaces relevant for calculation of transition probabilities for molecules interacting with light.

The potential for the M normal coordinates is a multiplicative operator and in general a function

of *all M* normal coordinates simultaneously

$$\hat{V} = V(q_1, q_2, ..., q_M) \tag{2.3}$$

However, the inclusion of full coupling between all modes becomes prohibitively complicated when the number of modes become large. Furthermore, not all couplings are equally important. Analysis, simplification, and approximations are thus in general needed to proceed. We therefore in the following try to investigate different routes for doing so.

2.3.1 Taylor expansion of the potential

A rather obvious idea for the representation of the potential is to expand the potential around some minimum using ordinary Taylor expansions and truncate this expansion at some order N_t

$$V(\mathbf{q}) = V(\mathbf{0}) + \sum_{n=1}^{N_t} \frac{1}{n!} \sum_{i_1 i_2, ... i_n} \frac{\partial^n V}{\partial q_{i_1} \partial q_{i_2} ... \partial q_{i_n}} q_{i_1} q_{i_2} ... q_{i_n} \tag{2.4}$$

Consider now matrix elements of basis functions in the $M-$mode space that are in a direct product form, also called a Hartree-product. Thus we assume

$$|\Phi_{\mathbf{r}}\rangle = \prod_{m=1}^{M} \phi_{r^m}^m(q_m) \tag{2.5}$$

where the different one-mode functions $\phi_{r^m}^m(q_m)$ are assumed orthonormal. Matrix elements over the Taylor expanded potential becomes

$$
\begin{aligned}
\langle \Phi_{\mathbf{r}}|V|\Phi_{\mathbf{s}}\rangle &= V(\mathbf{0})\langle \Phi_{\mathbf{r}}|\Phi_{\mathbf{s}}\rangle + \sum_{n=1}^{N_t} \frac{1}{n!} \sum_{i_1 i_2, ... i_n} \frac{\partial^n V}{\partial q_{i_1} \partial q_{i_2} ... \partial q_{i_n}} \langle \Phi_{\mathbf{r}}|q_{i_1} q_{i_2} ... q_{i_n}|\Phi_{\mathbf{s}}\rangle \\
&= V(\mathbf{0})\delta_{\mathbf{rs}} + \sum_{n=1}^{N_t} \frac{1}{n!} \sum_{i_1 i_2, ... i_n} \frac{\partial^n V}{\partial q_{i_1} \partial q_{i_2} ... \partial q_{i_n}} \prod_{m=1}^{M} \langle \phi_{r^m}^m(q_m)|q_m^{\sum_{j=1}^{n} \delta_{i_j m}}|\phi_{s^m}^m(q_m)\rangle
\end{aligned}
\tag{2.6}
$$

The attractive feature with such potentials is clearly that the multidimensional integrals factorize into products of one-dimensional integrals. One-dimensional integrals is simple to evaluate in numerical calculations, should analytical expressions not be available.

The problem with Taylor expansions is that they cannot in general be expected to be convergent (most likely the contrary). A Taylor expansion represent the view on a PES as seen from a single expansion point and its derivatives. Thus, the expansion may be less well-motivated if we for example seek an accurate description away from equilibrium, or the molecular motions are far away from equilibrium. Thus, a one-center Taylor expansion is fully unsatisfactory for studying reactions. For molecular vibrations Taylor expansion the case is seemingly more favorable. Thus, if the molecular is fairly "rigid" meaning in other words that the second derivative terms of the Harmonic oscillator treatment covers most of the motions and the distortion away from equilibrium is after all minor, the Taylor expansion seemingly have a case. Therefore it has been successfully applied in perturbation theoretical improvements on the normal-coordinate harmonic oscillator treatment of molecules. However, for more general (higher) vibrational states and less rigid systems the Taylor expansion is less satisfactory. In fact, for explicit wave function calculations of the nuclear motion with higher and systematic accuracy, the Taylor expansion should be used with some caution. Away from the expansion point the Taylor expansion is unbound and it is typical that the potential contains artificial "holes" where a variational calculation will go astray. Thus the wave functions for nuclear motion will, artificially, tunnel through such artificial walls and fall into artificial holes and give nonsense results.

Generalized product form of the Hamiltonian

The simplifying feature of the Taylor expanded Hamiltonian comes from that it a sum over products. A general sum-over-product (SOP) Hamiltonian can be written as a sum over P product terms.

$$\hat{H} = \sum_{p=1}^{P} c_p \prod_{m=1}^{M} \hat{h}_m^p \tag{2.7}$$

giving a similar simple structure of the matrix elements.

$$\langle \Phi_{\mathbf{r}} | \hat{H} | \Phi_{\mathbf{s}} \rangle = \sum_{p=1}^{P} c_p \prod_{m=1}^{M} \langle \phi_{r^m}^m(q_m) | \hat{h}_m^p | \phi_{s^m}^m(q_m) \rangle \tag{2.8}$$

In general, the sum over product form has some advantages and disadvantages. The most obvious disadvantage is that it is clear the potential does not come out in this particular form from solving the electronic structure problem for different arrangements of the nuclei. There is as such not much indicating that it should be particularly attractive from that perspective. However, one can turn this around. It is seemingly so that we will never obtain an accurate analytical form for the operator. We will have to do with various intermediate realizations. They can be based on defining the potential through its values on a grid of points directly. Or it could include the sum over product form in Eq. 2.7 as an intermediate numerical step. As an intermediate numerical step one can argue that 1) there is lacking knowledge of a convenient form that in general will be significantly better, 2) the form includes the simpler Taylor expansion as special case; and 3) it is computationally a very advantageous form for the operator. In particular, the product form of the matrix element over Hartree Products given in Eq.2.8 is crucial for efficient wave function calculations.

2.3.2 Restricted mode coupling

We now try to define a sequence of approximate potentials providing a more and more accurate description of the full coupling by including more and more mode-couplings. We will thus in the simplest approximation include only single mode terms in the potential, in the next approximation only single and pair-mode terms, and so on, leading to a set of approximations denoted $V^{(1)}$, $V^{(2)}$, ..., $V^{(M)}$.

For this purpose we define the sequence of potentials where only one, two, etc up to M coordinates are different from zero

$$V^{m_1} = V(0, ..., 0, q_{m_1}, 0, ..., 0) \tag{2.9}$$

$$V^{m_1 m_2} = V(0, ..., 0, q_{m_1}, 0, ..., 0, q_{m_2}, 0, ..., 0) \tag{2.10}$$

etc. The mode number indices m_i runs from mode 1 to mode M. However, we require that $m_1 \neq m_2$ and so on. The last potential in this sequence

$$V^{m_1 m_2, ... m_M} = V(\mathbf{q}) \tag{2.11}$$

is equivalent to the exact potential since all indices must be different. We will generally denote these potentials as $V^{\mathbf{m}_n}$ where $\mathbf{m}_n$ is an n-dimensional vector of indices $m_1, m_2, ..., m_n$. A set of mode numbers, $\mathbf{m} = \{m_0, m_1, ...\}$ will in the following be denoted a mode combination (MC). Note that the $V^{\mathbf{m}_n}$ are symmetric with respect to permutation of the m_i indices and accompanying coordinates. For example

$$V^{m_1 m_2}(q_{m_1}, q_{m_2}) = V^{m_2 m_1}(q_{m_2}, q_{m_1}) \tag{2.12}$$

These potentials are individual contributions that can be combined into the desired sequence of potentials. By summing over all modes, for example summing V^{m_1, m_2} over all pairs (m_1, m_2), we ensure that all modes are treated equivalently. However, we thereby also introduce over-countings. For example, for the sum over all mode pairs we have when one of the coordinates is zero

$$\sum_{m_1 < m_2 = 1}^{M} V^{m_1 m_2}(q_{m_1}, q_{m_2} = 0) = \sum_{m_1} V^{m_1}(M - m_1) \tag{2.13}$$

A summation over $\mathbf{m}_n$ denoted with a double prime refers to a restricted summation over the m indices

$$\sum_{\mathbf{m}_n}'' = \sum_{m_1} \sum_{m_2 > m_1} \dots \sum_{m_n > m_{n-1}} \tag{2.14}$$

while a single prime denotes not-equal-to summation convention

$$\sum_{\mathbf{m}_n}' = \sum_{m_1} \sum_{m_2 \neq m_1} \dots \sum_{m_n \neq \{m_{n-1}, m_{n-2}, \dots, m_1\}} \tag{2.15}$$

The over-counting problem originates from the fact that if one or more of the coordinates in $V^{\mathbf{m}_n}$ is zero on obtains the same result as a lower mode-combination. For example $V^{\mathbf{m}_n}(q_{m_n} = 0) = V^{\mathbf{m}_{n-1}}$. Therefore, we consider a set of potentials $\bar{V}^{\mathbf{m}_n}$ closely related to $V^{\mathbf{m}_n}$ that per definition satisfies that they give zero if any coordinates are zero

$$\bar{V}^{\mathbf{m}_n}(\dots, q_i = 0, \dots) = 0 \tag{2.16}$$

For example,

$$\bar{V}^{m_1}(0) = 0 \tag{2.17}$$

$$\bar{V}^{m_1 m_2}(0, q_{m_2}) = \bar{V}^{m_1 m_2}(q_{m_1}, 0) = 0 \tag{2.18}$$

The $\bar{V}^{\mathbf{m}_n}$ is related to $V^{\mathbf{m}_n}$ such that they separate out the term where redundancies are present. That is, we require

$$V^{\mathbf{m}_n} = \bar{V}^{\mathbf{m}_n} + \sum_{n'=1}^{n-1} \sum_{\mathbf{m}_{n'}} C^{\mathbf{m}_{n'}} \bar{V}^{\mathbf{m}_{n'}} \tag{2.19}$$

where the indices in $\mathbf{m}_{n'}$ is restricted to those present in $\mathbf{m}_n$. The $C^{\mathbf{m}_{n'}}$ coefficients are determined by the conditions in Eq.(2.16). We obtain in this way

$$V^{m_1} = \bar{V}^{m_1} \tag{2.20}$$

$$V^{m_1 m_2} = \bar{V}^{m_1 m_2} + \bar{V}^{m_1} + \bar{V}^{m_2} = S^{m_1 m_2}(\bar{V}^{m_1 m_2} + 2\bar{V}^{m_1}) \tag{2.21}$$

$$V^{m_1 m_2 m_3} = \bar{V}^{m_1 m_2 m_3} + \bar{V}^{m_1 m_2} + \bar{V}^{m_1 m_3} + \bar{V}^{m_2 m_3} + \bar{V}^{m_1} + \bar{V}^{m_2} + \bar{V}^{m_3}$$
$$= S^{m_1 m_2 m_3}(\bar{V}^{m_1 m_2 m_3} + 3\bar{V}^{m_1 m_2} + 3\bar{V}^{m_1}) \tag{2.22}$$

Here $S^{\mathbf{m}_n}$ is an operator that symmetrizes with respect the n m-indices. For example

$$S^{m_1 m_2} F(m_1, m_2) = \frac{1}{2}(F(m_1, m_2) + F(m_2, m_1)) \tag{2.23}$$

In general

$$V^{\mathbf{m}_n} = S^{\mathbf{m}_n} \sum_{n'=1}^{n} \binom{n}{n'} \bar{V}^{\mathbf{m}_{n'}} \tag{2.24}$$

Eq.(2.24) can be seen in the following way: For a given n $V^{\mathbf{m}_n}$ can contain contributions from all $\bar{V}^{\mathbf{m}_{n'}}$ where $n' = 1, 2, \dots, n$. The $n' = n$ term is obvious. For a given n' contribution we have in $\bar{V}^{\mathbf{m}_{n'}}$ n' coordinates that are non-zero and $n - n'$ coordinates that are zero. The n' non-zero coordinates

can be selected in $\begin{pmatrix} n \\ n' \end{pmatrix}$ ways. The operator $S^{\mathbf{m}_n}$ generates the different permutations of indices necessary. The examples above exemplify the formulae.

One can prove that

$$\bar{V}^{\mathbf{m}_n} = S^{\mathbf{m}_n} \sum_{n'=1}^{n} (-1)^{n-n'} \begin{pmatrix} n \\ n' \end{pmatrix} V^{\mathbf{m}_{n'}} \tag{2.25}$$

From the $\bar{V}^{\mathbf{m}_n}$ we can define the potential functions

$$\Delta V^{(n)} = \sum_{\mathbf{m}_n}{}'' \bar{V}^{\mathbf{m}_n} \tag{2.26}$$

We have for example

$$\Delta V^{(1)} = \sum_{m_1} \bar{V}^{m_1} \tag{2.27}$$

$$\Delta V^{(2)} = \sum_{m_1} \sum_{m_2 > m_1} \bar{V}^{m_1 m_2} \tag{2.28}$$

The terms $\bar{V}^{\mathbf{m}_n}$ depends on n coordinates and are per definition taken as a linear combination of the $V^{\mathbf{m}_p}$ potentials of Eqs.(2.9-2.11) for $p \leq n$. From the definition of $\bar{V}^{\mathbf{m}_n}$ the potential $\Delta V^{(n)}$ is zero unless n and exactly n coordinates are different from zero. We can thus obtain the k-mode coupling representation of the potential as

$$V^{(k)} = \sum_{n=1}^{k} \Delta V^{(n)} \tag{2.29}$$

In this manner we have defined the sequence of potentials

$$V^{(1)}, V^{(2)}, V^{(3)}, ..., V^{(M)} \tag{2.30}$$

which is guaranteed to converge to the exact potential. The $V^{\mathbf{m}_n}$ terms can - due to the over-counting of the lower mode couplings - not be expected to become smaller with increasing n. Yet, in many ways, the physical higher mode couplings can be expected to be smaller compared to lower-mode couplings. This physical situations is thus not directly revealed in this expression for $V^{(k)}$. However, the potentials $\Delta V^{(k)}$

$$\Delta V^{(1)} = V^{(1)} \tag{2.31}$$

$$\Delta V^{(k)} = V^{(k)} - V^{(k-1)} \text{ for } k > 1 \tag{2.32}$$

will for a given k provide the new effect included from the k-mode coupling and only k-mode coupling.

In practical terms the useful aspect is that the construction of $V^{(k)}$ can proceed through Eq.(2.26). We can thus generally write the potential as a sum over contribution from different MCs where for each MC the contribution is in the form of a bar potential, that in turn is obtained from the appropriate linear combination of "cuts" of the original potential with only some coordinates non-zero. A specific approximation or representation of the potential is therefore defined by exactly which MCs are included.

We may talk about the mode-combination-range (MCR) of the potential, meaning the set of MCs which are included in the potential,

$$V = \sum_{\mathbf{m} \in \text{MCR}[V]} \bar{V}^{\mathbf{m}} \tag{2.33}$$

This rewrite shows that the formulation is flexible and general with respect to inclusion of MCs. There is no double prime in the sum as the sum over MCs in the MCR is assumed to include the relevant MCs one and only one time. In many contexts the MCR will be defined to the full set of one and two-mode couplings, or in general up to n-mode couplings. However, one can in principle include and leave out couplings in a flexible way using the bar potentials.

Interpolation and fitting procedures

The restricted mode representation of the Hamiltonian is not directly of a sum-over-products form. Fitting to a direct product of one-mode functions implicitly casts the vibrational Hamiltonian to a sum-over-products form,

$$\hat{H} = \sum_{t=1}^{T} c_t \prod_{m=1}^{M} \hat{h}^{m,t} \tag{2.34}$$

The kinetic energy operator will be discussed in the next subsection. For the potential part the overall idea is that in the absence of an analytical form we will have to either use a grid approach or an intermediate analytical representation. This intermediate analytical representation might as well be the convenient direct product form. This can be done with fitting. Here it is important to appreciate that the MC expansion allows us to fit small pieces at a time, as opposed to fitting the full PES.

To be concrete a straightforward approach is to perform a n-dimensional linear least-squares fits in a direct product polynomial basis of the $\bar{V}^{\mathbf{m}_n}$ functions. For numerical reasons often the scaled normal coordinates are used $y_i = \sqrt{\omega_i} Q_i$, reducing round off errors when fitting to high-degree polynomials. The size of the direct-product basis grows exponentially with the mode-coupling level n of the particular $\bar{V}^{\mathbf{m}_n}$ potential. This affects the computational efficiency of the vibrational structure calculations as the number of terms in the potential increase similarly. This can be limited slightly by using cut-offs to limit the total degree of the polynomial basis set used in the fitting procedure. Thus, in the fit of the form:

$$\bar{V}^{\mathbf{m}_n} = \sum_{i_1,i_2,\ldots,i_n} c_{i_1,i_2,\ldots,i_n} y_{m_1}^{i_1} \cdots y_{m_n}^{i_n} \tag{2.35}$$

the summation can be restricted with $i_k \leq N_k, (k = 1, \ldots, n)$ as the only restriction or with the additional constraint $\sum_k i_k \leq N$.

The set of points fitted can be a set of numbers from original electronic structure calculations. Or alternatively it can be a set of points obtained from an interpolating function constructed from a smaller set of points or from a smaller set of points supplying also derivatives of the energy. The set of points which are used as input date for the fit is typically also a direct product grid.

We may in different context choose to emphasize different aspects of the sum of products of one-mode operators. E.g. we can write the operator as

$$H = \sum_{n} \sum_{m_0 < m_1 < \ldots < m_{n-1}} \sum_{o_1,o_2,\ldots,o_{n-1}} c_{m_0 m_1 \cdots m_{n-1}}^{o_0 o_1 \cdots o_{n-1}} \, h^{m_0 o_0} \, h^{m_1 o_1} \, \cdots \, h^{m_{n-1} o_{n-1}}, \tag{2.36}$$

Approximate operators can be described by setting the appropriate coefficients to zero. Here $h^{m_i o_i}$ is a one-mode operator for mode m_i of type o_i. Thus we imagine that for each mode there is a restricted set of types of operators. The gain here is that in wave function calculations one only has to address the action of a limited number of one-mode operators to the wave function. In other word, the catch is that in the fitting of all mode-couplings it can be advantageous to have a common and limited basis of one-mode operators, like e.g. the operators q_m^i.

2.3.3 Building the potential through its values on a grid

Consider as a first step a grid approach where one discretizes each q_m coordinate using g single points. In this case $g^M = \exp(M ln(g))$ points are needed to map out the full M-mode PES. The exponential increase in the number of points with increasing M makes this presently unrealistic for M larger than 6, making larger than 4-atomic molecules out of reach in full dimensionality.

Nevertheless, one would like to be able to construct potential energy surfaces and generally molecular property surfaces in a cost-efficient and hopefully sufficiently accurate manner. In the restricted mode-coupling approach of the previous subsections each mode coupling will be represented as a direct product grid of points, however each mode-coupling is of (much) lower dimensionality than M. The number of required points will be dominated by the computation of the highest n-mode couplings included in the molecular PES

$$N_{points} = \sum_{k=1}^{n} \left(\begin{array}{c} M \\ k \end{array} \right) g_k^k \tag{2.37}$$

Here g_k is the number of single points per direction in the generation of the direct grid at level mode-coupling level k. The g_k^k factor can be significant but the M scaling is determined by the $\left(\begin{array}{c} M \\ k \end{array} \right)$ factor being roughly of order M^k for $k << M$. Clearly, bringing the worst scaling down to low order polynomial ($M^n g_n^n$) is a tremendous improvement compared to g^M. The choice of n and the effective g_n are now the rate-determining factors.

2.3.4   Midas PES construction methods

MidasCpp allows the construction of potential energies and property surfaces based on various methods that are based on the general aspects above.

2.3.5   Taylor expansion

Following section 2.3.1 MidasCpp can employ arbitrary order Taylor expansions to compute potential energy and property surfaces as described in Ref. [6]

Users should be aware that the due to its limited radius of trust the use of Taylor expanded PES can give quite dramatic failures when combined with advanced vibrational wave functions. The advanced wave function can simply detect "holes" and nonphysical cornes of the Taylor potential giving non-physical results. On the other hand, for some molecular properties a low order Taylor expansion may be fully adequate.

> **Consider to cite:**
>
> 1. J. Kongsted, O. Christiansen, "Automatic generation of force fields and property surfaces for use in variational vibrational calculations of anharmonic vibrational energies and zero-point vibrational averaged properties", *J. Chem. Phys.* **2006**, *125*, 124108–16

2.3.6   Static grids expansion

With the framework of the mode-coupling expansion/n-mode expansion of the previous sections one can manually define the grid and manually adjust grid sizes and densities in different mode-coupling levels as well as employ interpolation approach. This so called static grid implementation is described in Ref.[7] and for use of energy derivatives, in Ref.[8].

Consider to cite:

1. D. Toffoli, J. Kongsted, O. Christiansen, "Automatic generation of potential energy and property surfaces of polyatomic molecules in normal coordinates", *J. Chem. Phys.* **2007**, *127*, 204106–14

2. E. Matito, D. Toffoli, O. Christiansen, "A hierarchy of potential energy surfaces constructed from energies and energy derivatives calculated on grids", *J. Chem. Phys.* **2009**, *130*, 1341041–13

2.3.7   Double incremental expansion

The double incremental expansion exploits ideas of restricting mode–mode couplings (as described in Sec. 2.3.2) and fragmentation of the molecule into subsystems. Therefore, this approach addresses both scaling problems: i) an increasing dimensionality of PESs with the system size and ii) a scaling of single-point electronic structure calculations. Being applied together with localized Falcon coordinates (for details, see Sec. 1.5), it significantly reduces the overall computational cost giving rise to the so-called double incremental expansion in Falcon coordinates (DIF) approximation [9]. A further reduction of the computational cost is possible if remaining non-local coordinates are represented approximately by an auxiliary set of coordinates with a purely local character. This can be done using the double incremental expansion in FALCON coordinates with auxiliary coordinate transformation (DIFACT) approach [9].

In order to derive a doubly-incremented form of the potential energy surface, we first need to consider the total energy of the molecular system E. Assuming that the supersystem is divided into N subsystems, the expression for the energy can be written as

$$E = E(z_1, z_2, ..., z_N), \tag{2.38}$$

where z_i represents particular conformations of subsystems. In these notations, $z_i = 0$ means that no atoms of the subsystem i are present in the considered conformation. In a similar way as was done in Sec. 2.3.2, we can define a sequence of functions

$$E_{f_1} = E(0, ..., 0, z_{f_1}, 0, ..., 0), \tag{2.39}$$

$$E_{f_1 f_2} = E(0, ..., 0, z_{f_1}, 0, ..., 0, z_{f_2}, 0, ..., 0), \tag{2.40}$$

etc. Here, the former expression corresponds to the total energy of a single subsystem f_1, while the latter is the total energy of two interacting subsystems f_1 and f_2. We set $E(0, ..., 0)$ to be equal to zero. Then, in a full analogy to Eq. (2.44), the expression for the total supersystem energy E can be written as

$$E = \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \underline{E}_{\mathbf{f}_l}, \tag{2.41}$$

where $\mathbf{f}_l$ is a fragment combination (FC) from a fragment combination range (FCR). Note that the equality holds only if the summation runs over all possible FCs of the full FCR.

The relation between the total potential $V(\mathbf{q})$ and the total energy $E(\mathbf{z})$ is given by

$$V(\mathbf{q}) = E(\mathbf{z}(\mathbf{q})) - E(\mathbf{r}_0), \tag{2.42}$$

where $E(\mathbf{r}_0)$ is the energy of the reference structure. By expanding both terms at the right-hand side of Eq. (2.42), we obtain

$$V(\mathbf{q}) = \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \underline{E}_{\mathbf{f}_l} - \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \underline{E}_{\mathbf{f}_l}(\mathbf{r}_{0, \mathbf{f}_l}) = \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \Delta \underline{E}_{\mathbf{f}_l}. \tag{2.43}$$

Here, $\mathbf{r}_{0,\mathbf{f}_l}$ is the reference conformation of FCs $\mathbf{f}_l$. Finally, expanding in a similar fashion a sub-potential $V^{\mathbf{m}_{n'}}$ from Eq (2.25) and inserting the resulting expression into Eq. (2.44), one obtaines the doubly-incremented form of the total potential,

$$V = S^{\mathbf{m}_n} \sum_{n'=1}^{n} (-1)^{n-n'} \begin{pmatrix} n \\ n' \end{pmatrix} \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \Delta \underline{E}_{\mathbf{f}_l}^{\mathbf{m}_{n'}}. \tag{2.44}$$

> **Consider to cite:**
>
> 1. C. König, O. Christiansen, "Linear-scaling generation of potential energy surfaces using a double incremental expansion", *J. Chem. Phys.* **2016**, *145*, 064105

2.3.8 The Adaptive density guided approach

The Adaptive density guided approach (ADGA) is an iterative technique to building the PES and property surfaces.

The advantage is that much manual work is avoided and instead a few thresholds is introduced. These are based on converging the product of the potential times wave function density to stability in intervals.[10] Extensions allows for multilevel approaches where electronic structure information at different levels are used at different levels of mode combination. [11]. The most recent update to ADGA allows any fit basis to be used and various approaches to finding the grid boundaries, see Ref. [12].

> **Consider to cite:**
>
> 1. M. Sparta, D. Toffoli, O. Christiansen, "An Adaptive Density-Guided Approach for the generation of potential energy surfaces of polyatomic molecules", *Theor. Chem. Acc.* **2009**, *123*, 413 –429
>
> 2. M. Sparta, D. T. I.-M. Høyvik, O. Christiansen, "Potential energy surfaces for vibrational structure calculations from a multiresolution adaptive density-guided approach: implementation and test calculations", *J. Phys. Chem. A* **2009**, *113*, 8712–8723
>
> 3. E. L. Klinting, B. Thomsen, I. H. Godtliebsen, O. Christiansen, "Employing general fit-bases for construction of potential energy surfaces with an adaptive density-guided approach", *The Journal of Chemical Physics* **Feb. 2018**, *148*, 064113

Chapter 3

Wave-functions for nuclear motion for systems with many degrees of freedom: An overview

We shall intially discuss a few aspects about wave functions for many degrees of freedom. We do this initially with emphasis on time-independent theory, but somewhat similar considerations also applies in time-dependent theory.

3.1 Introduction, notation, and the scaling problem

The number of modes (or degrees of freedom) are denoted by M. We shall usually use m, m', m'' as indices for the modes. We use $\phi_{s^m}^m(q_m)$ to denote the one-mode function s^m for mode m. The dimension of the basis for mode m is N_m. The set of basis functions for mode m is

$$\{\phi_{s^m}^m(q_m), s^m = 1, N_m\} \tag{3.1}$$

The individual basis functions in this basis are one-mode functions in the different modes and there is no restrictions on similarity or di-similarity between the basis functions for the different modes. The basis functions are orthonormal in the sense

$$\langle \phi_{r^m}^m(q_m)|\phi_{s^m}^m(q_m)\rangle = \delta_{r^m s^m} \tag{3.2}$$

Note that for the electronic problem the set of basis of functions is the same for all electrons since the electrons are indistinguishable. Here, we may speak of the different modes individually and each may have its own basis. We denote the one-mode functions for *modals*.

Consider now the system of M modes described in the above basis. In the following we shall work with M-mode wave functions constructed from products of the above one-mode wave functions. A simple product of one-mode wave-functions will be specified by a vector $\mathbf{s}$ of indices specifying the relevant one-mode function for each mode. We use the following notation

$$\Phi_{\mathbf{s}}(\mathbf{q}) = \prod_{m=1}^{M} \phi_{s^m}^m(q_m) = \phi_{s^1}^1(q_1)\phi_{s^2}^2(q_2)....\phi_{s^M}^M(q_M) \tag{3.3}$$

The $\mathbf{s}$ vector is an index vector in the M-dimensional index space. In this way we have built a direct-product space for the M-mode system. We will denote such product functions as Hartree-Products.

A general wave-function in the M-mode space can be written as a linear combination of the direct-product functions

$$\Psi_{\mathbf{c}}(\mathbf{q}) = \sum_{\mathbf{s}} c_{\mathbf{s}} \Phi_{\mathbf{s}} = \sum_{s^1}^{N^1} \sum_{s^2}^{N^2} ... \sum_{s^M}^{N^M} c_{s^1 s^2 ... s^M} \Phi_{s^1 s^2 ... s^M}(q_1, q_2, ..., q_M) \qquad (3.4)$$

The wave-function is thereby completely specified in terms of the vector $\mathbf{c}$ in the space spanned by the $\Phi_{\mathbf{s}}(\mathbf{q})$. The exact wave-function to the "vibrational problem" (the problem of treating N distinguishable particles in a potential) is obtainable as a limiting case with (i) a complete one-mode basis, (ii) an untruncated M-mode function in Eq.(3.4).

From the above general wave-function we see clearly the problem in doing quantum dynamics for many degrees of freedom. The dimensionality of the problem is equal to the number of C coefficients in the above sum. Assume for simplifcty that all modes are treated with a basis of the same size, N_b. Then the number of coefficients are $N_b^M = \exp(M \ln(N_b))$. Thus, we see that formally there is a exponential increase in the number of C parameters with the number of modes, M. Thus, even assuming we could have finite and close-enough-to-exact one-mode basis sets, it will be quite impossible to be exact as the number of modes M increases. This extreme increase in complexity with system size, prevents "'exact'" quantum dynamics for more that a few degrees of freedom, even if exact the exact potentials (and couplings) were know. We recall also that obtaining the potential to high dimensionality involved a detrimental increase in computational cost with the size of the system. So quantum mechanics for many degrees of freedom is unrealistic. But then, on the other hand some things can be done, though these issues are issues of current research. So, while calculations that pretend to be exact (for given potentials) are only possible with 2-4 atomic molecular systems, there are approximate models that do go further than this, as well as there are some models for constructing potentials that are less severe the previously discussed simple direct grid approach. We shall cover only a few aspects here on wave functions as a taster.

Often we shall consider one particular product function as reference and label the one-mode functions according to this reference. We use the nomenclature that i, j, k, l denote one-mode functions occupied in the product function, while a, b, c, d denote one-mode functions not present in the product function. General one-mode functions (occupied or unoccupied) are denoted by r and s. This means for example that ϕ_i denotes one of the set $\{\phi_{i^m}^m(q_m), m = 1, M\}$. The vector $\mathbf{i}$ is correspondingly an index vector $(i^1, i^2, ..., i^M)$ denoting which modals are occupied in the reference product function.

3.2 The vibrational self consistent field method

In this section we describe the vibrational self consistent field (VSCF) method in a standard first quantization language. In another section we introduce second quantization and give a second quantization formulation of VSCF. In other later section we describe more advance wave functions some of which may use the VSCF state as a reference state.

Consider the vibrational Hamiltonian in the following form

$$\hat{H} = \hat{H}_1 + \hat{H}_r \qquad (3.5)$$

where $\hat{H}_1$ is a sum of operators working on one mode at a time

$$\hat{H}_1 = \sum_{m=1}^{M} \hat{h}^m \qquad (3.6)$$

$\hat{H}_r$ contains simply the rest. This partitioning of the Hamiltonian is not very specific. A particularly relevant choice is to take $\hat{H}_1$ as a sum of one-dimensional Harmonic oscillators, but there are other possibilities. We shall neglect $\hat{H}_r$ for a moment and consider a set of modals $\phi_{s^m}^m(q_m)$ that are eigen-functions for the one-mode operators in $\hat{H}_1$

$$\hat{h}^m \phi_{s^m}^m(q_m) = \epsilon_{s^m}^m \phi_{s^m}^m(q_m) \tag{3.7}$$

We thereby have the eigenvectors of $\hat{H}_1$ as product functions

$$\hat{H}_1 \Phi_{\mathbf{s}} = \sum_{m=1}^M \hat{h}^m \Phi_{\mathbf{s}} = \left(\sum_{m=1}^M \epsilon_{s^m}^m\right)\Phi_{\mathbf{s}} = E_{1,\mathbf{s}}\Phi_{\mathbf{s}} \tag{3.8}$$

Thus if the Hamiltonian contained only one-mode terms the wave function would be obtainable as a simple product of one-mode functions, and we simply had to solve for these one-mode eigen-functions.

In the vibrational self consistent field (VSCF) model we seek to find the optimal one-mode functions when the M-mode wave function is a simple product function. Since the Hamiltonian in general does not have the simple form of H_1 this M-mode wave function will not be exact. We begin by selecting a reference index vector which we denote by $\mathbf{i}$. We will think of this as the ground state (or at least we will not take any special action in relation to that it might be an excited state). We shall variationally optimize the energy

$$E_{\mathbf{i}} = \langle \Phi_{\mathbf{i}} | \hat{H} | \Phi_{\mathbf{i}} \rangle \tag{3.9}$$

under variation of the one-mode functions $\phi_{i^m}^m(q_m)$. The one-mode functions are required to be normalized

$$\langle \phi_{i^m}^m(q_m) | \phi_{i^m}^m(q_m) \rangle = 1 \tag{3.10}$$

We may write the first-order variation in the energy upon variation of $\phi_{i^m}^m(q_m)$ as

$$
\begin{aligned}
\delta^m E_{\mathbf{i}} &= 2Re\{\langle \delta\phi_{i^m}^m(q_m) \prod_{m'=1,\neq m}^M \phi_{i^{m'}}^{m'}(q_{m'}) | \hat{H} | \prod_{m''=1}^M \phi_{i^{m''}}^{m''}(q_{m''}) \rangle\} \\
&= 2Re\{\langle \delta\phi_{i^m}^m(q_m) | \hat{F}^{m,\mathbf{i}} | \phi_{i^m}^m(q_m) \rangle\}
\end{aligned}
\tag{3.11}
$$

where we have defined the effective operator

$$\hat{F}^{m,\mathbf{i}} = \langle \prod_{m'=1,\neq m}^M \phi_{i^{m'}}^{m'}(q_{m'}) | \hat{H} | \prod_{m''=1,\neq m}^M \phi_{i^{m''}}^{m''}(q_{m''}) \rangle \tag{3.12}$$

Thus, the $\hat{F}^{m,\mathbf{i}}$ operator for mode m is obtained by integrating the Hamiltonian over all other modes. The normalization of the one-mode functions restricts the allowed variations to satisfy

$$\delta\{\langle \phi_{i^m}^m(q_m) | \phi_{i^m}^m(q_m) \rangle\} = 0 \tag{3.13}$$

or

$$Re\{\langle \delta\phi_{i^m}^m(q_m) | \phi_{i^m}^m(q_m) \rangle\} = 0 \tag{3.14}$$

We may combine these constraints with Eq.(3.11) and write the variational criteria as

$$0 = Re\{\langle \delta\phi_{i^m}^m(q_m) | \hat{F}^{m,\mathbf{i}} | \phi_{i^m}^m(q_m) \rangle\} - \epsilon_{i^m}^m Re\{\langle \delta\phi_{i^m}^m(q_m) | \phi_{i^m}^m(q_m) \rangle\} \tag{3.15}$$

where $\epsilon_{i^m}^m$ is a Lagrangian multiplier. We see that $\epsilon_{i^m}^{m*} = \epsilon_{i^m}^m$ and thus $\epsilon_{i^m}^m$ is real. We may therefore write Eq.(3.15) as

$$0 = Re\{\langle \delta\phi_{i^m}^m(q_m) | \hat{F}^{m,\mathbf{i}} - \epsilon_{i^m}^m | \phi_{i^m}^m(q_m) \rangle\} \tag{3.16}$$

If $\delta\phi_{i^m}^m(q_m)$ is completely arbitrary (apart from the normalization constraint) then both $\delta\phi_{i^m}^m(q_m)$ and $i\delta\phi_{i^m}^m(q_m)$ are allowed variations. From these two different variations we obtain two equations which can simply be combined into

$$0 = \langle\delta\phi_{i^m}^m(q_m)|\hat{F}^{m,\mathbf{i}} - \epsilon_{i^m}^m|\phi_{i^m}^m(q_m)\rangle \tag{3.17}$$

Since this equality must hold for arbitrary $\delta\phi_{i^m}^m(q_m)$ the $\phi_{i^m}^m(q_m)$ one-mode functions must be eigen-functions of the $\hat{F}^{m,\mathbf{i}}$ operator with eigenvalue $\epsilon_{i^m}^m$

$$\hat{F}^{m,\mathbf{i}}\phi_{i^m}^m(q_m) = \epsilon_{i^m}^m\phi_{i^m}^m(q_m) \tag{3.18}$$

The total energy is obtained from Eq.(3.9)

$$E_{\mathbf{i}} = \langle\Phi_{\mathbf{i}}|H|\Phi_{\mathbf{i}}\rangle \tag{3.19}$$

using the modal that satisfies the VSCF eigenvalue equations. The eigenvalue can be written in terms of the Hamiltonian and the eigenfunctions as

$$\epsilon_{i^m}^m = \langle\phi_{i^m}^m(q_m)|\hat{F}^{m,\mathbf{i}}|\phi_{i^m}^m(q_m)\rangle = \langle\Phi_{\mathbf{i}}|\hat{H}|\Phi_{\mathbf{i}}\rangle = E_{\mathbf{i}} \tag{3.20}$$

Note that the total energy and the "one-mode" energy for the modes are the same! This may appear as unusual. It is for example in contrast to the SCF solution for indistinguishable fermions. In the particular relation above for VSCF above it should be recalled the energy for one-mode relates to the average of the full Hamiltonian, and in this way contributions from other modes are also included.

Up to this stage the VSCF treatment is general. The modals may be represented in an analytical basis or represented in another form and the equations solved numerically. In the following section we shall discuss further the use of VSCF with one-mode basis sets.

3.3 Basis set expansion for VSCF

We shall now expand each $\phi_{s^m}^m(q_m)$ function in a set of basis functions $\chi_{v^m}^m(q_m)$ for each mode.

$$\phi_{s^m}^m(q_m) = \sum_{v^m} C_{v^m s^m}^m \chi_{v^m}^m(q_m) \tag{3.21}$$

We denote the $\chi_{v^m}^m(q_m)$ basis as the primitive basis. The metric matrix is defined

$$S_{v^m u^m}^m = \langle\chi_{v^m}^m(q_m)|\chi_{u^m}^m(q_m)\rangle \tag{3.22}$$

We shall here require that the primitive basis is orthonormal

$$S_{v^m u^m}^m = \delta_{v^m u^m} \tag{3.23}$$

If the basis set is not orthonormal it is not a problem to include the metric matrix $S_{v^m u^m}^m$ in the formulas.

Expressed in matrix form in the primitive basis set the VSCF modal eigenvalue equations in Eq.(3.18) becomes

$$\mathbf{F}^{m,\mathbf{i}}\mathbf{C}^m = \mathbf{C}^m\epsilon^{\mathbf{m}} \tag{3.24}$$

where $\mathbf{F}^{m,\mathbf{i}}$ is the $\hat{F}^{m,\mathbf{i}}$ operator in the primitive basis

$$F_{u^m v^m}^{m,\mathbf{i}} = \langle\chi_{u^m}^m(q_m)|\hat{F}^{m,\mathbf{i}}|\chi_{v^m}^m(q_m)\rangle \tag{3.25}$$

and $\epsilon^{\mathbf{m}}$ is a diagonal matrix with the $\mathbf{F}^{m,\mathbf{i}}$ eigenvalues $\epsilon_{r^m}^m$ as elements.

A very relevant example for the set of basis functions is the set of Harmonic oscillator functions but there are other possibilities. If the potential is separable into a sum of Harmonic oscillators, and if we for each mode take the set of Harmonic oscillator eigen-functions with frequency corresponding to the harmonic approximation as basis functions, we have that the solution to the above equations is the harmonic oscillator solutions, meaning that $\mathbf{C}^m$ becomes the unit matrix and $\epsilon_{r^m}^m$ has the harmonic oscillator energy levels as diagonal elements. Thus the formalism expresses easily the exact solution in this limiting case.

For an anharmonic potential the M different matrix eigenvalue equations of the type Eq.(3.25) are coupled. Starting from an initial guess, for example the harmonic oscillator solution discussed above, the equations may be solved iteratively. That is, based on the given set of modes, the eigenvalue equations for one mode is solved leading to a new modal wave function for this mode. Thereafter the eigenvalue equations for the next mode is solved (with the updated first one) and an updated modal is obtained for this mode as well. This sequence continues through all the modes. Since all modals are changed the first mode is no longer the solution to the eigenvalue equations for the new set of modals. The complete sequence may thus be runned again, and the process continued until further changes are considered below a given threshold (if convergent). We shall later return to the solution of the VSCF equations in more detail and with more refined algorithms.

We see that an efficient scheme for calculation of the primitive basis set integrals is needed and (at least parts of or points on) the PES must be determined. It is a major challenge of vibrational structure theory to handle this issue efficiently and accurately.

3.4   Correlated Vibrational Wave functions : basic aspects

3.4.1   A few comments

The VSCF wave function and energies are real anharmonic wave function. If a harmonic potential is used VSCF gives the harmonic oscillator solution, but if a true anharmonic potential is used a different wave function results. Let us divide the anharmonic terms in two parts: i) anharmonic terms in one mode; ii) mode-couplings term relating to two or more modes at the same time. The one-mode terms are treated exactly in VSCF - or at least as exact as the one-mode basis allows. Thus if there are no mode-coupling terms, the separation inherent in VSCF is exact. A numerical VSCF will be accurate as far as the numerical implementation is exact, the basis is close enough to being complete etc. The problem is in the mode-coupling part. This is not treated exact. It is treated partially since it is included in the mean field. However, correlated motion beyond the mean field is not accounted for. That is: each mode feels the other modes only due to the averaged field of the coupling terms over these modes.

For those familiar with electronic structure theory, this is quite similar to the situation for Hartree-Fock, which is also some-times denoted Self-Consistent-Field theory. VSCF and SCF are as the name indicates conceptually similar theories. The error of electronic SCF can be denoted the electron correlation problem. The problem in VSCF is similar. We may denote this problem as the vibrational correlation problem or the mode-mode correlation problem. Below we outline a few principal different approaches for dealing with mode-mode correlation.

3.4.2   Vibrational Configuration Interaction

The vibrational configuration interaction (VCI) pasteurization of the wave-function is an application of the general linear expansion of the wave function in the context of vibrational theory. The particular VCI methods to be discussed here are various flavors of VCI based upon a VSCF

reference state.

Given a particular choice of reference state the linear VCI ansatz can be written

$$|\text{VCI}\rangle = C_{\mathbf{i}} |\Phi_{\mathbf{i}}\rangle + \sum_{\mu} C_{\mu} \tau_{\mu} |\Phi_{\mathbf{i}}\rangle \tag{3.26}$$

in terms of a reference state and a set of orthonormal M-mode states generated by modal excitations out of that reference state. The C_{μ} parameters are the variational VCI parameters determined from VCI eigenvalue equations. The operator τ_{μ} generates an excited configuration

$$\tau_{\mu} |\Phi_{\mathbf{i}}\rangle = |\mu\rangle \tag{3.27}$$

Thus, μ is a compound index defining which modes are excited and to which modals. Thus for example $\mu = (m, a)$ defines excitation from the referenced modal i^m of mode m to a^m.

By including the full set of excitations in the sum over μ, the Full VCI (FVCI) wave function is obtained. The FVCI wave function is the exact wave function for the given Hamiltonian and one-mode basis set. The difference between the VSCF energy and the FVCI energy is the correlation energy in the given one-mode basis set.

Including not all excitation but only up to n-mode excitations we obtain approximated wave functions with much fewer parameters. A hierarchy of VCI models, VCI[1], VCI[2], VCI[3], VCI[4], ... , is obtained in this way that is guaranteed to converge to the FVCI results for that particular set of modals. Thus, a set of wave functions is obtained that give increasing accuracy at the prize of an increasing number of parameters and an accompanying increasing computational cost. If VCI is carried out to sufficiently high excitation level, and there are no other errors (from the PES etc), high accuracy can in principle be obtained. This is possible for systems with relative few degrees of freedom, for example formaldehyde, see fig 3.1.

The parameters and energies for the VCI method is found by solving the VCI eigenvalue equations

$$\mathbf{HC} = E\mathbf{C} \tag{3.28}$$

where the vector $\mathbf{C}$ contains the VCI coefficients and the Hamiltonian matrix $\mathbf{H}$ is defined

$$H_{\mu\nu} = \langle \mu | \hat{H} | \nu \rangle \tag{3.29}$$

The VCI eigenvalue equations are obtained in various standard ways. Either directly from the time-independent Schrödinger equation by inserting the ansatz and projecting (NB ATKINS ch1.) or from applying the variational criterium. Thus, the VCI wave functions is variational.

In stead of first doing a VSCF, followed by a VCI, one could in principle also do a Multi-Configurational (MC) SCF where both the modals and the C coefficients are optimized variationally. This is not widespread in time-independent theory, but actually the foundation of the so called Multi-configurational Time-dependent Hartree (MCTDH) approach, see later.

3.4.3 Vibrational Møller-Plesset perturbation theory

Defining a zeroth order Hamiltonian $\hat{H}_0$ and the corresponding zeroth order wave function from the VSCF mean-field description

$$\hat{H} = \hat{H}_0 + \hat{U} = \hat{F}^{\mathbf{i}} + \hat{U}^{\mathbf{i}} \tag{3.30}$$

$$\hat{F}^{\mathbf{i}} = \sum_{m} \hat{F}^{m,\mathbf{i}} \tag{3.31}$$

perturbation theory can be used to obtain corrections to the VSCF energies and wave functions. The $U^{\mathbf{i}}$ operator describes the difference between the true many-mode interaction and the VSCF
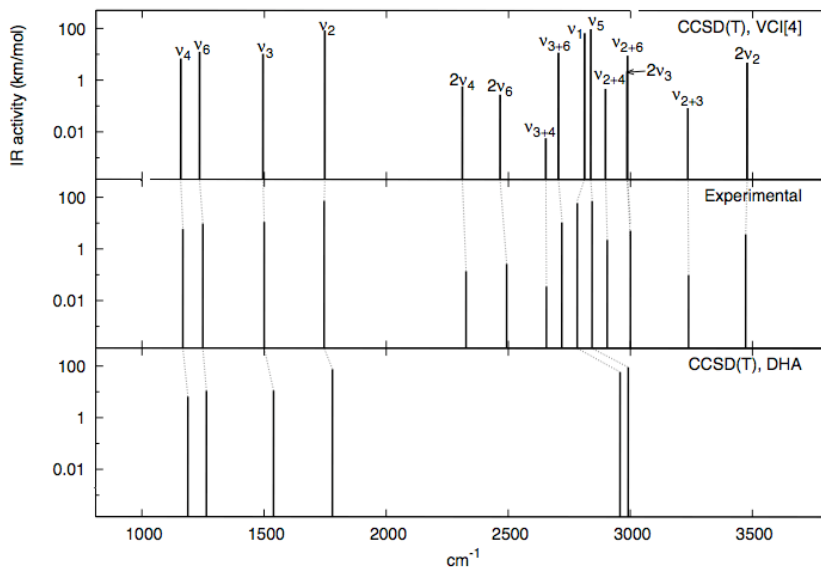
Figure 3.1: Comparison of formaldehyde vibrational excitation energies in experiment, the double harmonic approximation, and VCI[4] using a relative accurate potential) (Seidler and Christiansen, 2007).

mean field representation. Thus, this so-called Vibrational Møller-Plesset theory is the application of perturbation theory to vibrational problems using a VSCF reference state. The perturbation separation is unique to each state motivating the **i** index vector above.

A sequence of perturbational corrections to the VSCF energy for each state is obtained: VSCF, VMP2, VMP3, VMP4, .. etc. Here, the VSCF energy is in fact the VMP1 total energy. Note that the VMP theory is different from the more standard vibrational perturbation theory where the zeroth order problem is the Harmonic oscillator solution.

The VMP expansion is formally written as

$$E_{\mathbf{i}} = \sum_{n=0}^{\infty} E_{\mathbf{i}}^{(n)} \tag{3.32}$$

$$|\Psi_{\mathbf{i}}\rangle = \sum_{n=0}^{\infty} |\Psi^{(n)_{\mathbf{i}}}\rangle \tag{3.33}$$

Introducing the perturbation expansion into the time-independent Schrödinger equation and collecting terms according to order, equations for the perturbed wave function and energies can be obtained.

A problem in VMP theory is that the convergence of the VMP perturbation series can not be guaranteed and in fact convergence is probably generally unlikely. This is illustrated in fig.3.2. Thus, while the VMP2 approach may be a cost-efficient strategy in the calculation of low-lying and well-separated vibrational states, high order VMP is not a vehicle for obtaining arbitrary accuracy in vibrational structure calculations.

31
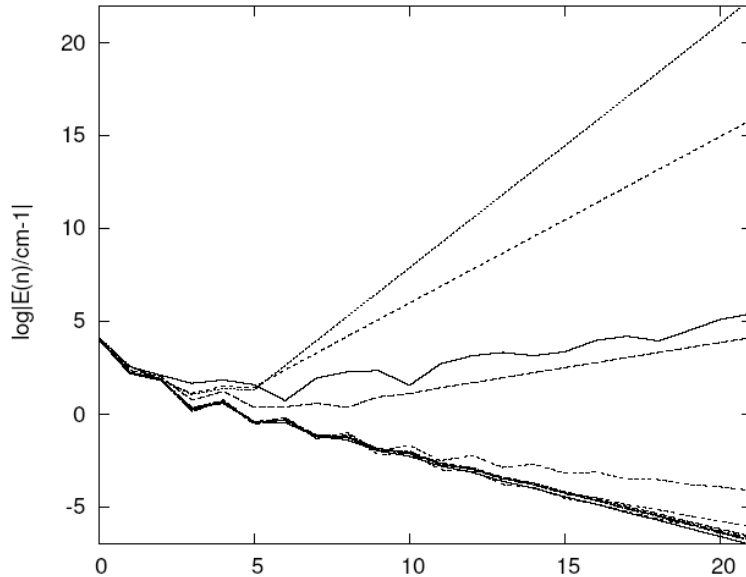
Figure 3.2: Contributions in different orders of perturbation theory in VMP for the fundamental vibrations of ethylene.

We note in passing the VMP is not variational, and for example the VMP ground state energy is not a lower bound to the exact ground state energy. The arbitrary order VMP implementation is described in ref.[13]

3.4.4   Vibrational coupled cluster theory

The vibrational CC (VCC) ansatz for the wave function is based on an exponential parameterization of the wave function in terms of excitations out of a reference state.

$$|\text{VCC}\rangle = \exp(T)|\Phi_\mathbf{i}\rangle. \tag{3.34}$$

Here $|\Phi_\mathbf{i}\rangle$ is the VSCF wave function (other single Hartree product reference state can in principle be used). and T is the cluster operator. The cluster operator contains a sum over allowed excitations

$$T = \sum_\mu t_\mu \tau_\mu \tag{3.35}$$

where t_μ are the VCC cluster amplitudes while τ_μ are the previously described excitation operators indexed by μ. The set of states $\{|\Phi_\mathbf{i}\rangle, \tau_\mu|\Phi_\mathbf{i}\rangle\}$ spans the complete set of states in the M-mode space when all possible excitations μ are included. In this way the VCC ansatz can describe any wave function with a non-vanishing weight of the reference state.

The VCC wave function ansatz is now introduced into the time-independent vibrational Schrödinger equation, and transformed with $\exp(-T)$ to obtain the VCC Schrödinger equation,

$$\exp(-T)\hat{H}\exp(T)|\Phi_\mathbf{i}\rangle = E_{VCC}|\Phi_\mathbf{i}\rangle. \tag{3.36}$$

If the cluster amplitudes are determined the VCC energy can be calculated from the following equation obtained by projection onto the VSCF reference state

$$E_{VCC} = \langle \Phi_{\mathbf{i}} | \exp(-T)\hat{H}\exp(T)|\Phi_{\mathbf{i}}\rangle = \langle \Phi_{\mathbf{i}}|\hat{H}\exp(T)|\Phi_{\mathbf{i}}\rangle. \qquad (3.37)$$

Equations for the cluster amplitudes are obtained by projection onto the manifold of excitations out of the reference state $\langle \mu | = \langle \Phi_{\mathbf{i}} | \tau_{\mu}^{\dagger}$

$$0 = e_{\mu} = \langle \mu | \exp(-T)\hat{H}\exp(T)|\Phi_{\mathbf{i}}\rangle. \qquad (3.38)$$

These equations determines the parameters also for approximate VCC schemes where only a restricted set of excitations are included in the cluster operator Eq.(3.35) and the same set of excitations are considered in Eq.(3.38).

In the limit of a complete cluster expansion, the "full" VCC becomes identical to the FVCI solution. Thus, we have only achieved to write the exact solution in a more complicated non-linear way. The advantage of coupled cluster methods lies in approximate theories where, for a number of reasons, one can expect that the convergence of the coupled cluster expansions is faster compared to that of the corresponding configuration interaction. Thus, if we construct a hierarchy of VCC[n] methods (VCC[1], VCC[2], VCC[3], etc.) analogous to the case for VCI, they will converge faster to the FVCI limit than the VCI[n] series. Figure 3.3 illustrate this.
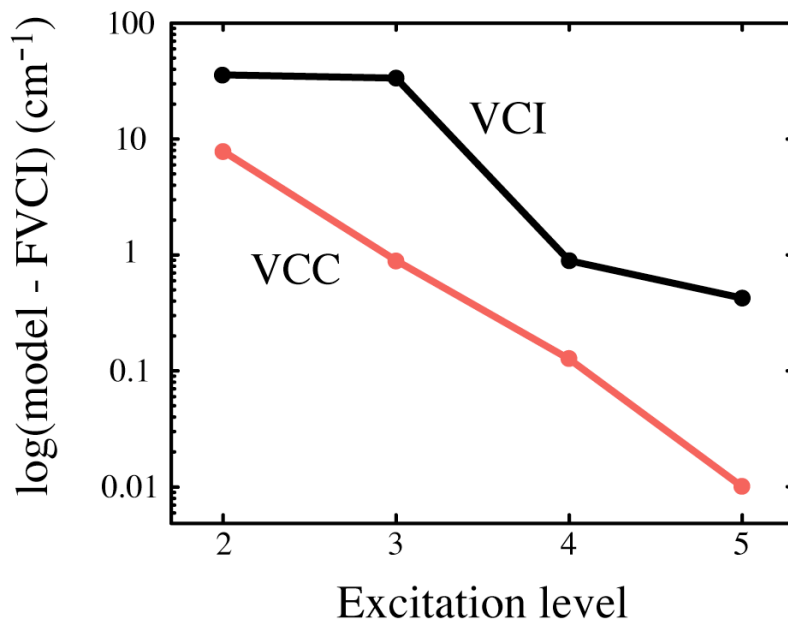
Figure 3.3: Comparison of vibrational excitation energies between VCI[n] and VCC[n]. Errors relative to FVCI in cm^{-1} for the eight lowest fundamental vibrations of ethylene for different truncation of the excitation space determined by n. (Seidler and Christiansen).

What one learn from these discussions is, that it is not only important to include correlation, but it is also crucial how it is done. VCC is much more complicated to implement and the theory above only scratches the surfaces. There is more to the theory for a rigorous formulation. The

above equations are actually quite similar to coupled cluster theory for electronic wave functions. However, this is largely misleading, since many aspects are different. VCC deals with distinguishable modes interacting based on a many-body Hamiltonian which include the molecules PES as the potential. The electronic correlation problem deals with the interaction of indistinguishable electrons, and the Hamiltonian is a priori known, involving only one and two body terms, relating to that all interactions are of Coulomb type.

Chapter 4

Many-mode wave-functions in MidasCpp

Below follows a brief listing of the wave function methods or MidasCpp.

4.1 Vibrational Self-Consistent Field

The original mean-field idea behind vibrational self-consistent field (VSCF) is old by now, and has in particular been pursued by Bowman et al.[14] and Gerber et al[15] since later 1970s.

The VSCF implementation of MidasCpp is described in Ref.[16] and stand out as based on second quantization[17], employing an efficient so-called active term algorithm, and demonstrated linear-scaling in the large molecule limit under reasonable assumptions. The VSCF codes assumes a sum-of-products (SOP) Hamiltonian, such as automatically constructed by the PES parts of MidasCpp.

> **Consider to cite:**
>
> 1. M. B. Hansen, M. Sparta, P. Seidler, O. Christiansen, D. Toffoli, "A new formulation and implementation of vibrational self-consistent field (VSCF) theory", *J. Chem. Theo. and Comp.* **2010**, *6*, 235–248
>
> 2. O. Christiansen, "A second quantization formulation of multi-mode dynamics", *J. Chem. Phys.* **2004**, *120*, 2140–2148

4.2 Vibrational Møller-Plesset perturbation theory

Second order vibrational Møller-Plesset perturbation (VMP) was first suggested Gerber and coworkers in the 1990s (under the name correlation corrected VSCF [18, 19]).

The arbitrary order VMP theory and implementation (and Pade approximants) of MidasCpp is described in Ref.[13]. An alternative to VMP is vibrational auto adjusting perturbation theory[20] but it has not been used much.

> **Consider to cite:**
>
> 1. O. Christiansen, "Møller-Plesset perturbation theory for vibrational wave functions", *J. Chem. Phys.* **2003**, *119*, 5773–5781
>
> 2. E Matito, J. M. Barroso, E. Besalú, O Christiansen, J. M. Luis, "The vibrational auto-adjusting perturbation theory", *Theor. Chem. Acc.* **2009**, *123*, 41–49

4.3 Vibrational Coupled Cluster

4.3.1 Ground-state Vibrational Coupled Cluster

Vibrational Coupled Cluster theory was introduced in Refs.[17, 21]

An efficient VCC[2] algorithm specialized to Hamiltonians with at most two-mode couplings showed the perspective in applications to large systems[22, 23].

The key reference making the intrinsically complicated VCC equations applicable in an efficient manner with higher order couplings in the wave functions is the general contraction algorithms of Ref.[24]. In Ref.[24] it is described how the detailed equations for VCC (and VCI) methods at a given mode-coupling levels can be automatically derived, analyzed for intermediates, and processed. This implementation thus covers, VCC[2], VCC[3], VCC[4], etc. The program is in principle open ended, but VCC[6] and higher is currently of limited practical value and only supported to a limited extent.

For Hamiltonian operators having at most 2M or 3M couplings a cost-efficient approach intermediate between VCC[2] and VCC[3] in computational cost is VCC[2pt3], see Ref.[25].

Similarly, for operators having at most 2M, 3M, or 4M couplings a cost-efficient approach intermediate between VCC[3] and VCC[4] in computational cost is VCC[3pt4], see Ref.[26].

The ground state equations are by standard solved by iterative algorithms as described by [27] normally converging fast.

Consider to cite:

1. O. Christiansen, "Vibrational coupled cluster theory", *J. Chem. Phys.* **2004**, *120*, 2149–2159

2. O. Christiansen, "A second quantization formulation of multi-mode dynamics", *J. Chem. Phys.* **2004**, *120*, 2140–2148

3. P. Seidler, M. B. Hansen, O. Christiansen, "Towards fast computations of correlated vibrational wave functions: Vibrational coupled cluster response excitation energies at the two-mode coupling level", *J. Chem. Phys.* **2008**, *128*, 154113–12

4. P. Seidler, O. Christiansen, "Automatic derivation and evaluation of vibrational coupled cluster theory equations", *J. Chem. Phys.* **2009**, *131*, 234109–15

5. P. Seidler, E. Matito, O. Christiansen, "Vibrational coupled cluster theory with full two-mode and approximate three-mode couplings: The VCC[2pt3] model", *J. Chem. Phys.* **2009**, *131*, 034115–12

6. A. Zoccante, P. Seidler, M. B. Hansen, O. Christiansen, "Approximate inclusion of four-mode couplings in vibrational coupled-cluster theory", English, *Journal of Chemical Physics* **2012**, *136*, 204118–204118–12

7. N. K. Madsen, I. H. Godtliebsen, O. Christiansen, "Efficient algorithms for solving the non-linear vibrational coupled-cluster equations using full and decomposed tensors", *The Journal of Chemical Physics* **2017**, *146*, 134110

4.3.2 VCC response theory

Vibrational response theory[28] is a way to compute also transition properties and vibrational properties. The description of excited states with VCC is strongly recommended to be done with VCC response theory as first reported in Ref.[29]

The general contraction algorithms of Ref.[24] was extended to general VCC response theory in Ref.[30] making also higher-coupling VCC response computations possible with low operation count.

The VCC response theory computation of excited states can proceed in a state by state fashion as in Refs.[24, 29, 30] using for example standard Davidson or Olsen types of algorithms with some means for targeting specific states.[31]

An alternative way is to employ damped response theory. Here the (complex) response function is computed to give the spectrum directly as opposed to first finding the states and then their absorption. This has been described in [32, 33] using respectively Lanczos and band-Lanczos algorithms. Alternatively one solves complex linear-response equations to achieve the same goal as described in Ref.[34]

> **Consider to cite:**
>
> 1. P. Seidler, O. Christiansen, "Vibrational excitation energies from vibrational coupled cluster response theory", *J. Chem. Phys.* **2007**, *126*, 204101
>
> 2. P. Seidler, M. Sparta, O. Christiansen, "Vibrational coupled cluster response theory: A general implementation", *J. Chem. Phys.* **2011**, *134*, 054119–15
>
> 3. B. Thomsen, M. B. Hansen, P. Seidler, O. Christiansen, "Vibrational absorption spectra from vibrational coupled cluster damped linear response functions calculated using an asymmetric Lanczos algorithm", English, *Journal of Chemical Physics* **2012**, *136*, 124101–124101–17
>
> 4. I. H. Godtliebsen, O. Christiansen, "A band Lanczos approach for calculation of vibrational coupled cluster response functions: simultaneous calculation of IR and Raman anharmonic spectra for the complex of pyridine and a silver cation", en, *Physical Chemistry Chemical Physics* **2013**, DOI `10.1039/C3CP50283J`
>
> 5. I. H. Godtliebsen, O. Christiansen, "Calculating vibrational spectra without determining excited eigenstates: Solving the complex linear equations of damped response theory for vibrational configuration interaction and vibrational coupled cluster states", en, *The Journal of Chemical Physics* **Oct. 2015**, *143*, 134108

4.4 Vibrational Configuation Interaction

Vibrational Configuration Interaction methods have been developed in many groups over the years.

The MidasCpp VCI code was developed in the same framework as the VCC code (see above), thus the key-engine in MidasCpp VCI computations are described in Ref.[24].

Special to MidasCpp is also the possibility to compute VCI spectra and vibrational properties from vibrational response theory[28] applied to VCI states[35–37]. Vibrational spectra can be computed directly from the VCI damped response functions as described in Refs.[33, 38] using respectively Lanczos and band-Lanczos algorithms. Alternatively one solves complex linear-response equations to achieve the same goal as described in Ref.[34].

Consider to cite:

1. P. Seidler, O. Christiansen, "Automatic derivation and evaluation of vibrational coupled cluster theory equations", *J. Chem. Phys.* **2009**, *131*, 234109–15

2. P. Seidler, M. B. Hansen, W. Györffy, D. Toffoli, O. Christiansen, "Vibrational absorption spectra calculated from vibrational configuration interaction response theory using the Lanczos method", *J. Chem. Phys.* **2010**, *132*, 164105–15

3. I. H. Godtliebsen, O. Christiansen, "A band Lanczos approach for calculation of vibrational coupled cluster response functions: simultaneous calculation of IR and Raman anharmonic spectra for the complex of pyridine and a silver cation", en, *Physical Chemistry Chemical Physics* **2013**, DOI 10.1039/C3CP50283J

4. I. H. Godtliebsen, O. Christiansen, "Calculating vibrational spectra without determining excited eigenstates: Solving the complex linear equations of damped response theory for vibrational configuration interaction and vibrational coupled cluster states", en, *The Journal of Chemical Physics* **Oct. 2015**, *143*, 134108

5. O. Christiansen, J. Kongsted, M. J. Paterson, J. M. Luis, "Linear Response functions for a vibrational configuration interaction state", *J. Chem. Phys.* **2006**, *125*, 214309

6. M. B. Hansen, O. Christiansen, C. Hättig, "Automated calculation of anharmonic vibrational contributions to first hyperpolarizabilities: Quadratic response functions from vibrational configuration interaction wave functions", *J. Chem. Phys.* **2009**, *131*, 154101

7. M. B. Hansen, O. Christiansen, "Vibrational contributions to cubic response functions from vibrational configuration interaction response theory", *Journal of Chemical Physics* **2011**, *135*, 154107–154107–14

4.5   Tensor-Decomposition Methods

In the context of VCC and VCI tensor-decomposition approaches has been studied for possible gains in VCC and VCI computations [27, 39–41].

At this moment CP-VCC is available for ground state VCC computations as described in Refs.[27, 41]. Extension for VCC response theory is work in progress.

Consider to cite:

1. N. K. Madsen, I. H. Godtliebsen, O. Christiansen, "Efficient algorithms for solving the non-linear vibrational coupled-cluster equations using full and decomposed tensors", *The Journal of Chemical Physics* **2017**, *146*, 134110

2. N. K. Madsen, I. H. Godtliebsen, S. A. Losilla, O. Christiansen, "Tensor-decomposed vibrational coupled-cluster theory: Enabling large-scale, highly accurate vibrational-structure calculations", *The Journal of Chemical Physics* **Jan. 2018**, *148*, 024103

Chapter 5

Primitive basis sets and integrals

In this chapter we describe different choices of primitive basis sets. By primitive basis we mean the one-mode basis that are at the root of the computation. Thus, e.g. VSCF one-mode functions (VSCF modals) are expanded in terms of these primitive one-mode function (primitive modals). There may in some context be several layers of basis functions, but the primitive basis is the root basis, and the basis for which the necessary integrals are computed. Thus we expand the one-mode functions as in VSCF

$$\phi_{s^m}^m(q_m) = \sum_{v^m} C_{v^m s^m}^m \chi_{v^m}^m(q_m) \tag{5.1}$$

All the integrals of the VSCF modals are in turn expressed in terms of the primitive modal integrals. That is, the integrals of one-mode operators h^{m,o^t} in the $\phi_{r^m}^m(q_m)$ basis are simple to obtain from the integrals in the $\chi_{v^m}^m(q_m)$ basis as

$$h_{r^m s^m}^{m,o^t} = \sum_{u^m v^m} (C_{u^m s^m}^m)^* h_{u^m v^m}^{m,o^t} C_{v^m s^m}^m. \tag{5.2}$$

Here the primitive one-mode integrals are

$$h_{u^m v^m}^{m,o^t} = \langle \chi_{u^m}^m(q_m)|h^{m,o^t}|\chi_{v^m}^m(q_m)\rangle. \tag{5.3}$$

Note, that even if the modal basis $\phi_{r^m}^m(q_m)$ often is chosen to be orthonormal, this choice does not depend on the primitive basis $\chi_{v^m}^m(q_m)$ is orthonormal. We shall below describe both the option for which we have the most versatile implementation, the B-spline basis, which is non-orthonormal local basis, as well as a standard choice of orthonormal basis, the Harmonic oscillator basis. We recommend the B-spline basis for most practical use as is discussed in a separate subsection.

5.1 B-spline basis functions

5.1.1 B-spline basis function definition

A set of B-spline basis functions can be fully defined in terms of their order k and a knot sequence **t**. It is required of the knot sequence that it is that defined on a given interval $I = [a, b]$ in a strictly non-decreasing order, i.e. $a = t_0 \leq t_1 \leq t_2 \ldots \leq t_m = b$.

The full set of B-spline basis functions can thereafter be constructed by means of a recursion formula [42], as

$$B_i^1(x) = \begin{cases} 1 & if\ x \in [t_i, t_{i+1}) \\ 0 & elsewhere \end{cases} \tag{5.4}$$

$$B_i^k(x) = \frac{x - t_i}{t_{i+k-1} - t_i} B_i^{k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1}^{k-1}(x). \tag{5.5}$$

Each B-spline basis function is a piecewise polynomial function that extends over $k+1$ knots, which are called the support of the B-spline. Each B-spline have non-zero value outside of the support but is exactly zero at the ends. The means that there will be k B-splines with non-zero contribution to any point in the interval I and the overlap matrix will therefore have a banded structure, since $B_i(x)B_j(x) = 0$ for $|i-j| > k$.

A certain continuity condition will in this regard arise at each knot, which can be ascribed a class $C^{(k-\mu_i-1)}$. The multiplicity μ_i is determined by the number of knots that hold the same value within the interval of interest. Usually interior knots will have multiplicity of 1, since only one knot will be placed at the same value in the knot sequence for these. Exterior knots, right at the end of the interval I will usually have multiplicity of k, which means that the knot sequence is clamped and the B-splines therefore non-periodic. The first and last B-spline function will thereby be unbounded and in MIDASCPP these will be disregarded and not used in the actual calculation as the emphasis is on bound states.

5.1.2   Integrals of B-spline basis functions

The integrals of B-spline basis functions are computed by numerical quadrature. The default choice is a separate Gauss-Legendre quadrature over each subinterval between adjacent knots. While analytical expressions can be found in some cases the numerical scheme is fully sufficient and general. The computation is also very efficient and the calculation of one-mode integrals should never be a bottleneck for any computation.

Standard settings are expected to give very high integral accuracies (order 10^{13} and often better).

5.1.3   Why B-spline basis sets are useful

We recommend to use a B-spline basis for most MidasCpp computations.

Why do we consider B-splines as particularly useful basis functions? The reason we consider B-splines a versatile and useful basis is a combination of different factors. In the PES section it was alluded to that some potentials (most extreme Taylor expanded potentials) may have bad behavior outsome some bounds, because it is simply unrealistic to achieve accurate coverage of the full space in the PES construction. This means there is a risk of performing dangerous extrapolations on the PES. The B-spline basis allows precise control over the limits of the basis. There is local support only. The wave function and the corresponding integral evaluation simply does not enter into other areas than those covered by the basis. This is quite different to for example more global basis functions like harmonic-oscillator basis functions.

The B-spline basis is not an economical primitive basis in the sense that only very few primitive basis functions are required to expand the lowest few bound vibrational states. There other basis sets like a set of Harmonic oscillator basis functions may be much better in some cases. However, here the perspective is that the VSCF procedure is extremely efficient[16] and it is in most applications unimportant whether the underlying basis set has 10, 50 or 100 basis functions. Only if the number of basis functions per mode are of order 1000 and more is the VSCF cost non-negligible in the bigger picture. This means that it is simple to use a large and well-controlled B-spline basis set in VSCF computations (and similar). After the VSCF computation a small set of VSCF modals can be chosen for subsequent computations where the choice of a compact basis is much more crucial. The set of VSCF modals are then used as basis for subsequent computations such as VCC. Thus, VSCF is here used to generate a good physical basis for the concrete case at hand and this VSCF basis spanned on top of an extended B-spline basis is assumed to be superior to

any primitive basis that solves some model potential. The basis of VSCF modals are optimal and can be conveniently truncated to a much smaller number than the primitive B-spline basis.

5.1.4 Defining the B-spline basis in MidasCpp

It is often relevant to define the basis set in terms of a density of the basis functions defined as

$$\rho = \frac{N}{b-a} \tag{5.6}$$

where N is the number of B-spline functions.

In MidasCpp the B-spline basis set can therefore be conveniently specified by giving the interval $[a, b]$, the polynomial order, k, and either the number of B-splines or the density of the basis as given in Eq. (5.6). The interval will in many cases be automatically determined. Typically, it is defined as the boundary of reliability of the PES. What remains is then the order k and either N or ρ. The choice of k is not very critical, and a choice of $k = 10$ has worked fine in many applications, and is the default choice.

5.1.5 B-spline basis sets literature and citations

For details on B-splines and their properties we refer to other sources such as de Boor's book [42].

The MidasCpp B-spline vibrational basis function implementation is described in Ref. [43].

> **Consider to cite:**
>
> 1. D. Toffoli, M. Sparta, O. Christiansen, "Accurate multimode vibrational calculations using a B-spline basis: theory, tests and application to dioxirane and diazirinone", *Mol. Phys.* **2011**, *109*, 673–685

5.2 Harmonic oscillator basis functions

Standard harmonic oscillator basis functions can be used. The computation of the integrals of harmonic oscillator functions proceeds by recursion relations for integrals of operators containing polynomials and first and second derivatives which is extremely fast. The negative consequence of that is that Harmonic oscillator basis cannot at this moment be used with arbitrary fit basis. Only polynomial PESs can be integrated, and the B-spline basis is thus the general work horse.

5.3 Distributed Gaussians

MidasCpp all allows employing distributed Gaussians as another non-orthogonal local basis. At this moment it appears that the above two options are the most interesting being either strictly local the B-spline basis functions are, or close to physical eigenstates in some simple limit, as Harmonic oscillator functions are.

Chapter 6

Machine Learning Algorithms

6.1 Gaussian Process Regression

Gaussian process regression (GPR), sometimes also referred to as kriging [44–46], is a special class of machine learning algorithms. It strength lies in the non parametric regression of multidimensional hypersurfaces using only limited input data. The procedure is based on Bayesian inference.[47] In our case a Gaussian Process (GP) is a statistical distribution of functions, for which every finite linear combination of samples has a joint Gaussian distribution. If we want to model a potential $V(x)$, we introduce the vector $\mathbf{v} = (V(\mathbf{x_1}), V(\mathbf{x_2}), ..., V(\mathbf{x_n}))$, where $\mathbf{x} = (x_1, x_2, ..., x_d)$ is also a vector of dimensionality d containing the coordinates of a given structure (data point). Let us assume that the elements of $\mathbf{v}$ have a multivariant Gaussian distribution. The joint probability density is given as

$$p(\mathbf{v}|\mathbf{m}, \mathbf{K}) = \frac{1}{(2\pi)^{n/2} |\mathbf{K}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{v} - \mathbf{m})\mathbf{K}^{-1}(\mathbf{v} - \mathbf{m})\right) \tag{6.1}$$

for which we use the short hand notation

$$\mathbf{v} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \quad . \tag{6.2}$$

Often a constant or zero mean function $\mathbf{m} = (m(\mathbf{x_1}), m(\mathbf{x_2}), ..., m(\mathbf{x_n}))$ is used. Without losing generality we also apply a zero mean here. In order to make predictions it is exploited that the observed/training data $\mathbf{v}$ and the unknown data $\mathbf{v_*} = (V(\mathbf{x_1^*}), V(\mathbf{x_2^*}), ..., V(\mathbf{x_n^*}))$ is distributed according to

$$\begin{bmatrix} \mathbf{v} \\ \mathbf{v_*} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m} \\ \mathbf{m_*} \end{bmatrix}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X_*}) \\ K(\mathbf{X_*}, \mathbf{X}) & K(\mathbf{X_*}, \mathbf{X_*}) \end{bmatrix}\right) \quad , \tag{6.3}$$

where $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ is a vector of the n data points. This distribution is referred to as prior distribution since it only contains the prior assumption on how the data points are related. By conditioning the joint Gaussian prior distribution on the observations one obtains the Bayesian conditional probability (or posterior distribution)

$$\mathbf{v_*}|\mathbf{v} \sim \mathcal{N}(\mu, \mathbf{\Sigma}) \tag{6.4}$$

with new mean and co-variance function

$$\mu = \mathbf{m_*} + K(\mathbf{X_*}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}(\mathbf{v} - \mathbf{m}) \tag{6.5}$$

$$\mathbf{\Sigma} = K(\mathbf{X_*}, \mathbf{X_*}) - K(\mathbf{X_*}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{X_*}) \tag{6.6}$$

which can be used to make predictions for not observed data. Graphically speaking all functions in the prior distribution are removed, which do not go through the data points. To make predictions

the new mean function μ is used as estimate for the function value for unknown geometries and the diagonal elements of the co-variance matrix $\mathbf{\Sigma}$ give a statistical confidence interval. In the current presentation we assumed that our data is noise free. This is somehow a good assumption since the data is in our case computed and not measured, but each calculation has only a finite precision. To account for this one usually adds a constant noise term σ^2 on the diagonal of the co-variance matrix

$$\mathbf{v} \sim \mathcal{N}(\mathbf{m}, \mathbf{K} + \sigma^2 \cdot \mathbf{I}) \quad . \tag{6.7}$$

Incorporating this term all working equations remain the same. The noise term can be seen as a regularization. Effectively it adds flexibility to the model. The GPR has not to go through each data point exactly anymore, which can result in a smoother regression. We also experimented with it and concluded to add a small constant noise of 10^{-8} as default. To override this setting take a look at the #2 Noise keyword.

6.1.1 Representing the potential

The time dominating costs for GPR is the inversion of the co-variance matrix $\mathbf{K}$, which scales with $\mathcal{O}(n^3)$. As usual the inversion can be avoided by solving the linear systems of equations

$$\mathbf{K}\omega = (\mathbf{v} - \mathbf{m}) \quad . \tag{6.8}$$

Since $\mathbf{K}$ is positive definite the Cholesky decomposition can be used for this purpose, which is quite efficient and stable. In cases of near singularities we switch to the singular value decomposition (SVD). However this can not resolve all numerical problems with duplicated or close data. To ensure diversity in the training data we enforce that two structures m and n differ by a geometrical RMSD

$$\mathrm{RMSD}_{\mathrm{geo}} = \sqrt{\frac{1}{N_{\mathrm{coord}}} \sum_{i}^{N_{\mathrm{coord}}} \left(x_i^{(m)} - x_i^{(n)} \right)^2} \tag{6.9}$$

at least by 10^{-3}, where N_{coord} is the number of coordinates used to represent the molecule i.e. internal coordinates.

Using GPR the potential is represented as

$$V(\mathbf{x}) = \sum_{k} \omega_k k(\mathbf{x}, \mathbf{x}_k) \quad , \tag{6.10}$$

where $k(\mathbf{x}, \mathbf{x}_k)$ is a so called kernel function. The kernel function specifies the co-variance matrix and will be introduced in the next section.

6.1.2 Kernel functions

Using a zero or constant mean function the only thing which defines the GP is the co-variance function/ matrix:

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_i, \mathbf{x}_i) & k(\mathbf{x}_i, \mathbf{x}_j) & \cdots & k(\mathbf{x}_i, \mathbf{x}_n) \\ k(\mathbf{x}_j, \mathbf{x}_i) & k(\mathbf{x}_j, \mathbf{x}_j) & \cdots & k(\mathbf{x}_j, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_i) & k(\mathbf{x}_n, \mathbf{x}_j) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \tag{6.11}$$

which is constructed from kernel functions. In `MidasCpp` we usually employ kernel from the Matérn Kernel family

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{l} \right)^{\nu} K_{\nu} \left(\sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \tag{6.12}$$

where Γ is the gamma function and K_{ν} the modified Bessel function. This general expression is not implemented, but for half integer values of ν some simple special cases can be obtained. The

most common of these special cases are implemented in `MidasCpp` and discussed below. For $\nu = 0$ the Ornstein-Uhlenbeck kernel (OUEXP) can be derived

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \tag{6.13}$$

and for $\nu = 3/2$ the Matérn $3/2$ kernel (MAT32)

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \frac{\sqrt{3}\,\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \exp\left(-\frac{\sqrt{3}\,\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \tag{6.14}$$

and for $\nu = 5/2$ Matérn $5/2$ kernel (MAT52)

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \frac{\sqrt{5}\,\|\mathbf{x} - \mathbf{x}'\|}{l} + \frac{5\,\|\mathbf{x} - \mathbf{x}'\|^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}\,\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \quad . \tag{6.15}$$

Furthermore for $\lim \nu \to \infty$ the squared exponential kernel (SQEXP)

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp -\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2l^2} \tag{6.16}$$

is obtained. As a rule of thumb the ability to describe smoother and smoother functions increases with ν.[47] The kernel function encodes the assumed correlation between the data points $\mathbf{x}$ and $\mathbf{x}'$.

Since the only crucial requirement for a kernel function is positive definiteness, there are many more choices and we note that different kernel functions can for example be combined by addition or multiplication to yield new kernel functions. Multiplication can be seen as an *and* operation since the new kernel has only high value if both base kernel have high value. Similarly addition can be seen as an *or* operation. The new kernel has high value if either one or both of the base kernels has high value. To find an overview about the kernels supported by `MidasCpp` take a look at the #2 Kernel keyword.

From the definition it is evident that these functions depend on parameters like σ and l the so-called hyper parameters. They are not fit parameters in the classical sense, but still carry information about the function, which should be described. σ^2 can for example be interpreted as the average distance of the function away from its mean and l as the length of the 'wiggles' in the function. Usually one can not extrapolate more than l units away from the training data.[47] So far we discussed kernel with a global length scale parameter, but in `MidasCpp` we also have kernel, which use a specific length scale parameter for each coordinate in $\mathbf{x}$. These kernel are referred to as specific coordinate kernel and a **SC** is used as prefix. For more information take a look at the #2 Kernel keyword.

6.1.3 Optimizing the hyper parameters

If a good guess for the hyper parameters is available the GPR can be set up with a fixed set of hyper parameters and just the inversion of the co-variance matrix is required. However, in most cases it is more advisable to optimize these parameteres based on the current training data. GPR offers an intrinsic route for this purpose by maximizing the logarithm of the marginal likelihood

$$\log p(\mathbf{v}|\mathbf{X}, \theta) = J_{vx}^\theta = -\frac{1}{2}(\mathbf{v} - \mathbf{m})^T \mathbf{K}^{-1}(\mathbf{v} - \mathbf{m}) - \frac{1}{2}\log|\mathbf{K}| - \frac{n}{2}\log 2\pi \tag{6.17}$$

with respect to the hyper parameters θ. The term $\frac{n}{2}\log 2\pi$ serves as a normalization constant and $\frac{1}{2}\log|\mathbf{K}|$ is a complexity penalty, where $|\mathbf{K}|$ is the determinant of the matrix $\mathbf{K}$. For a more detailed derivation take a look at Ref. 47.

The partial derivatives with respect to the hyper parameters are given as [47]

$$J_{vx}^{\theta} = \frac{\partial}{\partial \theta_i} = \frac{1}{2} \mathrm{tr} \left(\left(\omega \omega^{\mathrm{T}} - \mathbf{K}^{-1} \right) \frac{\partial \mathbf{K}}{\partial \theta_{\mathrm{i}}} \right) \quad , \tag{6.18}$$

where $\omega = \mathbf{K}^{-1} (\mathbf{v} - \mathbf{m})$. The costs for calculating the marginal likelihood are dominated by inverting the $\mathbf{K}$ matrix, which scales as $\mathcal{O}(n^3)$. If $\mathbf{K}^{-1}$ or equivalently the Cholesky decomposition of $\mathbf{K}$ is available the partial derivative with respect to the hyper parameters can be evaluated using $\mathcal{O}(n^2)$ scaling costs allowing the use of gradient based optimization algorithms. In `MidasCpp` we use resilient backpropagation [48] (Rprop) with an improvement (iRprop) described in Ref. [49]. Rprop uses adaptive update steps, which only use the sign of the gradient

$$\theta_i^{(n+1)} = \theta_i^{(n)} - \mathrm{sign} \left(\frac{\partial J_{vx}^{\theta,(n)}}{\partial \theta_i} \right) \Delta_i^{(n)} \quad . \tag{6.19}$$

The update step is determined according to

$$\Delta_i = \begin{cases} \min(\eta^+ \cdot \Delta_i^{(n-1)}, \Delta_{\max}), & \text{if } \frac{\partial J_{vx}^{\theta,(n-1)}}{\partial \theta_i} \cdot \frac{\partial J_{vx}^{\theta,(n)}}{\partial \theta_i} > 0 \\ \max(\eta^- \cdot \Delta_i^{(n-1)}, \Delta_{\min}), & \text{if } \frac{\partial J_{vx}^{\theta,(n-1)}}{\partial \theta_i} \cdot \frac{\partial J_{vx}^{\theta,(n)}}{\partial \theta_i} < 0 \\ \Delta_i^{(n-1)}, & \text{else} \end{cases} \tag{6.20}$$

If the sign of the partial derivative with respect to θ_i did not change, the update step Δ_i is increased by a factor $\eta^+ > 1$ in order to accelerate convergence. However, if the sign changes, the update step Δ_i is decreased by a factor $0 < \eta^- < 1$ and adaption in the succeeding step is inhibited by setting $\frac{\partial J_{vx}^{\theta,(n-1)}}{\partial \theta_i} = 0$. In case of a change of sign of their partial derivative, the previous weight update is reverted. In the original Rprop optimization algorithm this is always done and in the improved scheme iRprop only if the gradient actually increased. The step length is initially set to Δ_0 and bounded by $\Delta_{\min}$ and $\Delta_{\max}$. In `MidasCpp` we set Δ_0 to 0.5 of the gradient element normalized by the norm of the gradient. As recommended we set the other parameters as $\eta^+ = 1.2$, $\eta^+ = 0.5$, $\Delta_{\max} = 50$ and $\Delta_{\min} = 10^{-6}$.[48] It is to note that the optimization of the hyper parameters is not a trivial task since the marginal likelihood is not convex and multiple maxima/minima can occur.

Consider to cite:

1. G. Schmitz, O. Christiansen, "Gaussian process regression to accelerate geometry optimizations relying on numerical differentiation", *The Journal of Chemical Physics* **2018**, *148*, 241704

2. G. Schmitz, D. G. Artiukhin, O. Christiansen, "Approximate high mode coupling potentials using Gaussian process regression and adaptive density guided sampling", *The Journal of Chemical Physics* **2019**, *150*, 131102

6.2 Using derivative information in GPR

Differentiation is a linear operator and therefore the derivative of a Gaussian Process is also a Gaussian Process. Inference can be done based on the joint Gaussian distribution of function values and partial derivatives.

$$\text{cov}\left(V_p, \frac{\partial V_p}{\partial x_{q_i}}\right) = \frac{\partial k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{q_i}} \quad \text{cov}\left(\frac{\partial V_p}{\partial x_{p_i}}, \frac{\partial V_q}{\partial x_{q_j}}\right) = \frac{\partial^2 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{q_j}} \tag{6.21}$$

In the equation above V_p refers to $V(\mathbf{x}_p)$ and x_{p_i} is the i-th element of the vector $\mathbf{x}_p$. Let us assume that we have potential, gradient and Hessian observations and they are ordered as following

$$V(\mathbf{x}_1), \mathbf{g}(\mathbf{x}_1), \mathbf{h}(\mathbf{x}_1), ..., V(\mathbf{x}_N), \mathbf{g}(\mathbf{x}_N), \mathbf{h}(\mathbf{x}_N) \quad , \tag{6.22}$$

where N is the number of training data points. Furthermore let's assume that the elements of the gradient vector are ordered as $i = 1, n$ and the elements of the Hessian matrix are ordered as $j \le i = 1, n$. We can then write the co-variance matrix as:

$$\mathbf{K} = \begin{pmatrix}
\mathbf{k}_{1,1}^{0,0} & \mathbf{k}_{1,1}^{0,1} & \mathbf{k}_{1,1}^{0,2} & & \mathbf{k}_{1,N}^{0,0} & \mathbf{k}_{1,N}^{0,1} & \mathbf{k}_{1,N}^{0,2} \\
\mathbf{k}_{1,1}^{1,0} & \mathbf{k}_{1,1}^{1,1} & \mathbf{k}_{1,1}^{1,2} & \cdots & \mathbf{k}_{1,N}^{1,0} & \mathbf{k}_{1,N}^{1,1} & \mathbf{k}_{1,N}^{1,2} \\
\mathbf{k}_{1,1}^{2,0} & \mathbf{k}_{1,1}^{2,1} & \mathbf{k}_{1,1}^{2,2} & & \mathbf{k}_{1,N}^{2,0} & \mathbf{k}_{1,N}^{2,1} & \mathbf{k}_{1,N}^{2,2} \\
& & & & & & \\
\mathbf{k}_{N,1}^{0,0} & \mathbf{k}_{N,1}^{0,1} & \mathbf{k}_{N,1}^{0,2} & & \mathbf{k}_{N,N}^{0,0} & \mathbf{k}_{N,N}^{0,1} & \mathbf{k}_{N,N}^{0,2} \\
\mathbf{k}_{N,1}^{1,0} & \mathbf{k}_{N,1}^{1,1} & \mathbf{k}_{N,1}^{1,2} & \cdots & \mathbf{k}_{N,N}^{1,0} & \mathbf{k}_{N,N}^{1,1} & \mathbf{k}_{N,N}^{1,2} \\
\mathbf{k}_{N,1}^{2,0} & \mathbf{k}_{N,1}^{2,1} & \mathbf{k}_{N,1}^{2,2} & & \mathbf{k}_{N,N}^{2,0} & \mathbf{k}_{N,N}^{2,1} & \mathbf{k}_{N,N}^{2,2}
\end{pmatrix} \tag{6.23}$$

where we introduced the following shortcuts

$$k_{p,q}^{0,0} = k\left(\mathbf{x}_p, \mathbf{x}_q\right) \tag{6.24}$$

$$(k_{p,q}^{1,0})_i = \frac{\partial k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i}} \quad i = 1, n \tag{6.25}$$

$$(k_{p,q}^{2,0})_{ij} = \frac{\partial^2 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{p_j}} \quad j \le i = 1, n \tag{6.26}$$

$$(k_{p,q}^{1,1})_{ij} = \frac{\partial^2 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{q_j}} \quad i = 1, n \quad j = 1, n \tag{6.27}$$

$$(k_{p,q}^{2,1})_{ijk} = \frac{\partial^3 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{p_j} \partial x_{q_k}} \quad j \le i = 1, n \quad k = 1, n \tag{6.28}$$

$$(k_{p,q}^{2,2})_{ijkl} = \frac{\partial^4 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{p_j} \partial x_{q_k} \partial x_{q_l}} \quad j \le i = 1, n \quad l \le k = 1, n \tag{6.29}$$

With such a co-variance matrix inference can be done as usual, but we can also predict not only energy, but also gradient and Hessian information in a simple framework. We want to predict

$$\mathbf{v}^* \left(V(\mathbf{x}^*), \mathbf{g}(\mathbf{x}^*), \mathbf{H}(\mathbf{x}^*), \right)^T \quad , \tag{6.30}$$

which we approximate as usual by the mean of the conditional distribution $\mathbf{v}^*|\mathbf{v}$

$$\mathbf{v}^* = \mathbf{K}^* \left(\mathbf{K} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{v} = \mathbf{K}^* \boldsymbol{\omega} \quad , \tag{6.31}$$

where the matrix $\mathbf{K}^*$ is given as

$$\mathbf{K}^* = \begin{pmatrix}
\mathbf{k}_{*,1}^{0,0} & \mathbf{k}_{*,1}^{0,1} & \mathbf{k}_{*,1}^{0,2} & & \mathbf{k}_{*,N}^{0,0} & \mathbf{k}_{*,N}^{0,1} & \mathbf{k}_{*,N}^{0,2} \\
\mathbf{k}_{*,1}^{1,0} & \mathbf{k}_{*,1}^{1,1} & \mathbf{k}_{*,1}^{1,2} & \cdots & \mathbf{k}_{*,N}^{1,0} & \mathbf{k}_{*,N}^{1,1} & \mathbf{k}_{*,N}^{1,2} \\
\mathbf{k}_{*,1}^{2,0} & \mathbf{k}_{*,1}^{2,1} & \mathbf{k}_{*,1}^{2,2} & & \mathbf{k}_{*,N}^{2,0} & \mathbf{k}_{*,N}^{2,1} & \mathbf{k}_{*,N}^{2,2}
\end{pmatrix} \tag{6.32}$$

For each prediction the matrix $\mathbf{K}^*$ is formed and contracted with the vector $\boldsymbol{\omega}$, which has been created during the training period.

6.2.1 Squared exponential kernel

Having established that the only thing required for inference based on derivative information is to construct the derivatives of the kernel function, we turn to the squared exponential kernel as an example. The kernel in its specific coordinate form is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-0.5 \sum_{i=k}^{d} \frac{(x_k - x_k')^2}{l_k^2}\right) \quad . \tag{6.33}$$

It is straightforward to write the derivatives of the kernel up to any order. However, using gradient and Hessian information we need up to 4-th order.

$$\frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_i} = -l_i^{-2} (x_i - x_i') \, k(\mathbf{x}, \mathbf{x}') \tag{6.34}$$

$$\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_j \partial x_i} = -\delta_{ij} l_i^{-2} k(\mathbf{x}, \mathbf{x}') - l_i^{-2} (x_i - x_i') \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_j} \tag{6.35}$$

$$\frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial x_k \partial x_j \partial x_i} = -\delta_{ij} l_i^{-2} \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_k} - \delta_{ik} l_i^{-2} \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_j} - l_i^{-2} (x_i - x_i') \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_k \partial x_j} \tag{6.36}$$

$$\begin{aligned} \frac{\partial^4 k(\mathbf{x}, \mathbf{x}')}{\partial x_l \partial x_k \partial x_j \partial x_i} = &- \delta_{ij} l_i^{-2} \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_l \partial x_k} - \delta_{ik} l_i^{-2} \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_l \partial x_j} - \delta_{il} l_i^{-2} \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_k \partial x_j} \\ &- l_i^{-2} (x_i - x_i') \frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial x_l \partial x_k \partial x_j} \end{aligned} \tag{6.37}$$

The given derivatives were expressed w.r.t. the elements of $\mathbf{x}$. The derivatives w.r.t. the elements of $\mathbf{x}'$ are related to those of $\mathbf{x}$ via the relation

$$\frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_i'} = -\frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_i} \quad . \tag{6.38}$$

Thus, we can obtain the derivatives w.r.t. $\mathbf{x}'$ and mixed derivatives w.r.t. $\mathbf{x}$ and $\mathbf{x}'$ just be changing the sign accordingly. The sign has to be changed as

$$(-1)^l \quad , \tag{6.39}$$

where l is the number of occurrences of differentiation along the directions of $\mathbf{x}'$.

Hyper parameter derivatives

For the hyper parameter optimization it is also necessary to derive the derivatives w.r.t. the hyper parameters. For this purpose the already obtained derivatvies can differentiated again and we obtain:

$$\frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial l_i} = \frac{(x_i - x_i')^2}{l_i^3} k(\mathbf{x}, \mathbf{x}')$$

$$\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial l_j \partial x_i} = 2\delta_{ij} l_i^{-3} (x_i - x_i') \, k(\mathbf{x}, \mathbf{x}') - l_i^{-2} (x_i - x_i') \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial l_j}$$

$$\begin{aligned} \frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial l_k \partial x_j \partial x_i} = &2\delta_{ki} \delta_{ij} l_i^{-3} k(\mathbf{x}, \mathbf{x}') - \delta_{ij} l_i^{-2} \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial l_k} \\ &+ 2\delta_{ik} l_i^{-3} (x_i - x_i') \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_j} - l_i^{-2} (x_i - x_i') \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial l_k \partial x_j} \end{aligned}$$

$$\begin{aligned} \frac{\partial^4 k(\mathbf{x}, \mathbf{x}')}{\partial l_l \partial x_k \partial x_j \partial x_i} = &2\delta_{il} \delta_{ij} l_i^{-3} \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_k} - \delta_{ij} l_i^{-2} \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial l_l \partial x_k} \\ &+ 2\delta_{il} \delta_{ik} l_i^{-3} \frac{\partial k(\mathbf{x}, \mathbf{x}')}{\partial x_j} - \delta_{ik} l_i^{-2} \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial l_l \partial x_j} \\ &+ 2\delta_{li} l_i^{-3} (x_i - x_i') \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_k \partial x_j} - l_i^{-2} (x_i - x_i') \frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial l_l \partial x_k \partial x_j} \end{aligned}$$

$$
\begin{aligned}
\frac{\partial^5 k(\mathbf{x}, \mathbf{x}')}{\partial l_m \partial x_l \partial x_k \partial x_j \partial x_i} =& 2\delta_{mi}\delta_{ij}l_i^{-3}\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_l \partial x_k} - \delta_{ij}l_i^{-2}\frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial l_m \partial x_l \partial x_k} \\
&+ 2\delta_{mi}\delta_{ik}l_i^{-3}\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_l \partial x_j} - \delta_{ik}l_i^{-2}\frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial l_m \partial x_l \partial x_j} \\
&+ 2\delta_{mi}\delta_{il}l_i^{-3}\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_k \partial x_j} - \delta_{il}l_i^{-2}\frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial l_m \partial x_k \partial x_j} \\
&+ 2\delta_{mi}l_i^{-3}\left(x_i - x_i'\right)\frac{\partial^3 k(\mathbf{x}, \mathbf{x}')}{\partial x_l \partial x_k \partial x_j} - l_i^{-2}\left(x_i - x_i'\right)\frac{\partial^4 k(\mathbf{x}, \mathbf{x}')}{\partial l_m \partial x_l \partial x_k \partial x_j}
\end{aligned}
$$

The derivatives w.r.t. σ^2 are simple since in each derivative of the kernel function the factor σ^2 is reproduced in front.

6.3   Descriptors

Important for the performance of GPR and any ML algorithm is to encode the molecular structure in a suitable form for learning in a descriptor. It is beneficial if the descriptor fulfills physical properties like invariance with respect to rotations, translation and permutation of equivalent atoms by construction. Otherwise the ML algorithm has to learn this feature, which usually increases the training set size. The most important properties are invariance under rotations and translation since these properties can not be learned from a finite training set. The permutation of like atoms is desirable, but not crucial if the same order of atoms can be enforced in the descriptor. In MidasCpp several choices for a descriptor are implemented, which can be invoked by the #2 CoordType key word.

- MINT

  A minimal set of $3N_{\mathrm{atom}} - 6$ non redundant coordinates. The automatic detection algorithm searches for bond length, bond angles and dihedrals. But the user can also manually specify out-of-plane bending and define internal coordinates on his own. For this purpose take a look at the keyword #2 ReadIcoordDef. For this type of coordinate learning based on energy, gradient and Hessian information is supported.

- DIST

  Detects all interatomic $N_{\mathrm{atom}} \cdot (3N_{\mathrm{atom}} + 1)/2$ distances in the molecule and uses them as descriptor. For small molecules this descriptor can perform well, but is not advisable for larger systems. For this type of coordinate learning based on energy, gradient and Hessian information is supported.

- ZMAT

  Constructs a Z-matrix with bond length, bond angles and dihedrals. This descriptor is not recommended since all its functionality is also included in MINT.

- XYZ

  Using this option the GPR learns form plain XYZ coordinates. This descriptor lacks invariance under rotation and translation and is only advisable in few contexts and for debugging.

- XYZCENTER

  This option also uses XYZ coordinates, but centers each data point to its center of mass.

- EIGDIST

  This is an experimental descriptor, which extends the DIST descriptor to be invariant under permutation of like atoms.

Part III

MidasCpp Input/Output Reference Manual

Chapter 7

General input options

7.1 General keywords

#1 General

Read General input until next level #1 input key.

#2 AnalysisDir
< string >

Set the directory that is to be used for the analysis calculations.

#2 IoLevel
< integer >

Read line with an integer giving the general IoLevel for Midas in the interval $1-15$. This number defines the lowest IoLevel setting for all modules, i.e. local module IoLevels will be set equal to this number if set lower in the input. The IoLevels $1-5$ contain standard user-friendly output and is specified below, while the IoLevels $6-15$ contain more developer oriented output with description found in the developers manual.

1. Short calculation summary: Summarizes the requested results including but not necessarily limited to
   - Total number of single point calculations performed in constructing the potential and harmonic frequencies.
   - Vibrational energy, (VSCF, VMP2, VCI, VCC).
   - Vibrational frequencies.
   - Properties, (Dipole moments, polarizability, hyper-polarizability, moments of inertia etc.).

2. Detailed calculation summary: Additional frequently useful information on requested results as
   - Vibrational states, (modal information).

3. Short calculation flow: Summarizes key calculation steps in the various iterative algorithms available for `MidasCpp` as

50

- ADGA iterative convergence information.
- VCI and VCC iterative equations convergence information.

4. Detailed calculation flow: Additional frequently required information as produced during the various iterative algorithms available for example including.

- Basis set information.
- Grid bounds for basis and potential.

5. Input verification: Helpful comments in the output to ascertain that individual processes have been carried out, such as

- Write-out statements on processes carried out.
- Local settings.
- Number of single point calculations per ADGA iteration.

7.2   Advanced keywords

#2 **BufSize**
$< integer >$

Provides the maximum size of intermediate memory buffers used (for the DataCont class).

#2 **Debug**

Output lots(!) of debug information. Only for deep debugging.

#2 **FileSize**
$< integer >$

Read line with an integer. The integers provides the maximum data size for single files (for the DataCont class).

#2 **NumLinAlg**
$< string \ = \ $ `LAPACK` $>$ [`LAPACK`] [`MIDAS_CG`] [`MIDAS_JACOBI`]

Set the type of the numerical linear algebra package internally used. Read in line with a string. `LAPACK` sets the use of LAPACK routines in all cases. `MIDAS` sets the use of a combination of inherent routines (`MIDAS_CG`, `MIDAS_JACOBI`, etc.), and LAPACK routines for cases where inherent routines are not available.

#2 **ReserveModes**
$< integer >$

Reads in a line with an integer, which should match the number of modes in the molecular system. This is used to preallocate memory for the modes in order to more efficiently manage the available memory.

#2 **ReserveOper**
< integer >

Reads in a line with an integer, which should match the total number of operators for the molecular system. This is used to preallocate memory for these in order to more efficiently manage the available memory.

#2 **Seed**
< integer vector >

A Keyword to provide MidasCpp with a random seed for initializing start guesses etc. relying on random numbers. Intended to be able to obtain completely reproduce-able results for algorithms using random numbers.

#2 **TechnicalInformation**

This keyword indicates that information on the size of doubles, integers and STL containers are requested, which will then be written in the output.

#2 **Time**

This keyword indicates to Midas that time information is requested but only information regarding functions used during the calculation will be written in the output.

Chapter 8

Singlepoint input options

8.1 General keywords

#1 **SinglePoint**

Read SinglePoint input until next level #1 input key.

#2 **AssignBasis**
$< integer \quad string >$
$[< integer \quad string >]$

Assign the basis set to be used for the QM model when run by the DALTON program. The first line defines the number of types of atoms, followed by a list of pairs consisting of nuclear charge and the basis set label.
(Only has an effect for `SP_DALTON` SP calculations)

> **Example for AssignBasis**
>
> ```
> #2 ASSIGNBASIS
> 1  cc-pVDZ   // assign cc-pVDZ to hydrogen
> 8  cc-pVDZ   // assign cc-pVDZ to oxygen
> ```

#2 **BohrUnits**

Tells `MidasCpp` that atomic units are used for structures which are exchanged between `MidasCpp` and external electronic structure programs. When using this keyword, please make sure that:
In the Runscript (see #2 RunScript): there should be no conversion of the XYZ coordinates from Bohr to Angstrom.
In the Createscript (see #2 InputCreatorScript): Bohr units should be used, for instance in Orca write " ! Bohrs HF-3C ..." , that is, "Bohrs" is added.
(Only has an effect for `SP_Generic` SP calculations)

#2 **DaltonInput**
$< string >$
$[< string = $ #2 End of Dalton Input $>]$

Begin reading of dalton input. All lines following until `#2 End of Dalton Input` is passed

to the DALTON.INP file for Dalton electronic structure calculations.
(Only has an effect for `SP_DALTON` SP calculations)

#2 EsSymmetryOff

Electronic structure symmetry is dis-allowed.
(Only has an effect for `SP_DALTON` SP calculations)

#2 EsSymmetryOn

Electronic structure symmetry is allowed to be set on. (default)
(Only has an effect for `SP_DALTON` SP calculations)

#2 InputCreatorScript
< string >

Names the script for generating the input for the electronic structure program for generic singlepoint calculations.
(Only has an effect for `SP_Generic` SP calculations)

#2 Name
< string >

Reads in name of singlepoint to reference to it later in other input blocks.

#2 PartridgePotential

If defined then use the water potential as parametrized by Partridge and Schwenke (*J. Chem. Phys.*, **106**(1997), 4618). Look in the `test_suite` directory for input/output examples.
(Only has an effect for `SP_MODEL` SP calculations)

#2 PropertyInfo
< string = midasifc.propinfo *>*

Specifies the file, which holds identifiers of the property/properties that are to be calculated in the single point calculation. Below is an example of a property info file, which contains specifications for calculating ground state energy and dipole moments, see also the #3 SetInfo keyword.

Example for PropertyInfo

```
tens_order=(0),descriptor=(GROUND_STATE_ENERGY)
tens_order=(1),descriptor=(X_DIPOLE),Rot_group=(0),element=(0)
tens_order=(1),descriptor=(Y_DIPOLE),Rot_group=(0),element=(1)
tens_order=(1),descriptor=(Z_DIPOLE),Rot_group=(0),element=(2)
```

#2 RunScript
< string >

Names the script, which runs the electronic structure program in genereic single point calculations and converts its output to a readable format for `MidasCpp` in midasifc.prop_general.
(Only has an effect for `SP_Generic` SP calculations)

#2 **RotationThr**

$< \ real \ = \ 1 \cdot 10^{-5} \ >$

The rotation/symmetry threshold used in singlepoint calculations when trying to determine how the molecule has been rotated by the electronic structure program. For directonal properties like the gradient it is important to rotate it back to `MidasCpp` reference frame.
(If `MidasCpp` is unable to determine the rotation it can in some cases help to loosen this threshold slightly (e.g. by an order of magnitude).)

#2 **Type**

$< \ string \ >$

Type of singlepoint, which also defines what other input options are available. Possible types are:

> **Options**
>
> – SP_DALTON
>
> – SP_GENERIC
>
> – SP_MODEL
>
> – SP_TINKER

#2 **ValidationScript**

$< \ string \ >$

Names the script, which validates the output of the electronic structure program.
(Only has an effect for `SP_Generic` SP calculations)

#2 **MLDatabase**

$< \ string \ >$

Specifies which database should be used by the ML algorithm for inference.
(Only has an effect for `SP_ML` SP calculations)

#2 **MLDriver**

$< \ string \ >$

Specifies a MLTask datagroup, which will be used to construct the used GPR or to extract other machine learning oriented options. Options are for example the kernel, the noise level, setting to read a pre stored co-variance matrix or options for training the hyper parameters.
(Only has an effect for `SP_ML` SP calculations)

#2 MLProp

$< \; string \;\; = \;\; \texttt{GROUND\_STATE\_ENERGY} \;\; >$

Specifies for which property a ML algorithm should be used for prediction.
(Only has an effect for `SP_ML` SP calculations)

Chapter 9

Vib input options

9.1 General keywords

#1 Vib

Read vib input until next level #1 input key.

#2 Basis   `!VibBasis`

For defining the primitive one-mode basis to be used for vibrational structure calculations. Reads input for this until next #2 input key. See chapter 10 for a list of the basis-set keywords.

#2 Operat   `!Operator`

Read operator input until next #2 input key. See chapter 11 for a list of the operator keywords.

#2 TdH   `!TDH`

Read input for the TDH module until next #2 key. See chapter 15 for a list of the TDH keywords.

#2 Vcc   `!VCC` `!VCI` `!VMP`

Read input for the correlated vibrational wave function until next #2 key. See chapter 13 for a list of the VCC, VCI, and VMP keywords.

#2 Vscf   `!VSCF`

Read input for VSCF until next level #2 input key. See chapter 12 for a list of the VSCF keywords.

9.2   Advanced keywords

#2 IoLevel

$< integer \ = \ 0 \ >$

The IoLevel for the vibrational calculations. For general information look at the description of #2 IoLevel.

Chapter 10

Basis input options

10.1 General keywords

#3 BSplineBasis `!VibBasis`

    $< \; integer \;\; = \;\; 10 \;\; \mathbf{OR} \; integer \; vector \; >$

Indicates that a B-spline primitive basis should be used to represent the vibrational wave function. The number(s) indicate(s) the highest possible order B-spline function that can be used. It should be noted that the use of a B-spline primitive basis additionally requires specification of the number of B-spline functions (through #3 NPrimBasisFunctions or #3 PrimBasisDensity) and the basis set boundaries (#3 ReadBoundsFromFile or #3 TurningPoint and #3 ScalBounds or #3 GradScalBounds).
(Recommended: 10.)

#3 GaussianBasis `!VibBasis`

    $< \; real \;\; \mathbf{OR} \; real \; vector \; >$

Indicates that a distributed Gaussian primitive basis should be used to represent the vibrational wave function. The number(s) indicate(s) which exponent α should be present in the Gaussian functions to be used. It should be noted that the use of a distributed Gaussian primitive basis additionally requires specification of the number of Gaussian functions (through #3 NPrimBasisFunctions or #3 PrimBasisDensity) and the basis set boundaries (#3 ReadBoundsFromFile or #3 TurningPoint and #3 ScalBounds or #3 GradScalBounds).
(Recommended: 1.00)

#3 GradScalBounds `!VibBasis` `!ADGA`

    $< \; 2 \; reals \; >$

The keyword enables the use of gradients at the ADGA grid end points to scale the extent of the boundaries. The first number is a scaling factor for gradient guided determination of basis set boundaries and roughly corresponds to a percentwise indication of how much larger the basis should be, compared to the actual ADGA grid boundaries. The gradients are obtained individually from the ADGA grid left and right endpoints. If the gradient is large then the basis set boundary is not expanded much beyond the ADGA grid boundary but if the gradient is small then the basis set boundary is expanded greatly. The second number is the maximum extension of the basis set boundaries in Q-units. Compare with

#3 ScalBounds.
(Recommended: 1.5 30.0)

#3 **HOBasis** !VibBasis

$< integer$ **OR** $integer\ vector >$

Indicates that a harmonic oscillator (HO) primitive basis should be used to represent the vibrational wave function. The number(s) indicate(s) the highest possible quantum number v that can feature in a HO function, while the individual HO coefficients will be created by adding together Q_m^2 operator terms of the potential. It should be noted that a total of $v+1$ basis functions will be generated for each mode and that if a vector of integers is given, then this should have a size that corresponds to the number of modes in the systems.
(Recommended: 20)

#3 **IoLevel** !VibBasis

$< integer\ =\ 0\ >$

The IoLevel for the basis set parts.
(Will default to #2 IoLevel if that one is set.)

#3 **NPrimBasisFunctions** !VibBasis

$< integer$ **OR** $integer\ vector >$

Read in a line of either a single integer or a vector of integers. This indicates the number of distributed Gaussian or B-spline functions that are to be generated and inserted into the primitive basis for each mode but if only a single integer is given then this number is extended to all modes.
(Recommended: 200)

#3 **NoBasBeyondMaxPot** !VibBasis

Makes sure that the basis boundaries cannot be set further out than the coordinates corresponding to the maximum values of potential energy on the left and right side. The potential is scanned from zero to the basis boundaries, resulting from either the #3 ScalBounds or the #3 GradScalBounds keywords, in order to determine the maximum potential values, before a possible reduction is carried out.

#3 **PrimBasisDensity** !VibBasis

$< real$ **OR** $real\ vector >$

Read in a line of either a single double or a vector of doubles. This indicates the density with which the number of Gaussian or B-spline functions need to comply, depending on the actual primitive basis set boundaries. If only a single double is given then this number is extended to all modes.
(Recommended: 0.80)

#3 **ReadBoundsFromFile** !VibBasis

Tells the program to read grid boundaries (for either knots or gaussian centers) from file. The optional string specifies the path and name of the file. If no argument is given, it will default to <analysis>/OneMode_Grid_Bounds, where <analysis> is the #2 AnalysisDir directory.

(To be used in conjunction with the #3 NPrimBasisFunctions or #3 PrimBasisDensity keywords.)

#3 ScalBounds   !VibBasis   !ADGA
< 2 *real*s >

Enables the extension (or possibly the shortening) of the grid boundaries for either a B-spline or Gaussian basis beyond what is given, e.g. by an ADGA calculation. The first number is a scaling factor, while the second number defines the maximum extension of the basis set. Compare with #3 GradScalBounds.

(The first number (scaling factor) should be in the range $[0.9, 2.0]$. The second number (max. extension) should be larger than 5 if used in ADGA calculations and larger than -10.0 in general. Recommendation for most calculations: 1.5 20.0)

#3 UseScalingFreqs   !VibBasis

If scale factors are present in the operator file(s) due to the use of frequency-scaled coordinates, then these can simply be read and used in the vibrational structure calculation. If this keyword is not enabled, these scaling factors will alternatively be constructed by the program by reading through the operator file(s) and collecting all harmonic terms.

(This option is recommended if scale factors are present in the operator file(s).)

10.2   Advanced keywords

#3 KeepUnboundedBsplines   !VibBasis

Indicates that the first and last B-spline functions of the primitive basis set should not be disregarded under integral evaluation, even though they may be considered unbounded.

#3 TurningPoint   !VibBasis
< *integer* >

The integer will be used to determine the classical turning points, which in turn are used as the boundaries for the Gaussian and B-spline basis sets.

(Recommended: 10)

Chapter 11

Operator input options

11.1 General keywords

#3 IoLevel !Operator

$< \ integer \ = \ 0 \ >$

The IoLevel for the operator parts.
(Will default to #2 IoLevel if that one is set.)

#3 KineticEnergy !Operator

$< \ string \ =$ Simple $>$ `[None] [Simple] [General] [Polyspherical] [Watson] [ApproxWatson]`
`[ConstWatson] [LinWatson]`

$[< \ string \ >]$

Indicates the kind of nuclear kinetic energy operator to be automatically supplied by `MidasCpp`, unless the Polyspherical option is choosen. Current options covers:

None No kinetic energy operator will be supplied.

Simple A simple kinetic energy operator will be supplied, i.e. `MidasCpp` will automatically generate the $-\frac{1}{2}\frac{\partial^2}{\partial Q_m^2}$ terms for each vibrational coordinate Q_m.

General Currently not implemented.

Polyspherical The kinetic energy operator in polyspherical coordinates will be used and should be supplied by the user as a standard operator file. Note that an additional line of input to this keyword is needed for this option, which is used to specify the file holding the kinetic energy operator.

Watson The full Watson operator for a non-linear molecule will be used, where the simple kinetic energy operator, mass-dependent Coriolis coupling term and the mass-dependent constant potential correctional term are automatically supplied. This requires the additional specification of the **??** or **??**, **??** and **??** keywords. Note that if the **??** keyword is not specified, then this option amounts to the **ApproxWatson** option.

ApproxWatson The approximated Watson operator for a non-linear molecule will be used, where the simple kinetic energy operator, mass-dependent Coriolis coupling term and the mass-dependent constant potential correctional term are automatically supplied. The elements of the inverse effective moment of inertia tensor will be approximated with the equilibrium values. This requires the additional specification of the **??** or **??** and **??** keywords.

ConstWatson The simple kinetic energy operator and the mass-dependent constant potential correctional term of the Watson operator will be used and automatically supplied. This requires the additional specification of the **??** or **??** keywords.

LinWatson The full Watson operator for a linear molecule will be used, where the simple kinetic energy operator and mass-dependent Coriolis coupling term are automatically supplied. This requires the additional specification of the **??** or **??**, **??** and **??** keywords. If the **??** keyword is not given, then this option amounts to the **ApproxWatson** option.

#3 **Name** `!Operator`

< *string* >

The name that will be used to label the operator.

#3 **OperFile** `!Operator`

< *string* >

The string shall give the path and name of the file that contains information on the operator in question. See section 11.3.
(For the Hamiltonian used for the VSCF in an ADGA calculation, this *must* be set to "prop_no_1.mop" of the savedir directory.)

#3 **ScreenZeroTerm** `!Operator`

< *real* = 0 >

Read in a line with a double that specifies the threshold for coefficients in the sum-over-product Hamiltonian. Coefficients lower than this thresholds will be screened but this only applies for operators read from file. This threshold defaults to the used floating point type precision, which is usually a double.
(In most cases it's reasonable to set this to around the machine epsilon.)

11.2 Advanced keywords

#3 **SetInfo** `!Operator`

< *string=string* pairs >

Read in the type of operator and relevant parameters. The input is given as a list of "KEY = VALUE" entries on the next line. The type is specified using the TYPE identifier. (Everything is case insensitive.) Current types and related parameters are:

Type	Parameters
[X,Y,Z]_DIPOLE	*no parameters*
[XX,XY,XZ,YY,YZ,ZZ]_POL	FRQ = External frequency (au)
[XXZ,YYZ,ZZZ,XZX,YZY,ZXX,ZYY]_POL	FRQ1 = Ext. frq. 1 (au) FRQ2 = Ext. Frq. 2 (au)

Example for SetInfo

```
#3 SetInfo
type=xx_pol frq=0.0428
```

11.3 Operator input from file

The operator file can have any name and the may be given in several formats. The format is given by the first line in the input. Two examples are "DALTON_FOR_MIDAS" and "GEN_VSCF_95".

In former case each line is assumed to contain the coefficient followed by integers defining the modes active in this term (everything is separated by spaces). Thus, "1.0 1 1" is a term quadratic in mode "1" ($1.0q_1^2$) while "0.5 1 1 2 2" corresponds to $0.5q_1^2q_2^2$. The input is open ended with respect to the number of integers (thus the number of modes and the power the normal coordinates are raised to).

In the latter case the first line of the file contains in addition to "GEN_VSCF_95" also "N_PROD = " followed by an integer giving the number of products in the terms. Thereafter each line, as in the previous case, defines a term in the operator. In this case we have $2N\_PROD$ integers followed by the coefficient. The first N_PROD integers denotes the modes, and the next N_PROD integers gives the power these coordinates are raised to. Thus with "N_PROD = 6" the line "2 3 6 0 0 0 1 1 2 0 0 0 1.92595e-09" defines the term $1.92595 \times 10^{-9}q_2^1q_3^1q_6^2$.

Chapter 12

VSCF input options

12.1 General keywords

#3 Basis `!VSCF`
  $< string >$

Read the name of the primitive one-mode basis set used in the VSCF calculation. If only one basis set is defined, this keyword can be omitted.

#3 CalcDensities `!VSCF`

Request calculation of the one-mode reduced density matrices. Note that although this keyword is automatically turning on if an ADGA calculation is requested it works with all types of VSCF calculations. See also #3 PlotDensities.

#3 MaxIter `!VSCF`
  $< integer\ =\ 100\ >$

Set maximum number of iterations for VSCF optimization.

#3 Name `!VSCF`
  $< string\ =\ $ 'No name' $>$

Assign a name to the VSCF calculation.
(If performing more than one VSCF calculation, one should assign a unique name to each.)

#3 OccAllFirstOver `!VSCF`

Optimize all first overtones.

#3 OccAllFund `!VSCF`

Optimize all fundamentals.

#3 **OccFirstOver**   !VSCF

$< integer\ vector >$

Read in a vector of mode numbers. Optimize the corresponding first overtones.

#3 **OccFundamentals**   !VSCF

$< integer\ vector >$

Read in a vector of mode numbers. Optimize the corresponding fundamentals.

#3 **OccGenCombi**   !VSCF

$< string\ vector >$

.

.

$< string\ vector >$

Read in lines until the next keyword. Each line gives the modes and excitation levels of a state to be optimized using the notation: $<mode>\hat{}<exci\text{-}level>$.

> **Example for OccGenCombi**
> ```
> In order to optimize the (0,1,2) and (1,0,3) states of water, write:
> #3 OccGenCombi
> 1^1 2^2
> 0^1 2^3
> ```

#3 **OccGroundState**   !VSCF

Optimize the vibrational ground state.
(If nothing else is defined, the ground state will be optimized as default. Therefore, this keyword is only strictly necessary if the ground state is requested together with e.g. fundamentals or overtones. Note that this keyword will append _0 to the name of the VSCF data files. If the ground state is optimized without this keyword _0.0 will be appended.)

#3 **Oper**   !VSCF

$< string >$

Read the name of the Hamiltonian operator used in the VSCF calculation. If only one operator is defined, this keyword can be omitted.

#3 **PlotDensities**   !VSCF

$< integer\ =\ 0 >$

Read in a line with a single integer, which indicates how many modals for each vibrational mode that is to be plotted. The $1M$ wave function and density are plotted and files are dumped to the #2 AnalysisDir directory. Note that the #3 CalcDensities keyword needs to

be specified additionally for this keyword to have any effect and that the fitted potential files will need to be situated in the #2 AnalysisDir directory if harmonic oscillator or Gaussian functions are used as primitive basis for the wave function.

#3 Restart !VSCF

Restart the calculation.

#3 Threshold !VSCF
$< \ real \ = \ 100\epsilon_{\mathrm{machine}} \ >$

Read convergence threshold for VSCF optimization. The convergence criterion is energy difference between iterations.

12.2 Advanced keywords

#3 IoLevel !VSCF
$< \ integer \ = \ 0 \ >$

Set output level for VSCF.
(Will default to #2 IoLevel if that one is set.)

#3 Occup !VSCF
$< \ integer \ vector \ >$

Directly specify an arbitrary state to be optimized in the VSCF calculation. The number of integers must equal the number of modes of the system.
(Can be cumbersome for large system; consider using the keywords #3 OccGroundState, #3 OccFundamentals, #3 OccFirstOver, #3 OccAllFund and #3 OccAllFirstOver for cases that fit with those descriptions.)

Example for Occup

```
#3 Occup      // Optimizes the (1,0,0,2,0,0) state
1 0 0 2 0 0   // of e.g. formaldehyde (6 modes).
```

Chapter 13

VCC input options

`MidasCpp` is able to perform the different types of correlated vibrational-structure calculations in many different ways using very specialized options which can make the input section very verbose. Thus, for standard calculations we strongly recommend using the #3 Method keyword which automatically sets up all the necessary defaults for performing VCC (with and without tensor decomposition), VCI, and VMP calculations.

13.1   General keywords

#3 AllFirstOvertones `!VCI`

Optimize all first overtones with VCI.

#3 AllFundamentals `!VCI`

Optimize all fundamentals with VCI.

#3 Basis `!VibBasis`
$< string >$

Read the name of the primitive one-mode basis set used in the calculation. If only one basis set is defined, this keyword can be omitted.

#3 CROP `!VCC`
$< integer = 3 >$

Use CROP algorithm. Set number of subspace vectors.
(3 vectors is usually enough. Increase if convergence problems occur.)

#3 DIIS `!VCC`
$< integer = 3 >$

Use DIIS algorithm for solving VCC equations. NB: Only the implementation in the tensor-based solver works properly!
(Use CROP instead)

#3 EnergyDiffMax `!VCI`
$< real = 5 \text{ cm}^{-1} >$

Set maximum energy difference for ENERGY and OVERLAPANDENERGY targeting methods

#3 FirstOvertones `!VCI`
$< integer\ vector >$

Optimize selected first overtones with VCI.

#3 Fundamentals `!VCI`
$< integer\ vector >$

Optimize selected fundamentals with VCI.

#3 IOLevel
$< integer = 0 >$

Set IO level for correlated calculations.

#3 ItEqBreakDim `!VCI`
$< integer = 100000 >$

Set break dimension of iterative subspace for solving eigenvalue equations.

#3 ItEqEnerThr `!VCC` `!VCI`
$< real >$

Set energy-change threshold for solving VCC and VCI equations.

#3 ItEqMaxDim `!VCI`
$< integer = 100 >$

Set maximum dimension of iterative subspace for solving linear and eigenvalue equations.

#3 ItEqMaxIt `!VCC` `!VCI`
$< integer = 40 >$

Set maximum iterations of iterative equation solver for correlated calculations.

#3 ItEqResidThr `!VCC` `!VCI`
$< real = 1.e\text{-}6 >$

Set residual threshold for solving VCC and VCI equations.

#3 LimitModalBasis
$< integer\ vector\ =\ \{6\}\ >$

Set the number of VSCF modals for each mode to use in the correlated calculation. If all modals are requested, the #3 UseAllVscfModals keyword can be set
(Using 6-10 modals per mode usually gives converged results.)

#3 Method
$< string >$ [VCC[n]] [CP-VCC[n]] [VCI[n]] [VMPn[m]]
[$< string\ vector\ =\ $ NVCISTATES 1 (for VCI only) $>$] [NVCISTATES <nstates>]

Set the vibrational-structure method to use. For VCI it is also possible to set the number of states to calculate by direct diagonalization of the Hamiltonian. For (CP-)VCC the order (n) can be any natural number, 2pt3 or 3pt4 for perturbative methods, or 2H2 for the fast VCC[2] method with 2-mode-coupled Hamiltonians. For VCI all natural numbers are allowed, but for orders higher than 6 the general-excitation-level algorithm (which may be less optimal) is used. For VMP it is possible to set the order of the energy correction n (due to the $2n + 1$ rule, higher-order energies may be calculated as well) and the maximum excitation level to include in the calculation m. Note that the [m] is optional and can be omitted.

> **Example for Method**
>
> ```
> In order to perform a VCC[2pt3] calculation, write:
> #3 Method
>   VCC[2pt3]
>
> A VCI[3] calculation of the 5 lowest states is performed by writing:
> #3 Method
>   VCI[3]
>   NVCISTATES 5
>
> The VMP3 energy (and VMP1 wave function) is obtained by:
> #3 Method
>   VMP3
> ```

#3 Name
$< string\ =\ $ 'No name' $>$

Assign a name to the calculation.
(If performing more than one calculation, assign a unique name to each.)

#3 NewtonRaphson   !VCC

Use Newton-Raphson algorithm for solving VCC equations. The following options are only for the tensor-based equation solver.
(Use this if serious convergence problems occur.)

#4 **AdaptiveMaxExci** !VCC
    $< real = $ -1 (disabled) $ >$

Increase the excitation level of the Jacobian if the error changes less than this threshold times the current largest error decrease.

#4 **AdaptiveThresholdScaling** !VCC
    $< real = 0.02 >$

Set convergence threshold for linear equations to this value times the norm of the RHS.

#4 **JacobianMaxExci** !VCC
    $< integer = $ -1 (disabled) $ >$

Set maximum excitation order for Jacobian transformation. Use zeroth-order Jacobian for the rest.
(Makes the calculations cheaper, but CROP is more efficient at an even cheaper cost.)

#4 **MaxIter** !VCC
    $< integer = 100 >$

Maximum iterations in the linear-equation solver.

#4 **NoAdaptiveLinSolverThreshold** !VCC

Disable adaptive thresholds for linear equations.

#4 **NoLinSolverPrecon** !VCC

Disable diagonal preconditioner for solving linear equations.
(May be useful if the 0th-order Jacobian is a very bad approximation to the true inverse Jacobian matrix.)

#4 **StepsizeControl** !VCC
    $< real = $ -1 (disabled) $ >$

Set maximum step size for Newton-Raphson algorithm.

#3 **Oper** !Operator
    $< string >$

Read the name of the Hamiltonian operator used in the calculation. If only one operator is defined, this keyword can be omitted.

#3 OverlapMin `!VCI`
$< real = 0 >$

Set minimum squared overlap between target and solution vector for OVERLAP targeting method
(0.1 is the recommended setting)

#3 OverlapSumMin `!VCI`
$< real = 0 >$

Set minimum overlap sum for OVERLAPSUM targeting method
(0.8 is the recommended setting)

#3 Reference `!VCI`

Optimize the VCI reference state.

#3 Restart
$[< integer >]$

Attempt to restart the calculation. Optionally, read excitation level of the calculation to restart from (if restarting from lower-order calculation with same name).

#3 Rsp `!VCC` `!VCI` `!RSP`

Pass control to the response input driver to perform VCC and VCI response calculations (see chapter 14).

#3 SimpleQuasiNewton `!VCC`

Use simple quasi-Newton method for solving the VCC equations. This is default in the old solver.
(The default for the tensor-based solver (CROP) is always better.)

#3 TargetingMethod `!VCI`
$< string = $ BESTHITS $ >$ [BESTHITS] [OVERLAP] [ENERGY] [OVERLAPANDENERGY] [OVERLAPSUM]

Set algorithm for targeting specific VCI states.
(OVERLAPSUM is often a good method)

#3 UseAllVscf

Perform separate correlated calculations on top of each defined VSCF calculation. If the #3 Oper or #3 Basis keywords have been set, only the VSCF calculations using the same operator and basis will be used.

#3 UseVscf
< string >

Set the name of the VSCF calculation to use as reference. If multiple calculations are defined in the same VSCF input block, a separate correlated calculation will be run for each. Also note that it is not necessary to define a separate VSCF input if a default calculation on the vibrational ground state is requested. In that case this keyword can be omitted.

13.2 Advanced keywords

#3 AdaptiveDecompGuess !VCC !TENSOR

Use the amplitudes from the previous iteration as starting guess for recompression in CP-VCC. This does not work together with CROP!
(Do not use this feature.)

#3 AdaptiveDecompThreshold !VCC !TENSOR
< real = -1 (disabled) >

Read C_t. Recompress the CP-VCC amplitudes in iteration n to $T_{\mathrm{CP},t}^{(n)} = C_t ||\Delta \mathbf{t}^{(n)}||$.
(The recommended value (0.005) is automatically set when using #3 Method to set up a CP-VCC calculation.)

#3 AdaptiveTensorProductScreening !VCC !TENSOR
< real >

Adapt the screening threshold in CP-VCC to the decomposition threshold of the individual MCs in the error vector.
(Not automatically set because it is relatively untested, but 0.01 seems to work fine.)

#3 AdaptiveTrfDecompThreshold !VCC !TENSOR
< real = -1 (disabled) >

Read C_e. Recompress the CP-VCC error vectors in iteration n to $T_{\mathrm{CP},e}^{(n)} = C_e T_{\mathrm{CP},t}^{(n)}$ (see #3 AdaptiveDecompThreshold).
(It is more efficient to use individual thresholds for all MCs which is automatically set when using #3 Method to set up a CP-VCC calculation.)

#3 AllowBackstep !VCC

Allow the use of backtracking line search. This is default for all other methods than CROP and DIIS.

#3 BacktrackingThresh !VCC
< real = 0 (only backtrack if error increases) >

Use backtracking line search if error decreases (relatively) less than this value. Note that this is disabled for CROP and DIIS as default.

#3 CPVCCDefaults !VCC !TENSOR

Set defaults for CP-VCC method.
(Use the #3 Method keyword instead.)

#3 CheckIDJA !VCC

Print extra information about the IDJA Jacobian. This might affect computational time slightly.

#3 CheckLaplace !VCC !TENSOR

Check accuray of Laplace-decomposed modal-energy denominators for CP-VCC.

#3 CheckNewtonRaphson !VCC

Print extra information about the Newton-Raphson update. This might affect computational time slightly.

#3 DiagMeth
$< string = LAPACK >$

Set method for diagonalization of matrices.

#3 DisableCheckMethod !VCI !VCC

Allow the V3 transformer to use VCC and VCI methods that have not been tested and verified.

#3 DoNotSave !VSCF

Do not save modals to disk.

#3 EigValSeq !VCI
$< integer >$

Solve for this number of roots at a time when solving eigenvalue equations.

#3 EnerThrForOthers !VCI
$< real = 1000 >$

When using targeting, set energy threshold for states beyond the best hits to this value times the threshold. I.e. as default the extra states are not converged as tight as the best hits.

#3 FullRankAnalysis `!VCC` `!TENSOR`

Output detailed info on ranks of different tensors for CP-VCC.

#3 H2Start `!VCI` `!VCC`

Start solving equations from VCIH2 start guess. NB: Does not work for tensor-based VCC solver!

#3 IDJA `!VCC`
$< real = -1 \text{ (disabled)} >$

Enable *Improved Diagonal-Jacobian Approximation* (IDJA) and set threshold for updating the inverse-Jacobian approximation (only update if the norm of the residual change is larger than this threshold).
(Use of IDJA is not recommended. Use CROP instead.)

#3 IDJAUnitMatrixGuess `!VCC`

Start IDJA from a unit matrix instead of the 0th-order inverse Jacobian.
(Not recommended.)

#3 ISO
$< integer = 0 >$

Set the interaction-space order.

#3 ImprovedPreconInSubspaceMatrix `!VCC`

Use improved preconditioner in constructing the CROP/DIIS subspace matrix (only for the tensor-based solver).
(Only relevant when using improved preconditioning with CROP or DIIS which is not recommended!)

#3 ImprovedPrecond `!VCI`
$< integer >$

Use lower-order Jacobian as preconditioner instead of the zeroth-order approximation.
(Rarely worth the effort)

#3 IndividualMCDecompThresh `!VCC` `!TENSOR`
$< real vector = 1.0 \ 0.0 >$

Use individual decomposition thresholds for the different MCs in the error vector for CP-VCC. Read absolute and relative scaling coefficients.
(This is the recommended approach which is set when using #3 Method. Use the default settings.)

#3 IntermediateRestrictions !VCC !VCI

Set restrictions for storing intermediates in the V3 transformer.
(The default is usually fine)

#4 MaxForward !VCC !VCI
$< integer = 0 >$

Set maximum number of forward contractions on saved intermediate.

#4 MinDown !VCC !VCI
$< integer = 1 >$

Set minimum number of down contractions on saved intermediate.

#4 MinForward !VCC !VCI
$< integer = 0 >$

Set minimum number of forward contractions on saved intermediate.

#3 ItEqMaxItMicro !VCC !VCI
$< integer = 10 >$

Set maximum number of micro iterations of iterative equation solver for correlated calculations (only use-able with #3 UseOldVccSolver).

#3 ItEqMicroThrFac !VCC !VCI
$< real = 2.e\text{-}2 >$

Converge equations of micro iterations to this threshold times the current residual norm (only use-able with #3 UseOldVccSolver).

#3 LambdaScan
$< integer\, real\, real >$

Read number of lamdas, min and max lambda. Solve equations for $H = H_0 + \lambda U$ for all lambdas given.

#3 Lambdas
< real vector >

Solve equations for $H = H_0 + \lambda U$ for all lambdas given.

#3 LaplaceInfo $\boxed{\text{!VCC}}$ $\boxed{\text{!TENSOR}}$

Pass control to the Laplace input driver for Laplace-decomposed energy denominators. See chapter 26.

#3 Level2Solver $\boxed{\text{!VCI}}$ $\boxed{\text{!VCC}}$
< string >

Set the library for solving linear and eigenvalue equations for improved preconditioning in VCI and Newton-Raphson (only use-able with #3 UseOldVccSolver) in VCC.

#3 MaxBacksteps $\boxed{\text{!VCC}}$
< integer = 10 >

Set maximum number of times the solver is allowed to reduce the step size before quitting.

#3 MaxBacktracks $\boxed{\text{!VCC}}$
< integer = 1 >

Set maximum number of step-size reductions in a given backtracking iteration.

#3 MaxExPrMode
< integer vector >

Maximum excitation level for each mode for the correlated wave-function calculation.

#3 MaxExci
< integer >

Maximum excitation level for the correlated wave-function calculation.
(Use the #3 Method keyword instead.)

#3 MaxNormConvCheck $\boxed{\text{!VCC}}$

In the tensor-based non-linear solver, converge the VCC equations based on the maximum error of a given MC instead of the norm of the full error vector.
(Use this for large systems and for non-interacting subsystems)

#3 NModes

$< integer >$

Set the number of modes in the calculation.
(This is rarely necessary.)

#3 NResVecs ![VCI]

$< integer >$

Read number of restart vectors.

#3 NoBackstep ![VCC]

Disable the use of backtracking line search. This is default for CROP and DIIS.

#3 NoOlsen ![VCI]

Disable the Olsen method and use Davidson instead for solving eigenvalue equations.

#3 NoPrecon ![VCC]

Disable preconditioner (inverse-Jacobian approximation) for VCC (only the tensor-based solver).
(Not recommended unless the 0th-order Jacobian is very bad.)

#3 OccAllFirstOver ![VSCF]

Optimize all first overtones.
(Define the reference state in a separate VSCF input block and use the #3 UseVscf and #3 UseAllVscf keywords.)

#3 OccAllFund ![VSCF]

Optimize all fundamentals.
(Define the reference state in a separate VSCF input block and use the #3 UseVscf and #3 UseAllVscf keywords.)

#3 OccFirstOver ![VSCF]

$< integer vector >$

Read in a vector of mode numbers. Optimize the corresponding first overtones.
(Define the reference state in a separate VSCF input block and use the #3 UseVscf and #3 UseAllVscf keywords.)

#3 OccFundamentals ![VSCF]

$< integer vector >$

Read in a vector of mode numbers. Optimize the corresponding fundamentals.
(Define the reference state in a separate VSCF input block and use the #3 UseVscf and #3 UseAllVscf keywords.)

#3 OccGenCombi `!VSCF`

< *string vector* >

.

.

< *string vector* >

Read in lines until the next keyword. Each line gives the modes and excitation levels of a state to be optimized using the notation: <mode>ˆ<exci-level>.
(Define the reference state in a separate VSCF input block and use the #3 UseVscf and #3 UseAllVscf keywords.)

#3 OccGroundState `!VSCF`

Optimize the vibrational ground state.
(Define the reference state in a separate VSCF input block and use the #3 UseVscf and #3 UseAllVscf keywords.)

#3 Occup `!VSCF`

< *integer vector* >

Set occupation vector for the reference state.
(Define the reference state in a separate VSCF input block and use the #3 UseVscf and #3 UseAllVscf keywords.)

#3 Pade `!VMP`

Use Pade resummation techniques in resumming the VMP perturbation series.

#3 PreDiag `!VCI`

< *integer* >

Use prediagonalization to solve VCI equations. Input dimension of prediagonalization matrix.

#3 PreconSavedResiduals `!VCC` `!TENSOR`

Save preconditioned residuals in CROP/DIIS subspace for CP-VCC.
(Not recommended. Use the default instead.)

#3 ResidThrForOthers `!VCI`

< *real* = 1000 >

When using targeting, set residual threshold for states beyond the best hits to this value

times the threshold. I.e. as default the extra states are not converged as tight as the best hits.

#3 SaveRankInfo `!VCC` `!TENSOR`

Save and output statistics on ranks of tensors during CP-VCC optimization.

#3 ScreenIntermedAmps `!VCI` `!VCC`
$< real = 0 >$

Screen terms in the V3 transformer based on amplitudes.

#3 ScreenIntermeds `!VCI` `!VCC`
$< real = 0 >$

Screen terms in the V3 transformer.

#3 SizeIt

Output size of various quantities.
(Only for testing.)

#3 TargetSpace `!VCI`

Target the states delared in the analysis/RspTargetVectors file.
(Only use this for targeting advanced states, not fundamentals and first overtones.)

#3 TensorDecompInfo `!VCC` `!TENSOR`

Read tensor-decomposition settings for recompressing the CP-VCC amplitudes. Passes control to tensor-decomposition input reader (see chapter 25).
(Not required when setting CP-VCC via #3 Method)

#3 TensorProductLowRankLimit `!VCC` `!TENSOR`
$< integer = 20 >$

In CP-VCC, convert direct product of lower rank than this directly to CP format instead of using recompression.

#3 TensorProductScreening `!VCC` `!TENSOR`
$< real\ vector >$

Read screening threshold for direct products (and CP tensors) during calculation of the CP-VCC error vector.
(The recommended value (relative to the convergence threshold) is automatically set when using #3 Method to set up a CP-VCC calculation.)

#3 TimeIt

Time various things such as iterations, etc.

#3 Transformer !VCI !VCC
< string vector >

Set up a transformer for VCI and VCC calculations. The many options are explained in table 13.1 on page 83.
(Use the #3 Method keyword instead unless you want something non-standard!)

#3 TransformerAllowedRank !VCC !TENSOR
< integer >

Read maximum allowed rank for the VCC error vector during transformation.
(The recommended value (1000) is automatically set when using #3 Method to set up a CP-VCC calculation.)

#3 TransformerTensorDecompInfo !VCC !TENSOR

Read tensor-decomposition settings for recompressing the CP-VCC error vectors. Passes control to tensor-decomposition input reader (see chapter 25).
(Not required when setting CP-VCC via #3 Method)

#3 TrueHDiag !VCI !VCC

Use true VCI Jacobian diagonal as preconditioner instead of the zeroth-order approximation.
(Rarely worth the effort)

#3 UseAllVscfModals

Use all VSCF modals for correlated calculation as opposed to using the default in #3 LimitModalBasis
(Rarely necessary)

#3 UseOldVccSolver !VCC

Use the old non-linear-equation solver for solving the VCC equations.
(Only set this keyword if you want to use an option that is not implemented in the tensor-based equation solver.)

#3 VAPT
< integer >

Use VAPT (vibrational auto-adjusting perturbation theory) as correlated wave-function method and read the maximum wave-function order.

#3 VCC !VCC

Use VCC as correlated wave-function method. The order is specified with the #3 Max-Exci and #3 Transformer keywords.

(Use the #3 Method keyword instead unless you want to do something non-standard.)

#3 VCI !VCI
$< integer >$

Use VCI as correlated wave-function method and read the requested number of states. The order is specified with the #3 MaxExci and #3 Transformer keywords. If VCI-related targeting keywords (e.g. #3 Reference, #3 AllFundamentals, etc.) are set, the number of states will be overwritten.

(Use the #3 Method keyword instead unless you want to do something non-standard.)

#3 VMP !VMP
$< integer >$

Use VMP as correlated wave-function method and read the wave-function order (energy will have order $2n + 1$). The maximum excitation level is specified with the #3 MaxExci keyword.

(Use the #3 Method keyword instead unless you want to do something non-standard.)

#3 VMPStart !VCI !VCC

Start solving equations from VMP start guess. (Only use-able with #3 UseOldVccSolver.)

#3 VscfMaxIter !VSCF
$< integer = 100 >$

Set maximum number of iterations for reference VSCF calculation. NB: This cannot be used together with #3 UseVscf or #3 UseAllVscf

(Do not set VSCF input in VCC block. Use #3 UseVscf or #3 UseAllVscf instead!)

#3 VscfScreenZero !VSCF
$< real = 0 >$

Set screening threshold for reference VSCF calculation. NB: This cannot be used together with #3 UseVscf or #3 UseAllVscf.

(Do not set VSCF input in VCC block. Use #3 UseVscf or #3 UseAllVscf instead!)

#3 VscfThreshold !VSCF
$< real = 100\epsilon >$

Set energy-change threshold for reference VSCF calculation. NB: This cannot be used together with #3 UseVscf or #3 UseAllVscf

(Do not set VSCF input in VCC block. Use #3 UseVscf or #3 UseAllVscf instead!)

13.3 Tables

Table 13.1: Possible setting associated with the #3 Transformer keyword.

`trf=orig`	Default. Selects the original, general excitation level transformer. Appropriate for VCC, VCI, and VMP
`t1transh=[true|false]`	Only relevant for VCC. Use T_1-transformed one-mode integrals. Default: `true`
`outofspace=[true|false]`	Only relevant for VCC. Use large intermediate spaces. Possible for historical reasons. Default: `false`
`rsu=[true|false]`	Remove Selected Unlinked. Only relevant for VCC. Certain unlinked terms are removed in the evaluation of the VCC error vector. Currently inconclusive if this is a good strategy.Default: `false`
`trf=2m`	Use fast two-mode only transformer. Requires two-mode excitation space (set using #3 Max-Exci) and two-mode Hamiltonian. Appropriate for VCC, VCI, and VMP
`screening=x`	`x` is a floating point number specifying a screening threshold for expensive contributions in the transformer. Negative `x` turns screening of. Default: `x=0.0`
`trf=v3`	Only available for VCC and VCI[gs]. Use faster V3 transformer.
`type=string`	Sets the specific type of calculation. Current possibilities are `gen` (general), `vcc1pt2`, `vcc2pt3`, `vcc3pt4`, and `vcc3pt4F`. Default: `gen`
`fileprefix=string`	Set prefix for files containing V3 contributions used by transformer. If files do not exist they will be generated. Default: `INSTALL_DATADIR` as specified during configuration.
`t1transh=[true|false]`	Only relevant for VCC. Use T_1-transformed one-mode integrals. Default: `true`
`latex=[true|false]`	Write LaTeX file with V3 contributions. So far only works for general VCC. Default: `false`

Chapter 14

Response input options

14.1 General keywords

#4 AllFirstOvertones `!RSP`

Target all first overtones.

#4 AllFundamentals `!RSP`

Target all fundamental excitations.

#4 BandLanczosIr `!RSP`
< string vector >

Used in conjunction with #4 LanczosRsp to create infra-red spectra from the damped linear response functions. Needs `x-`, `y-`, and `z-dipoles`.

> **Example for BandLanczosIr**
> ```
> #4 LanczosRsp
>   chainlen=215
>   frq_range=(0:4000:5)
>   rsp=(order=2, opers=(x,y,z), gamma=10, name=water)
> #4 BandLanczosIr
>   name=birspec blrsp=water analyze=minima
> ```

#4 BandLanczosRaman `!RSP`
< string vector >

Used in conjunction with #4 LanczosRsp to create Raman spectra calculated from the damped linear response functions. Needs `xx-`, `yy-`, `zz-`, `xy-`, `xz-`, and `yz-polarizabilities` for a given frequency.

> **Example for BandLanczosRaman**
>
> ```
> #4 LanczosRsp
>   chainlen=215
>   frq_range=(0:4000:5)
>   rsp=(order=2, opers=(xx,yy,zz,xy,xz,yz), gamma=10, name=water)
> #4 BandLanczosRaman
>   name=bramspec blrsp=water freq=0.0428 analyze=minima
> ```

#4 **EigenVal** `!VCC` `!VCI` `!VSCF` `!RSP`
    $< integer >$

Read how many excitation energies to calculate. If no targeting is used, the lowest-energy states are calculated.

#4 **FirstOvertones** `!RSP`
    $< integer\ vector >$

Target selected first overtones.

#4 **Fundamentals** `!RSP`
    $< integer\ vector >$

Target selected fundamentals.

#4 **ItEqBreakDim** `!RSP`
    $< integer\ =\ 100000 >$

Set break dimension of iterative subspace for solving eigenvalue equations.

#4 **ItEqEnerThr** `!RSP`
    $< real >$

Set energy-change threshold for solving VCC and VCI response equations.

#4 **ItEqMaxDim** `!RSP`
    $< integer\ =\ 100 >$

Set maximum dimension of iterative subspace for solving linear and eigenvalue equations.

#4 **ItEqMaxIt** `!RSP`
    $< integer\ =\ 20 >$

Set maximum iterations of iterative equation solver for solving linear and eigenvalue equaitons.

#4 **ItEqResidThr**

$<\ real\ =\ 1.e\text{-}6\ >$

Set residual threshold for solving VCC and VCI response equations.

#4 **LanczosRsp**

$<\ string\ vector\ >$

.

.

$[<\ string\ vector\ >]$

Used to define response functions to be calculated by the Lanczos chain method. To specify a response function use `rsp=(<order>, <options>)`. Order 2 defines linear response functions, and order 3 defines quadratic response, etc. Presently only linear and quadratic response is available.

Example for LanczosRsp

```
#4 LanczosRsp
   chainlen=215
   frq_range=(0:4000:5)
   rsp=(order=2, opers=(x,y,z), gamma=10, name=water)
```

#4 **RspFunc**

$<\ string\ vector\ >$

.

.

$[<\ string\ vector\ >]$

Read response functions to calculate. Each line begins with an integer specifying the order of the response (1 is expectation value, 2 is linear response, 3 is quadratic response, etc.) followed by the name(s) of the operator(s) and optionally a set of frequencies. Transition moments are calculated with -1 for ground state to excited state and -11 for excited-excited transitions (only available for VCI). The excited states to calculate ground-to-excited transition moments for are chosen by specifying 'ForStates <state numbers>'. For excited-excited transitions one specifies 'ForLeftStates <state numbers> ForRightStates <state numbers>'. To calculate response in an open frequency interval use 'ForFrq <begin> <end> <steps>', where '<begin>' and '<end>' are in atomic units. A damping factor in atomic units can be supplied by specifying 'ForGamma <gamma>'. Using 'InverseCM' allows to supply frequencies in inverse centimeters instead of in atomic units.

```
#4 RspFunc
// Calculate expectation value for operator named 'oper'
1 oper
// Calculate linear response function for operators 'x' and 'y'
// at zero frequency
2 x y 0.0
// Calculate ground-to-excited transition moments for operator 'x'
// for excited states 0-5
-1 x ForStates [0..5]
// Calculate excited-excited transition moments for operator 'x'
// for excited states 0 and 2
-11 x ForLeftStates 0 2 ForRightStates 0 2
// Calculate damped linear response for operator 'x'
// in a specific frequency interval
2 x x ForFrq 3000 3200 50 ForGamma 10 InverseCM
```

#4 **TargetingMethod**   `!RSP`

$< string =$ BESTHITS $>$ [BESTHITS] [OVERLAP] [ENERGY] [OVERLAPANDENERGY] [OVERLAPSUM]

Set algorithm for targeting specific states. If the tensor-based solver is used for calculating response eigenvalues, OVERLAPSUM is the only option.
(OVERLAPSUM is often a good method)

14.2 Advanced keywords

#4 **EnergyDiffMax**   `!RSP`

$< real = 5 \text{ cm}^{-1} >$

Set maximum energy difference for ENERGY and OVERLAPANDENERGY targeting methods

#4 **OverlapMin**   `!RSP`

$< real = 0 >$

Set minimum squared overlap between target and solution vector for OVERLAP targeting method
(0.1 is the recommended setting)

#4 **OverlapSumMin**   `!RSP`

$< real = 0 >$

Set minimum overlap sum for OVERLAPSUM targeting method
(0.8 is the recommended setting)

Chapter 15

TDH input options

15.1 General keywords

#3 Basis !TDH

$< string >$

Name of the primitive one-mode basis.
(This must be the same basis as the one used in VSCF for generating the initial wave packet.)

#3 ExpectationValues !TDH

$< string\ vector >$

.

.

$< string\ vector >$

Read in names of operators to calculate expectation values for. There can be multiple operators per line and properties are read until the next keyword.

#3 ExponentialTdH !TDH

Use the X-TDH parameterization of the wave function.

#3 ImagTime !TDH

Do imaginary-time propagation in order to obtain the ground-state energy and wave function.

#3 InitialWF !TDH

$< string\ vector >$ [`VSCF <vscf-name>`]

Policy for initializing the TDH wave function. For now, the only option is initializing from the result of a VSCF calculation.

#3 Integrator !TDH

Pass control to the ODE input reader (see section 24). Note that this keyword is mandatory!

#3 IoLevel

!TDH

$< integer = 1 >$

Set IO level for TDH calculation.

#3 LinearTdH

!TDH

Use the L-TDH parameterization of the wave function.

#3 Name

!TDH

$< string = TDH\_0 >$

Assign a name to the TDH calculation.
(Assign a unique name to each calculation if there are multiple TDH input blocks in the input file.)

#3 Oper

!TDH

$< string >$

Name of the operator the TDH wave-packet propagation is performed on.

#3 Properties

!TDH

$< string\ vector >$ [energy] [phase] [autocorr] [halftimeautocorr] [kappanorm]
.
.
$< string\ vector >$ [energy] [phase] [autocorr] [halftimeautocorr] [kappanorm]

Read in the properties that should be calculated and saved to disk. There can be multiple properties per line and properties are read until the next keyword. Options are: energy, phase ($F(t)$ in $e^{-iF(t)}$), auto-correlation function ($S(t)$), auto-correlation function calculated using the half-time wave function, and norms of the kappa vectors (only for X-TDH).

#3 Spectra

!TDH

$< string\ vector >$ [autocorr] [halftimeautocorr]
.
.
$< string\ vector >$ [autocorr] [halftimeautocorr]

Read in names of correlation functions to calculate spectra for. Note that the correlation functions must also be set under #3 Properties.

#3 **TwoModeDensities** !TDH

< *string vector* >

.

.

< *string vector* >

Read in pairs of modes (i.e. mode labels, e.g. Q0, Q1,...) on separate lines to write the two-mode densities for.
(Calculating the values of the wave function on a grid is expensive, so only do this if you want to inspect the wave function.)

#3 **WaveFunctionForModes** !TDH

< *string vector* >

Read in modes (i.e. mode labels, e.g. Q0, Q1,...) to write the wave function for.
(Calculating the values of the wave function on a grid is expensive, so only do this if you want to inspect the wave function.)

15.2 Advanced keywords

#3 **ExpTdHDerivativeProjectionTerms** !TDH

Print detailed information on the different terms contributing to the X-TDH derivative.
(This is only a debug option.)

#3 **ExpTdHMaxFailedDerivatives** !TDH

< *integer* = 1 >

The number of times the step length of the integrator is reduced before the kappa vector is reset to zero.
(Use the default setting unless you have good reason for changing it.)

#3 **KappaNormThreshold** !TDH

< *real* = 0.95 >

Threshold (T_R) for resetting the X-TDH kappa vector to zero for a given mode. This is done if $R_m \geq T_R \times \frac{\pi}{2}$.
(Use the default setting unless you have good reason for changing it.)

#3 **LimitModalBasis** !TDH

< *integer vector* >

Limit the number of basis functions for each mode. This option automatically sets the #3 SpectralBasis keyword and cuts the basis functions corresponding to the highest VSCF modal energies. Therefore, this option makes most sense when using the modal basis of the reference VSCF calculation.

#3 **LinearTdHConstraint** !TDH

$< string = \text{ZERO} >$ [ZERO] [FOCK]

Set the constraint/gauge operator for L-TDH.
(The ODE integrators are usually better at guessing the initial step size when using the ZERO constraint. But there is no big difference.)

#3 **NoAutocorrConvolution** !TDH

Disable convolution of the auto-correlation function. This will increase the effect of Gibbs' phenomenon, but make the peaks sharper.
(Do not disable the convolution.)

#3 **NoLinTdHNormalize** !TDH

Do not re-normalize the L-TDH wave function after each time step.
(Do not use this option with imagnary-time propagation unless you have good reason for doing so. For real-time propagation it does not make much difference.)

#3 **ScreenZeroCoef** !TDH

$< real = 0 >$

Threshold for screening away operator terms in the construction of the mean-field matrix/vector. This is only based on the magnitude of the coefficient in front of the operator.
(Do not use screening as it introduces additional approximations and the TDH code is fast enough without it.)

#3 **SpectralBasis** !TDH

$[< string >]$

Transform the primitive basis to an intermediate orthonormal basis which is the VSCF spectral basis of a given operator. The name of the operator may be provided as an argument. The default is to use the modal basis of the VSCF calculation used for generating the initial wave packet (this is also the only option for X-TDH). This keyword is set automatically if the primitive basis set is non-orthogonal.
(Unless a special intermediate basis (e.g. the eigenbasis of the position operator) is required, this keyword can be omitted.)

#3 **Spectrumenergyshift** !TDH

$< real >$

Set a value (in a.u.) to shift the spectrum towards lower energies. If the string 'E0' is given, the spectrum will be shifted by the energy of the initial wave packet.

#3 **SpectrumInterval** !TDH

$< real vector >$

Read two numbers. Only output the spectrum between these two frequencies.

#3 SpectrumOutputScreening !TDH

< real >

Only write the parts of the spectrum with intensity (relative to highest peak) larger than this threshold.

#3 WfGridDensity !TDH

< integer = 10 >

Density of grid points where the modals are evaluated when plotting the wave function.

#3 WritePrimitiveBasisForModes !TDH

< string vector >

Plot the primitive-basis functions for a given set of modes (i.e. mode labels).

Chapter 16

Pes input options

This section details the input options available to the PES module of `MidasCpp`. The output of which is an analytical representation of the potential energy surface (PES) and/or molecular property surface. The PES can afterwards be used in vibrational structure calculations in order to obtain vibrational ground or excited state energy and excitation energies. Should the PES be calculated along with the dipole surfaces or polarizability surfaces then infra-red (IR) or Raman spectra can be calculated.

16.1 General keywords

#2 **AdgaGridInitialDim**   !ADGA
  $< integer\ =\ 2\ >$

Defines the initial grid dimension, i.e. the initially spanned space in terms of the turning points for a harmonic oscillator of the indicated quantum level.

#2 **AdgaAbsScalFact**   !ADGA
  $< real\ vector\ =\ 1.0\ 1.0\ 10.0\ 10.0\ 10.0\ 10.0\ >$

Defines a prefactor for the absolute ADGA convergence criterion at different mode combination levels, see the **??** keyword.

#2 **AdgaInfo**   !ADGA
  $< integer\ =\ 0\ >$
  $[< string\ >]$

Specifies the mode combination range for an ADGA calculation in order to construct potential energy and/or molecular properties surfaces. The second argument is used to specify the name of the VSCF calculation, which is to calculate the vibrational density for the ADGA. If only a single line of input is given to this keyword, then the ADGA will assume that there is only one VSCF input specified and that this is the one to use.

#2 **AdgaRelScalFact**   !ADGA

$< real\ vector\ =\ 1.0\ 1.0\ 20.0\ 30.0\ 30.0\ 30.0\ >$

Defines a prefactor for the relative ADGA convergence criterion at different mode combination levels, see the #2 ItVDensThr keyword.

#2 **AnalyzeDens**   !ADGA

$< string\ =\ MEAN\ >$ [MEAN] [MAX]

The Adga will use either the mean or maximum VSCF density over all available states in the construction of potential energy and/or molecular property surfaces. The number of states to be considered is specified by the #2 AnalyzeStates keyword.

#2 **AnalyzeStates**   !ADGA

$< integer\ =\ 4\ \mathbf{OR}\ integer\ vector\ >$

Defines the number of states per mode that is to be included in the vibrational density analysis to determine the convergence of the ADGA. This analysis will use either the average or the maximum vibrational density over all state depending on the specifications given to the #2 AnalyzeDens keyword. Note that the vector should have as many elements as there are modes in the system but if only a single integer is given, then this number of states is extended to be set for all modes.

#2 **AvoidRestart**   !ADGA   !Static

Avoid the use of already calculated single point information.

#2 **BoundariesPreOpt**   !ADGA

$< string\ >$

Reads a #1 SinglePoint name, which will be used in the calculation to generate a non-coupled $(1M)$ potential energy surface with which to determine the starting point for the actual ADGA calculation. Initial ADGA potential boundaries can in this way be pre-optimized, preferably with a cheap electronic structure method, before the actual ADGA calculation commences.

#2 **CalcDensAtEachMCL**   !ADGA

Do a VSCF calculation to obtain the vibrational density after each ADGA iteration in order to ascertain the performance of the current potential. Note that this keyword only enables the calculation of the vibrational density and subsequent plotting, i.e. vibrational density based on potentials with higher order mode combination levels will only be used by the ADGA if the #2 UpdMeanDensAtEachMCL keyword is specified.

#2 CalcEffInertiaInvTens !ADGA !Static

Calculate the effective (Coriolis corrected) inertia inverse tensor components for each single point geometry generated in a grid based potential energy surface calculation. Force fields are then constructed by the way of fitting for the six independent components, which will be added to the list of calculated properties. References values and the inertia moment for the reference geometry will also be written to files.

#2 CartesianCoord !Taylor

Indicates that numerical derivatives should be done with respect to Cartesian coordinates.

#2 CutAnalysis

It generates XY type files containing mono and multidimensional cuts of the PES. As example a consistent input is given:

> **Example for CutAnalysis**
>
> ```
> #2 CutAnalysis
> Free 1   fixed 2 0.5 (1D cut with q1 free and q2 fixed at value 0.5)
> Free 0 1 fixed 2 1.0 (2D cut with q0, q1 free and q2 fixed at value 1.0)
> ENDOFCUTANALYSIS
> ```

#2 CutAnalysisMesh
$< integer = 10 >$

It determines the number of point used to generate the grid in the CutAnalysis procedure.

#2 DynamicAdgaExt !ADGA

Allows the ADGA to calculate and check the density between the potential grid bounds and the basis set grid bounds in order to dynamically find an optimal point for extending the potential grid. The grid is expanded when the integrated value of the density between the basis bound and the considered point is larger than half the value given under the #2 ItResDensThr keyword. Note that this procedure is only invoked if the fraction of VSCF density outside the ADGA grid is larger than what is permitted by the #2 ItResDensThr keyword.

#2 **DisableCoriolisAndInertia** !ADGA !Static

This keyword disables the calculation of Coriolis coupling matrices and the moment of inertia tensor in potential calculations. These are otherwise always calculated from the equilibrium geometry, as specified in the molecule file on input.

#2 **Displacement** !Taylor

$< string =$ FREQDSIPLACEMENT $>$ [`SIMPLEDISPLACEMENT`] [`FREQDSIPLACEMENT`]
$< real = 0 >$

Defines the type of displacement and step size to be used in calculation of the numerical derivatives. If the FREQDISPLACEMENT option is choosen, then the displacement will be weighted by means of the harmonic frequencies, which will need to be supplied (Recommended step sizes are in the interval of $10^{-1} - 10^{-4}$)

> **Example for Displacement**
>
> ```
> #2 Displacement
>  SimpleDisplacement
>  1.0e-3
> ```

#2 **DoAdgaExtAtEachMCL** !ADGA

Allows the ADGA to extend the potential grid boundaries at each iteration for all nM potential energy surfaces if required.

#2 **DoExtInFirstIter** !ADGA

Allows the ADGA to extend the potential grid boundaries already from the first iteration. This is in contrast to an ordinary ADGA calculation, where the potential grid boundaries are only extended from the second iteration and onwards. Specifying this keyword will in general converge the ADGA in fewer iterations but at the possible cost of additional single points in the potential. Note that this is not a recommended option when employing a polynomial fit-basis.

#2 **DoubleIncremental** !Static !DIE

$< string >$ [`DIF`] [`DIFACT`]
$< string >$

Indicates that a double incremental expansion (DIE) calculation is to be carried out. The first line specifies the type of double incremental scheme to be used, while the second line specifies the file in which information on the fragment combinations is placed. This file can be created using the #4 WriteIncrInput keyword of the FALCON procedure and be edited manually if necessary. Note, however, that such modifications should be made with caution as there are currently no additional checks on whether the fragment combination range is legitimate. This file together with all .mmol files containing the information for the fragment

combinations need to be placed in the FC_savedir folder, which should be present in the main directory. This is done automatically, when those files are generated using FALCON. Additionally, one has to provide a static grid PES input. Note that the combination of DIE and ADGA is not yet implemented.

#2 **DumpSPInterval** !ADGA !Static

$< integer \ = \ 100 \ >$

Indicates the interval at which calculated single point properties will be dumped to disc in order to avoid complete restart of a potential calculation in the event that it crashes or otherwise fails to complete.

#2 **ExtendGridSettings** !Static

$< integer \ vector \ >$

.

.

$[< integer \ vector \ >]$

$< \#2 \ ENDOFEXTENDEDGRIDSETTINGS \ >$

Reads lines in input until #2 ENDOFEXTENDEDGRIDSETTINGS is found. This keyword is used to change the default settings given under the #2 Staticinfo keyword. It may be used either to overwrite some information in the #2 Staticinfo or to add more mode combinations than the ones defined under #2 Staticinfo. The information for each mode combination should should be provided on one line containing (1) dimension of the mode coupling (M_{dim}), (2) the specific mode coupling (length of vector is M_{dim}), (3) the number of grid points for each mode within the specific mode coupling (length of vector is M_{dim}) and (4) the fractioning for each mode within the specific mode coupling (length of vector is $2 * M_{dim}$). For the fractioning the first M_{dim} elements refer to "minus" displacements while the next M_{dim} elements refer to "plus" displacements. The example below adds a 3-Mode coupling (3) consisting of mode number (0 1 2) with 4 grid points in each mode (4 4 4) and a fractioning of 1/2 for both "minus" and "plus" displacements for each mode (1/2 1/2 1/2 1/2 1/2 1/2). To be used in connection with calculations using grids.

> **Example for ExtendGridSettings**
>
> ```
> #2 EXTENDEDGRIDSETTINGS
> 3 0 1 2 4 4 4 1/2 1/2 1/2 1/2 1/2 1/2
> #2 ENDOFEXTENDEDGRIDSETTINGS
> ```

#2 **FitBasis** !ADGA !Static

Directs input to the fit-basis block which defines the functions that will be used in the fitting routine in order to obtain potential energy or molecular property surfaces, see section 17.

#2 FreqScalCoordInFit $\boxed{\text{!ADGA}}$ $\boxed{\text{!Static}}$

Use frequency-scaled coordinates in the fitting routine, i.e. use $\sqrt{\omega_m}Q_m$. The default is true.

(Recommended when fitting with higher order polynomials.)

#2 GenHessian $\boxed{\text{!Taylor}}$

$< integer >$

$< string >$

.

.

$[< string >]$

Indicate for which property or properties Hessians (and gradients) should be generated for. Note that normal coordinates and harmonic vibrational frequencies will additionally be calculated by specifying this keyword.

Example for GenHessian

```
#2 GenHessian
 4
 Ground_State_Energy
 X_Dipole
 Y_Dipole
 Z_Dipole
```

#2 GridMaxDim $\boxed{\text{!Static}}$

$< integer = 10 >$

Defines the maximum physical dimension of the grid in terms of the turning points for a harmonic oscillator of the indicated quantum level.

#2 ItDensThr $\boxed{\text{!ADGA}}$

$< real = 1.0 \cdot 10^{-2} >$

Convergence threshold for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed relative error of integrals over the VSCF density. (Keywords #2 ItVDensThr and #2 ItNormVDensThr are generally more recommended options.)

#2 ItNormResEnThr $\boxed{\text{!ADGA}}$

$< real = 1.0 \cdot 10^{-6} >$

Convergence threshold ϵ_{abs} for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed absolute error of integrals over norm-like

quantities

$$\sqrt{\int_i \left(\sqrt{\rho_{\text{curr}}^{\text{ave}}(Q_m)} V_{\text{curr}}^{\vec{m}_n}(Q_m) - \sqrt{\rho_{\text{prev}}^{\text{ave}}(Q_m)} V_{\text{prev}}^{\vec{m}_n}(Q_m) \right)^2 dQ_m} < \epsilon_{\text{abs}}$$

$$\wedge \quad \sqrt{\int_i \left(\sqrt{\rho_{\text{curr}}^{\text{ave}}(Q_m)} V_{\text{curr}}^{\vec{m}_n}(Q_m) \right)^2 dQ_m} < \epsilon_{\text{abs}},$$

where intervals with associated integrals that fall below this threshold are considered converged in the ADGA.

(Not recommended for mode combinations of around 3 or higher, since the above quantities are not (currently) efficiently computable utilizing the sum-over-product structure of the potential. In such cases, consider using the #2 ItResEnThr keyword instead.)

#2 ItNormVDensThr !ADGA

$< \text{ real } = 1.0 \cdot 10^{-2} >$

Convergence threshold ϵ_{rel} for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed relative error of integrals over norm-like quantity

$$\frac{\sqrt{\int_i \left(\sqrt{\rho_{\text{curr}}^{\text{ave}}(Q_m)} V_{\text{curr}}^{\vec{m}_n}(Q_m) - \sqrt{\rho_{\text{prev}}^{\text{ave}}(Q_m)} V_{\text{prev}}^{\vec{m}_n}(Q_m) \right)^2 dQ_m}}{\sqrt{\int_i \left(\sqrt{\rho_{\text{curr}}^{\text{ave}}(Q_m)} V_{\text{curr}}^{\vec{m}_n}(Q_m) \right)^2 dQ_m}} < \epsilon_{\text{rel}},$$

where intervals with associated integrals that fall below this threshold are considered converged in the ADGA.

(Not recommended for mode combinations of around 3 or higher, since the above quantities are not (currently) efficiently computable utilizing the sum-over-product structure of the potential. In such cases, consider using the #2 ItVDensThr keyword instead.)

#2 ItResDensThr !ADGA

$< \text{ real } = 1.0 \cdot 10^{-3} >$

Convergence threshold ϵ_ρ for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the maximum allowed fraction of VSCF density that can lay outside the ADGA grid. If too much density is found outside of the current ADGA grid, then the potential boundaries are expanded until this is no longer the case.

#2 ItVDensThr !ADGA

$< \text{ real } = 1.0 \cdot 10^{-2} >$

Convergence threshold ϵ_{rel} for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed relative error of the integral over VSCF density times potential. The intervals with relative error of the integrals over this products that falls below this threshold are considered converged in the ADGA.

#2 IoLevel !ADGA !Static

$< \text{ integer } = 0 >$

Indicates the IoLevel of the PES module.

#2 ItPotThr !ADGA

$< real = 1.0 \cdot 10^{-2} >$

Convergence threshold for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed relative error of integrals over the potential. (Keywords #2 ItVDensThr and #2 ItNormVDensThr are generally more recommended options.)

#2 ItResEnThr !ADGA

$< real = 1.0 \cdot 10^{-3} >$

Convergence threshold ϵ_{abs} for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed absolute error of the integral over VSCF density times potential value. The intervals with absolute error and absolute value of the integrals over this products that falls below this threshold are considered converged in the ADGA.

#2 MeanDensOnFiles !ADGA

Specifying this keyword disables the VSCF calculations performed as part of the ADGA, which means that the files containing the mean density should be provided by the user and placed in the appropriate #2 AnalysisDir directory. Note that a VSCF input can still be given in order to perform a VSCF calculation after the ADGA is finished.

#2 MultiLevelPes !ADGA !Static

$< label \quad integer >$

.

.

$[< label \quad integer >]$

The label is used to specify which #1 SinglePoint electronic structure method is to be used for a given level, while the integer specifies the maximum level of mode-coupling that should be achieved. Each multilevel calculation will be run in its own sub-directory of the main- and scratch directory according to the specified order in which they are given. After all multilevel calculations are done the surfaces will be merged according to the input of this keyword and the final merged surfaces will appear in a sub-directory named FinalSurfaces, (i.e. in case of only a single multilevel, then the surfaces found in the FinalSurfaces sub-directory and the multilevel-specific sub-directory will be identical). The initially spanned space for the first multilevel ADGA calculation will be that specified under the #2 AdgaGridInitialDim keyword but all subsequent multilevel ADGA calculation will use the grid boundaries of the previous converged ADGA calculations as the initially spanned space. This is done as a converged ADGA calculation will generally determine grid boundaries that are more appropriate than what can be determine from a harmonic oscillator guess. Note that all potential energy

or molecular properties surfaces in `MidasCpp` are treated as multilevel surfaces regardless of the actual number of surfaces and that multilevels are always counted from 1. An example is given below.

<div style="border:1px solid red; border-radius:8px;">

Example for MultiLevelPes

```
#2 MULTILEVELPES
  generic_sp_ccsdt 1
  generic_sp_ccsd 2
  generic_sp_rimp2 4
```

</div>

#2 NiterMax   `!ADGA`
$< integer\ =\ 10\ >$

Defines the maximum number of iterations that can be used in the construction of potential energy/molecular properties surfaces with the ADGA. Note that this value has to comply with $N_{\text{NiterMax}} + N_{\text{ItExpGridScalFact}} < 29$.

#2 NoNormalCoord

Indicates that normal coordinates should not be used in generating displaced structures for use with the potential energy and/or molecular property surface construction. Cartesian coordinates will instead be used.

#2 NoScalCoordInFit   `!ADGA` `!Static`

Do not use scaled coordinates in the fitting routine, i.e. use $1 \cdot Q_m$.

#2 Nthreads   `!ADGA` `!Static`
$< integer\ =\ $ `--numthreads` $\textbf{OR}\ integer\ vector >$ [`0 to --numthreads`]

Read a line with one (or more) integer(s), defining the number of threads per MPI rank used in the Midas PES module for each multilevel surface calculation specified under the #2 MultiLevelPes keyword. If only one integer is given but multiple multilevel surface calculations have been specified, then this will be used as the number of threads per MPI rank for each multilevel surface calculation. The keyword must be used in conjunction with the `--numthreads` command-line option, which will in any case set the maximum number of single-point calculations that can be run simultaneously, regardless of the value of this keyword. A value of 0 (for any/all of the multilevels) will run as many threads as has been specified by the `--numthreads` command-line option. Setting a value lower than `--numthreads` can be desired if the individual single-point calculations also does some parallelization on their own.

Note that this keyword can be used in connection with the command line arguments to `mpirun`, e.g. `-np <n_mpi>`. In the example setup described here Midas will run #2 Nthreads times `<n_mpi>` singlepoint calculations in parallel.

#2 **PesGridScalFac** !ADGA !Static

$< real = 1.0$ **OR** *real vector* $>$

Defines the scaling factors that are multiplied onto the (1 dimensional) classical turning points in order to setup the physical dimension of the grid. Numbers should be provided as a vector starting with mode 0 and up to mode $N-1$. The order of the modes are that given in the Molecule.mol file. If this keyword is inactive a common scaling factor of 1.0 is assigned.

#2 **PesFGMesh** !Static

$< integer = 2$ **OR** *integer vector* $>$

If specified then read in a sequence of integers $n_1, n_2, \ldots, n_{MCR}$ specifying the mesh of the finer grid onto which subsequent polynomial fitting will be performed. The number of integers should be equal to the maximum number of vibrational normal modes which are coupled. Then, for the i-th mode coupling, the number of intervals in each direction will be set to n_i times the number of intervals which defines the coarse grid (CG). Note that if a single integer is specified, this will be applied to the whole set of normal mode(s) couplings.

#2 **PesFGScalFac** !Static

$< real = 0.9$ **OR** *real vector* $>$

Defines the portion of the CG to be used when generating the finer grid. For example a value of 0.9 means that boundaries for the finer grids are equal to 0.9 times the boundaries actually set in the generation of the coarse grid. (and specified with the #2 PesGridScalFac key).

#2 **PesPlotRealEsp** !ADGA

If specified then **.data** files will be written to the #2 AnalysisDir directory over the course of an ADGA calculation with non-fitted single point information. This information is not limited to energy, i.e. all available property information will be written into seperate files.

#2 **PesUserDefinedBound** !Static

$< real vector >$ [`<integer>` AllPot] [`<integer>` OnlyFullPot]

Defines the physical dimension, when multiplies with the corresponding elements from #2 PesGridScalFac of the grid. Numbers should be provided as a vector starting with mode 0 and up to mode $N-1$. The order of the modes are that given in the Molecule.mol file. If this keyword is inactive the physical dimension will be chosen on the basis of the HO quantum number.

#2 **PlotAdga2D** !ADGA

Indicates that 2-dimensional potential cuts should be plotted and stored on file in the directory specified under the #2 AnalysisDir keyword.

#2 **PotValsFromFile** !ADGA

In the ADGA procedure for PES construction, potential values for the lowest mode combinations are obtained from the operator file, if not already available from a single-point calculation. This can happen if the ADGA grid becomes finer on higher-order mode combination than was necessary for its corresponding lower-order mode combinations.

#2 **SaveDispGeom** !ADGA !Static

Save all geometries that are generated during the constructing of the potential energy or molecular property surface calculation to a file named 'displaced_structures', which will be situated in the #2 AnalysisDir directory.

#2 **SaveEspOut** !ADGA !Static

Save each individual electronic structure calculation output file.

#2 **SaveInpFiles** !ADGA !Static

Save each individual electronic structure calculation input file.

#2 **SaveItOperFile** !ADGA

A copy of the operator file (.mop) is generated for each ADGA iteration if this keyword is specified. This is especially useful if one is interested in tracking the evolution of the operator files.

#2 **SaveSpScratchDir** !ADGA !Static

Specifying this keyword will save the scratch directory for each individual single point calculation, which is carried out while constructing the potential energy or molecular property surface.
(A recommended option for debug purposes.)

#2 **SaveTrainData** !ADGA !MLEARN

If #2 SaveDispGeom is set it will also save the needed data to train a Machine Learning algorithm. For each property i it will dump in the `SaveIoDir` xyz files as `database_prop_no_i.xyz`

#2 **Staticinfo** !Static
< *integer* >

< integer vector >
< integer vector >

Read information for default grid settings. Read three lines containing (1) the mode combination range (2) a vector of grid points in each mode mode combination level and (3) a vector containing the fractioning in each mode combination level.

Example for Staticinfo

```
#2 StaticInfo
 2          // Mode combination level is 2
16  8       // 16 and 8 grid points in 1- and 2-Mode couplings
 1  3/4     // Fractioning: 1 and 3/4 for 1- and 2-Mode couplings
```

#2 **Symthr**   [!ADGA] [!Static]

$< real = 1.0 \cdot 10^{-6} >$

Threshold for how similar different single point geometries need to be in order to be called symmetry equivalent.

#2 **Symmetry**   [!ADGA] [!Static]

< string >

Indicates the point group symmetry to be used for surface construction, where some single points might be symmetry equivalent. Currently the Abelian point groups, i.e. C_i, C_s, C_2, D_2, C_{2v}, C_{2h} and D_{2h} are implemented. If this keyword is not specified, then the point group is assumed to be C_1 and symmetry will therefore not be used in surface construction. The molecule must be oriented such that the symmetry elements (mirror planes, rotation axes) align with the Cartesian x, y, z planes/axes. Note that the symmetry used in surface construction does not, in any way, influence the symmetry that might be used by an electronic structure program. Note that you need to provide the correct symmetry labels for the vibrational coordinates in the .mmol file. Find a list of the irreps belonging to the supported point groups given in the notation used in the input file:

- **C1**: A
- **CS**: AP, APP
- **C2**: A, B
- **D2**: A, B1, B2, B3
- **C2V**: A1, A2, B1, B2
- **C2H**: Ag, Bg, Au, Bu
- **D2H**: Ag, B1g, B2g, B3g, Au, B1u, B2u, B3u

#2 **TaylorInfo**   [!Taylor]

< integer integer integer >

Defines how Taylor expansion in terms of derivative force fields and derivative property surfaces should be done in terms of i) property order, ii) mode combination range and iii) Taylor expansion maximum order. Property order is the order of the property from wich the numerical derivatives should be calculated which is at the moment limited to 0, e.g. the energy. The mode combination range is the maximum number of modes allowed to coupled. Note that specifying this keyword overrides any indication of MC level given under #2 MultiLevelPes but that an integer number is still required.

#2 **ThresFreq** !Taylor

$< \ real \ = \ 20 \ >$

Sets the threshold for sorting out harmonic frequencies based on their value in $cm^{-}1$ describing translation/rotation. To be used in connection with derivative force fields and derivative property surfaces.

#2 **UpdMeanDensAtEachMCL** !ADGA

Allows the ADGA to calculate and update the vibrational density at each iteration for all nM potential energy surfaces. Note that by specifying this keyword, the #2 CalcDensAtEachMCL keyword will automatically be set to true.

#2 **UseFitBasis** !ADGA !Static

$< \ string \ >$

Indicates the fit-basis to be used by name.

#2 **VibPol** !ADGA !Static !Taylor

Calculate the static vibrational polarizability using the double-harmonic approximation. Assumes that the dipole moment components have been requested in the input to the electronic structure calculation. To be used in connection with derivative force fields and derivative property surfaces.

16.2 Advanced keywords

#2 **BarFileStorageType** !ADGA !Static

$< \ string \ = \ \texttt{ONFILE} \ > [\texttt{ONFILE}] \ [\texttt{INMEMORY}]$

Defines how the single point information should be handled during the course of the interpolation and/or fitting procedures. The current options are

> **Options**
>
> - ONFILE
>   Information will only be read into memory from the .mbar files for a single mode combination and property at any given time.
>
> - INMEMORY
>   Information pertaining to all mode combinations and properties will be read into memory from the .mbar files before the interpolation and/or fitting procedures commences.

#2 Diatomic   `!ADGA` `!Static` `!Taylor`

Special option for diatomics used to reduce the number of electronic structure calculations. It is assumed that the molecule is placed along the Z-axis.

#2 ItExpGridScalFact   `!ADGA`
$< integer = 2 >$

The integer given to this keyword will be used as exponent to the number 2. This number is used as a scaling factor for the absolute size of the grid that can come under consideration in an ADGA calculation, i.e. the maximum boundaries for the box wherein a potential calculation can take place. This scaling factor is applied as a front factor in the expression for the box boundaries: $2^{N_{\text{ItExpGridScalFact}}} \cdot 2^{N_{\text{NiterMax}}}$, where NiterMax is set by the #2 NiterMax keyword. The maximum boundaries for a given potential calculation can thus be expanded by $2^{N_{\text{ItExpGridScalFact}}}$ compared to the case were the scaling factor is set to zero.

#2 NoScreenSymPotTerms   `!ADGA` `!Static`

If a symmetry calculation is requested the potential energy surface will be fitted with symmetry adapted polynomial function. The option inhibits the screening of the polynomial function.
(Useful for debuging.)

#2 PolyInterpolationPoints   `!ADGA`
$< integer = 4 >$

Defines the number of points to be used for polynomial interpolation specifically when the ADGA evaluated the integral over vibrational density.

#2 ScreenOperCoef   `!ADGA` `!Static`
$< string = $ `ON` $ > $ [ON] [OFF]
$< real = 1.0 \cdot 10^{-14} >$

Turns operator screening on or off when writing operator terms to file by specifying either ON or OFF in accordance with a screening threshold in case screening is turned on.

#2 GLQuadraturePoints

$< integer \ = \ 12 \ >$

Specifies the number of points used in numerical integration of potential energy operator terms by means of Gauss-Legendre quadrature. Note that the n-point Gaussian quadrature rule always applies, i.e. numerical integration can be considered correct for polynomials of degree $2n - 1$ or less.

(Recommended settings are in the interval of $6 - 12$.)

Chapter 17

FitBasis input options

This section details the input options available when performing fits in order to obtain analytical expressions for the potential energy or molecular property surfaces. There is no restrictions on the type of fit-basis functions to be used but no warranty is given for user-defined functions. Furthermore it should be noted that the default set of fit-basis functions contains only polynomial functions and if nothing else is specified in the input then these are automatically used in the fitting routine of `MidasCpp`.

17.1 General keywords

#3 AddFitFuncsConserv !FitBasis

Allows the ADGA to be more conservative in the addition of fit functions, compared to the standard addition scheme set by specifying the #3 FitFunctionsMaxOrder keyword. The number of fit functions will now be given as

$$N_{\text{func}}(Q_k) = \begin{cases} N_{\text{SP}} - 2, & \text{if } N_{\text{SP}} \text{ even,} \\ N_{\text{SP}} - 1, & \text{if } N_{\text{SP}} \text{ odd.} \end{cases}$$

This will in general ensure more stability in the fitting routine but can be especially helpful if many fit functions are used but only a few single points are added in each ADGA iteration.

#3 AutoFitFunctions !FitBasis

This will allow the program to carry out the linear fit with different numbers of fit-basis functions in order to establish when the best possible fit is achieved based on the root-mean-square deviation (RMSD) value.

#3 DefFromFile !FitBasis
< string >

Directs the input to a file containing all or additional fit-basis input. The format is the same as for the present section but the file should begin and end with the keywords `#0 MIDASFITINFOFILE` and `#0 MIDASFITINFOFILEEND` respectively.

#3 FitBasisEffInertiaInvMaxOrder !FitBasis
$< integer\ =\ 4$ **OR** $integer\ vector >$

The format is similar to the #3 FitBasisMaxOrder keyword and defines the maximum order of higher dimensional terms for fitting the effective inertia inverse tensor components, which is generated through the linear fitting procedure. Note that this keyword will only take any effect if the #2 CalcEffInertiaInvTens keyword is specified.

#3 FitBasisMaxOrder !FitBasis
$< integer\ =\ 12$ **OR** $integer\ vector >$

Defines the maximum order of higer dimensional terms in the potential, which are generated through the linear fitting procedure. The vector should have as many elements as the requested mode-coupling level but in case only a single integer is given, it is extended to all mode-coupling levels. The complete functional basis for higher dimensions will be generated as a direct product of the associated mono-dimensional bases.

#3 FitFunctionsEffInertiaInvMaxOrder !FitBasis
$< integer\ =\ 4\ 4\ 4\ 4$ **OR** $integer\ vector >$

The format is similar to the #3 FitFunctionsMaxOrder keyword and defines the maximum order of the one-mode fit-basis functions to be used in the linear fitting of effective inertia inverse tensor components. Note that this keyword will only take any effect if the #2 CalcEffInertiaInvTens keyword is specified.

#3 FitFunctionsMaxOrder !FitBasis
$< integer\ vector\ =\ 12\ 12\ 12\ 12$ **OR** $integer >$

Defines the maximum order of the one-mode fit-basis functions used in the linear fit of single points. If a vector of integers is given, then this should have as many elements as there are vibrational modes in the molecular system under consideration. Alternatively, if only a single integer is given, then this number is used to define the maximum order for all one-mode fit-bases. Note that the actual maximum order of fit-basis functions in a given iteration depends on the number of available single points in the potential. The number of fit functions is given as

$$N_{\text{func}}(Q_k) = \begin{cases} N_{\text{SP}}, & \text{if } N_{\text{SP}} \text{ even,} \\ N_{\text{SP}} - 1, & \text{if } N_{\text{SP}} \text{ odd.} \end{cases}$$

This applies until the maximum order of the fit functions is reached.

#3 GenFitBasisDef !FitBasis

Directs the input to the general fit-basis handler, see section 18.

#3 IoLevel `!FitBasis`

$< integer\ =\ 0\ >$

Indicates the IoLevel of the FitBasis block.

#3 Name `!FitBasis`

$< string\ >$

Identify the fit-basis by a name.

17.2 Advanced keywords

#3 Constants `!FitBasis`

$< string\quad real\ >$

Gives a series of constant names and values in the format "const_name 1.1213", the values are read as strings and real value.

#3 FitAllMCIterGrids `!FitBasis`

Specifying this keyword means that all grids for each mode combination will be fitted at each ADGA iteration, regardless of new points have been introduced into the grids. Note that all grids will be refitted at a given ADGA iteration if extrapolation/interpolation procedures using derivative information is used.

#3 FitMethod `!FitBasis`

$< string\ =\ \mathrm{SVD}\ >$ `[SVD]` `[NORMALEQ]`

Defines the method used for extracting the least squares parameters in the linear fitting. Options are `NORMALEQ` for which the solution is set via normal equations or `SVD` which make use of the singular value decomposition.

#3 Functions `!FitBasis`

Read a set of functions in the same format as #4 FitFunctions.

#3 NthreadsFit `!FitBasis`

$< integer\ =\ \texttt{--numthreads}\ >$ `[0 to --numthreads]`

Indicates the number of threads used locally in the fitting routine. If this keyword is not specified, then the program defaults to the number of threads specified at the command-line level when commencing a calculation. A value of 0 will run as many threads as has been specified by the `--numthreads` command-line option.

#3 **SVDFitCutOfThr**   !FitBasis

      $< \; real \;\; = \;\; 1.0 \cdot 10^{-13} \;\; >$

Specifies the relative cut off threshold for singular values in relation to the largest singular value during SVD fitting.

Chapter 18

GenFitBasisDef input options

This section details the input options available to the general fit-basis handler.

18.1   General keywords

#4 FitFunctions   !GenFitBasis

$< string >$

Defines the complete set of fit-basis functions, which are to be used in the linear fitting procedure.

#4 Modes   !GenFitBasis

$< integer\ vector >$

Defines the vibrational modes that should be linearly fitted with the set of fit-basis functions specified under the #4 FitFunctions keyword. Note that the modes are counted from 0 to $3N - 7(6)$ in accordance with appearance in the Molecule.mol file.

#4 OptFunc   !GenFitBasis

$< string >$

Give a function which is used to optimize the parameters of the functions given under #4 FitFunctions the format of the input is given in an example. First a function definition is specified, the name, "F", is not required here what is needed is a list of variables in a parenthesis separated by commas. Second a line giving the functional which should be fitted non linearly using the parameters given. Lastly a set of starting guesses where the first is irrelevant since we are not optimizing with respect to Q.
(Typical values for dissociation energy can be found in the range of 0.05 to 0.4.)

> **Example for OptFunc**
>
> ```
> #2 OptFunc
> F(Q,alpha,De) De*(1-EXP(-alpha*Q))^2 (0.0,0.1,0.1)
> ```

#4 OptFuncThr   !GenFitBasis

$< real\ =\ 1.0 \cdot 10^{-8} >$

This number determines the gradient convergence for the non-linear optimization of parameters in the function specified under #4 OptFunc.

#4 Props !GenFitBasis

< *string vector* >

Defines the properties that the set of fit-basis functions given under the #3 GenFitBasis-Def keyword should be applied to in the linear fitting procedure. The strings should match the common Midas property names, e.g. energy, inertia or dipole, as can be found in the last word, i.e. after the last "_", of the descriptor given in the `midasifc.propinfo` file.

Chapter 19

System input options

This section details the input options available to both defining or modifying a molecular system. The #2 SystemDef keyword block is used to define individual subsystems of the full molecular system but if only one subsystem is defined then this is treated as the full system. The #2 ModSys keyword block is used to specify in which manner the coordinates of a given subsystem should be modified.

19.1 General keywords

#1 System

Read System input until next level #1 input key.

#2 ModSys

Read definitions for modifying a molecular system.

#3 DefineSymmetry

Reads in a line with two strings, the first specify a point group and the second specify the primary axis. This keyword indicates which point group the vibrational coordinates should belong to, which can be either the same point group as the nuclear coordinates or a point group of lower symmetry. The default point group for a molecule not in a normal coordinate representation is the $extupC_1$ point group.
(Note that this keyword will only have an effect if the ROTCOORD scheme is selected under the #3 ModVibCoord keyword.)

#3 ModVibCoord   !ModVibCoord
$< string >$ [FREQANA] [FALCON] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

Indicates the scheme with which to modify the vibrational coordinates, see sections   21 , 22   for more details.

#3 RemoveGlobalTr

114

Remove the global translational and rotational degrees of freedom before entering the selected scheme under the #3 ModVibCoord keyword. This is useful for all global modification schemes, e.g. ROTCOORD or FREQANA, as it ensures that purely vibrational degrees of freedom are present.

#3 SysNames
$< integer >$
$< string >$

Indicates the number of subsystem names, followed by lines giving the names.

#2 SystemDef

Read definitions for a molecular system until next #2 SystemDef or #2 ModSys keyword is encountered. Note that several systems can be defined at the same time in which case these will be treated as individual subsystems of the full systems. If only one subsystem is specified then this is regarded as the full system.

#3 AddLocalTransRot

Switches on the addition of the local translations and rotations for each subsystem.

#3 CalcModes
$< string = NO >$ [NO] [CART] [LOCALTRANSROT]

Specifies if and which initial coordinates should be generated for the system. NO indicates the no additional coordinates should be generated, CART means that Cartesian coordinates should be generated, while LOCALTRANSROT means that translational and rotational coordinates should be generated.
(The modes are only initialized for groups that contain ACTIVE atoms.)

#3 FragmentType
$< string = PREDEFINED >$ [ATOM] [PREDEFINED]

Defines the partition of the total molecular system into subsystems. ATOM indicates that each atom is a subsystem, while PREDEFINED indicates that the specification of subsystems is found in the file given by the #3 MoleculeFile keyword.

#3 MoleculeFile
$< string >$ [MIDAS] [MOLDEN] [TURBOMOLE] [ORCAHESS] [SINDO]
$< string >$

The first string indicates in which format the molecule file is given, while the second string indicates the path and name of the file in which molecular structure information is provided.

Information given in the molecule file depends on the actual calculation at hand but usually involdes the reference geometry in Cartesian coordinates, vibrational frequencies and vibrational coordinates.

(The standard setting should be MIDAS for all purposes regarding surface generation. For more information on the MIDAS molecule format see A.1.)

Example for MoleculeFile

```
#3 MoleculeFile
   MIDAS                // Read input in MidasCpp molecule format (.mmol).
   mymolecule.mmol      // Read file 'mymolecule.mmol' in main-directory.
```

#3 Name

< string >

Indicates the name of the system or subsystem for later reference.

Chapter 20

Falcon input options

This section details the input options for the iterative merge of subsystems, which is known as the FALCON algorithm.

20.1 General keywords

#3 ModVibCoord   !ModVibCoord

    $< string >$ [FREQANA] [FALCON] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

When called with FALCON as argument, the following keywords are available.

#4 HessType   !FALCON

    $< string = $ NO $ >$ [NO] [GIVEN] [CALC]

Defines the way the Hessian is obtained for the mode optimization. Current possibilities are: i) Hessian is not used (NO) and modes are not optimized, ii) the full Cartesian Hessian is provided in the Molecule.mmol file (GIVEN), and iii) the reduced Hessian is calculated using FreqAna (CALC). If modes are not to be optimized, what can be the case in, e.g., a Falcon preparatory run, the default option NO should be used.

#4 MaxSubSysForFusion   !FALCON

    $< integer = \infty >$

Reads in the maximum number of initial subsystems, which can be fused in the Falcon algorithm. Reaching this threshold, the procedure is considered as converged.

#4 MaxSubSysForRelax   !FALCON

    $< integer = \infty >$

Reads in the maximum number of initial subsystems in a fusion groups to be relaxed.

#4 MinCouplingForFusion   !FALCON

    $< real = 0.0 >$

Reads in a value specifying an upper bound to the coupling estimates for fusion. Reaching this threshold both constituting subsystems are considered converged.

#4 MinCouplingForRelax   !FALCON
$< real = 0.0 >$

Reads in a value specifying an upper bound to the coupling estimate for relaxation of the fusion group.

#4 ModeOptType   !FALCON
$< string = NO >$ [NO] [HESSIAN]

Sets the way the (local) modes are optimized after separating out the translations and rotations. Current choices are: i) no further modification of the coordinates is performed (NO), ii) performing diagonalization of the reduced Hessian (HESSIAN). In the latter case, #4 HessType has to be set.

#4 WriteIncrInput   !FALCON
$< integer = 0 >$

Sets up the input for a double incremental calculation. Reads in the number for a maximal fragment combination level to be considered.

20.2 Advanced keywords

#4 Coupling   !FALCON
$< string = DIST >$ [DIST] [DISTACT]

Reads in one line with the mode coupling type to be used. The current options are: i) the closest distance between atoms (DIST), ii) the closest distance to an active atom (DISTACT).

#4 Degeneracy   !FALCON
$< real = 0.001 >$

Reads in the treshold for considering the coupling estimates degenerate in the fusion process. (The value of 0.001 is usually used in combination with the distance-based coupling)

#4 MaxDistForConn   !FALCON
$< real = \infty >$

Reads in the threshold for the maximum distance (in bohr) between the closest atoms in two subsystems to consider them connected. Reaching this threshold, the procedure is considered as converged.

#4 PrepRun

Performs only a preparation run, where subsystems are formed but no operations with modes are carried out.

Chapter 21

FreqAna input options

This section details the input options for performing (semi-numerical gradient-based) frequency analysis to obtain the harmonic frequencies and normal modes, see section 1.3. This is done using electronic single point calculations with input as described in section 8 in order to calculate gradients for the displaced nuclear configurations. The gradients are subsequently used to calculate the Hessian, which is afterwards diagonalized.

If #4 Targeting is enabled the program uses the modes in the MoleculeInfo file (i.e. those listed under the keywords `#1VIBCOORD -> #2COORDINATE` as described in A.1) as targets when iteratively solving mass-weighted Hessian eigenvalue problem.

21.1 General keywords

#4 DispFactor   `!ModVibCoord`

$< real = 0.01 >$

The displacement factor used when creating the displaced configurations. (δ in section 1.3.)

#4 DumpInterval   `!ModVibCoord`

$< integer = 10 >$

Interval for dumping property files to disc (as a safety in case of crash).

#4 ItEqBreakDim   `!ModVibCoord`

$< integer = 1000 >$

Break dimension for the iterative solver. Only applicable if #4 Targeting is enabled.

#4 ItEqEnerThr   `!ModVibCoord`

$< real = 10^{-8} >$

Convergence threshold for the relative change in energy for the iterative solver (ϵ_{energy} in section 1.3). Only applicable if #4 Targeting is enabled.

#4 ItEqMaxIt !ModVibCoord

$< integer \;=\; 1000\; >$

Maximum number of iterations for the iterative solver. Only applicable if #4 Targeting is enabled.

#4 ItEqResThr !ModVibCoord

$< real \;=\; 10^{-8}\; >$

Convergence threshold for the norm of the residual for the iterative solver (ϵ_{res} in section 1.3). Only applicable if #4 Targeting is enabled.

#4 MolInfoOutName !ModVibCoord

$< string \;=\;$ `MoleculeInfo_FreqAna.mmol` $>$

Name of the output file that the resulting MoleculeInfo (containing eigenfrequencies and -vectors) should be written to.

#4 NonSymmetricHessian !ModVibCoord

If this keyword is set the (reduced) Hessian is not symmetrized before diagonalization.

#4 OverlapNMax !ModVibCoord

$< integer \;=\;$ unlimited $>$

Maximum number of eigenvectors included in the overlap sum for each target, i.e. it can override the specification set under #4 OverlapSumMin ($n_{\mathrm{overlap,max}}$ in section 1.3). Only applicable if #4 Targeting is enabled.

#4 OverlapSumMin !ModVibCoord

$< real \;=\; 0.8\; >$

When expanding the reduced subspace the iterative solver uses as many of the eigenvectors of the last iteration as is required to achieve a minimum accumulated overlap of this much for *each* target ($\epsilon_{\mathrm{overlap}}$ in section 1.3). Eigenvectors are allowed to occur in the overlap sums for several targets. Only applicable if #4 Targeting is enabled.

#4 ProcPerNode !ModVibCoord

$< integer \;=\; 1\; >$

Number of processes per node, i.e. maximum number of single-points to run simultaneously. (Currently only support for running on 1 node.)

#4 RotationThr !ModVibCoord

$< \; real \; = \; 10^{-6} \; >$

The rotation/symmetry threshold used in singlepoint calculations when trying to determine the rotation that was done to the molecular structure by the electronic structure program(which is necessary for properly rotating the gradients back to the `MidasCpp` reference frame).

(If `MidasCpp` is unable to determine the rotation it can in some cases help to loosen this threshold slightly (e.g. by an order of magnitude).)

#4 SinglePointName !ModVibCoord

$< \; string \; = \; \texttt{SP\_FreqAna} \; >$

Name of singlepoint calculation. Must correspond to a name given under #2 Name in #1 SinglePoint (see chapter Singlepoint input options).

#4 Targeting !ModVibCoord

Tries to solve the Hessian eigenvalue problem iteratively using projection on a reduced space. The iterative solver tries to target the eigenvectors that have the greatest overlap with the given targets. The targets should be given as modes in the MoleculeInfo file, i.e. as coordinate vectors under the `#2COORDINATE` keyword. The keywords #4 ItEqEnerThr, #4 ItEqResThr, #4 ItEqBreakDim, #4 ItEqMaxIt, #4 OverlapSumMin, #4 OverlapNMax, only have effect if this keyword is set.

Chapter 22

RotCoord input options

This sections details the input options available for submitting the molecular subsystem to rotations of the vibrational coordinates in accordance with different optimization criteria in order to transform these to more suitable sets of vibrational coordinates.

The transformation of one general set of vibrational coordinates into another is carried out with the iterative Jacobi sweep algorithm in combination with numerical minimum search techniques. A combination of the numerical Fourier series expansion and Newton minimization techniques is applied in order to obtain an optimal rotation angle with regards to an optimization criterion.

22.1   General keywords

#4 ConvThr
!ModVibCoord

$< real = 1.0 \cdot 10^{-6} >$

Defines the convergece threshold for the vibrational coordinate rotation scheme with regards to the energy or property being optimized.

#4 FreqScreen
!ModVibCoord

$< integer = 100 >$

Defines that rotations in the Jacobi sweep algorithm can be disregarded or screened away if the mode-pairs to be rotated have a difference in frequencies that are larger than the number given to this keyword.
(The maximum frequency difference is assumed to be provided in units of reciprocal centimeters.)

#4 FunctionType
!ModVibCoord

$< string = GRID >$ [GRID]

Specifies the type of function for vibrational coordinate rotation. GRID means that the function will be treated in a grid-based fashion.

#5 FourierPt

$< integer = 2 >$

Defines the number p, which gives the number of points $2p + 1$ to be used in the Fourier series expansion step of the optimization.

#5 NewtonConv
$< real\ =\ 1.0 \cdot 10^{-6}\ >$

Defines the convergence threshold for the Newton minimization step of the optimization. (Note that the maximum number of iterations for the Newton minimization is hard-coded to 10.)

#5 OptParam
$< string\ =\ \mathrm{VALUE}\ >$ [GRADHESS] [GRADIENT] [GRADIENTNORM] [VALUE]

Specifies the parameter type with which to evaluate the changes to the optimization criterion. GRADHESS means that convergence checks are carried out by using the largest element of the optimization criterion gradient, while simultaneously checking that the Hessian is positive. GRADIENTNORM means that convergence checks are carried out by using the norm of the optimization criterion gradient. GRADIENT means that convergence checks are carried out by using the largest element of the optimization criterion gradient. VALUE means that convergence checks are carried out by using the the optimization criterion value directly.
(The GRADHESS option is generally recommended but will not work if the LOCALIZATION option is chosen under the #4 Measure keyword is chosen.)

#5 PlotOnly

Indicates that a single sweep should be carried out but without doing any rotations. The property chosen as optimization property can thereby be plotted as a function of rotation angle.

#5 PlotPoints
$< integer\ =\ 0\ >$

Defines the number r, which gives the number of points $2r + 1$ to be used for plotting the property chosen as optimization property as a function of rotation angle.
(If zero is given as input to this keyword, then no plots will be supplied.)

#4 Measure   !ModVibCoord
$< string >$ [HYBRID] [NONHYBRID]
$< string\ \ real >$ [LOCALIZATION] [OPTIMIZATION]
$[< string\ \ real >]$ [LOCALIZATION] [OPTIMIZATION]

Specifies the type of optimization criterion that should be used to check the convergence for the vibrational coordinate transformation. HYBRID means that the optimization criterion should be a combination of the two different measures given on the two following lines, which will lead to the generation of HOLCs. This furthermore requires the input of the weighting between the different measures. NONHYBRID means that the optimization

criterion will only be a single measure given on the following line, which will lead to the generation of OCs or LCs. This option does not requires the use of weights. OPTIMIZATION means that a minimization of the VSCF energy with respect to variations in both the modals and vibrational coordinates should be used. LOCALIZATION means that minimization with respect to a geometrical localization criterion should be used.

#5 CalulationType
$< string =$ VSCFCALC $>$ [GROUNDSTATE] [HARMONICGROUNDSTATE] [STATEAVERAGE]

Specifies the type of VSCF calculation to provide the energy. HARMONICGROUNDSTATE means that only harmonic contributions to the ground state energy will be used, which makes for a very fast calculation as no VSCF mean-field needs to be constructed. GROUNDSTATE means that only the VSCF ground state energy will be calculated. STATEAVERAGE means that the VSCF energy will be an average over multiple state energies, where both ground state energy and/or excited state energies will be calculated.
(The HARMONICGROUNDSTATE option should only be used if the vibrational coordinates are to be rotated in a purely harmonic potential.)

#5 Name
$< string >$

Specifies the name of the VSCF calculation.

#4 MinRotAng
!ModVibCoord

$< real = -1.0 >$

Defines the minimal rotation angle for which to carry out rotations of mode-pairs. If a rotation angle is smaller than the number given to this keyword, then the corresponding rotation will not be carried out.

#4 PrintRotMatToFile
!ModVibCoord

$< string >$

Specifies a file name which will be given to a file that contains the final rotation matrix.

#4 RandomStart
!ModVibCoord

$< integer = 0 >$

Defines the number of sweeps in which mode-pairs are randomly rotated that are to be performed before the commencing with vibrational coordinate optimization.

#4 RotationSuffix
!ModVibCoord

$< string = \_rotated >$

Specifies the suffix appended to all files which contains rotated values, e.g. .mop and .mol files.

#4 Sweeps   !ModVibCoord

$< integer\ =\ 6\ >$

Defines the maximum number of sweeps that can be carried out in the Jacobi sweep algorithm.

Chapter 23

Machine Learning options

23.1   General keywords

#1 MLTask

Read General input until next level #1 input key.

#2 CoVarAlgo   !MLEARN
$< string = $ CHOL $>$ [CHOL] [LU] [SVD] [BKD]

This keyword specifies which algorithm is used to decompose the co-variance matrix in GPR
and related methods.

> **Options**
>
> - **CHOL** Cholesky decomposition. This is usually the fastest option and robust in
>   many cases.
>
> - **LU** LU decomposition. This option is less efficient then using a Choleksy decom-
>   position and also less robust and thus deprecated.
>
> - **SVD** SVD. This option is helpful if the training data contains redundancies and
>   a Cholesky decomposition fails. However, SVD is also quite costly.
>
> - **BKD** Bunch-Kaufmann decomposition. It has comparable costs to a Cholesky
>   decomposition, but also works for indefinite matrices.

#2 CoordFile   !MLEARN
$< string >$

Specifies the location of the file containing the coordinates for which a ML algorithm should
be used for prediction or classification.

#2 CoordType   !MLEARN
$< string >$ [MINT] [XYZ] [ZMAT] [XYZCENTER] [DIST] [EIGDIST]

Specifies the coordinate type used to interpret the structures you specified via the xyz files.
(It is recommended to use internal coordinates MINT.)

< *string* >

Specifies the location of the file containing the database for training a ML algorithm. So far xyz files with an extension to also include gradient and hessian information are supported. An example is given below. First the usual geometry has to be given, followed by the gradient and/or hessian. Gradient information is indicated by the keyword Gradient followed by the number of lines to read this information. Each line contains the gradient for atom i in the x y z direction. The keyword hessian indicates hessian information followed by the number of lines to read this information. Each line starts with two integers and up to 6 floating point numbers. The first integer indicates the row of the hessian and the second integer the line currently printed for this row.

Example for Database

```
3
Energy =      -76.3587709915
 O  0.000000000  0.000000000  0.391495816
 H -0.757294103  0.000000000 -0.195747908
 H  0.757294103  0.000000000 -0.195747908
Gradient 3
   -0.000000000 -0.000000000  0.000476105
   -0.000302012 -0.000000000 -0.000238053
    0.000302012 -0.000000000 -0.000238053
Hessian 18
1 1  0.364873177  0.000000000  0.000000000 -0.182436588  0.000000
1 2 -0.141470455 -0.182436588  0.000000000  0.141470455
2 1  0.000000000  0.000000000  0.000000000  0.000000000  0.000000
2 2  0.000000000  0.000000000  0.000000000  0.000000000
3 1  0.000000000  0.000000000  0.244239667 -0.108365351  0.000000
3 2 -0.122119834  0.108365351  0.000000000 -0.122119834
4 1 -0.182436588  0.000000000 -0.108365351  0.199038732  0.000000
4 2  0.124917903 -0.016602144  0.000000000 -0.016552552
5 1  0.000000000  0.000000000  0.000000000  0.000000000  0.000000
5 2  0.000000000  0.000000000  0.000000000  0.000000000
6 1 -0.141470455  0.000000000 -0.122119834  0.124917903  0.000000
6 2  0.115911550  0.016552552  0.000000000  0.006208283
7 1 -0.182436588  0.000000000  0.108365351 -0.016602144  0.000000
7 2  0.016552552  0.199038732  0.000000000 -0.124917903
8 1  0.000000000  0.000000000  0.000000000  0.000000000  0.000000
8 2  0.000000000  0.000000000  0.000000000  0.000000000
9 1  0.141470455  0.000000000 -0.122119834 -0.016552552  0.000000
9 2  0.006208283 -0.124917903  0.000000000  0.115911550
```

#2 **DumpIcoordDef** !MLEARN

< *string* = 0 >

Specifies a file for saving the definition for internal coordinates `MidasCpp` automatically detected. This is more a debug option or can be used to generate an initial input for the #2 ReadIcoordDef keyword. For a description of the generated file format take a look at the #2 ReadIcoordDef keyword.

#2 **FuncEps** !MLEARN

$< real = 10^{-3} >$

Specifies the stopping criterion w.r.t to the change in the marginal likelihood in the tuning of hyper parameters. If ML training is performed after an ADGA PES construction, this keyword has only an effect if #2 SaveTrainData and #2 SaveDispGeom is present.

#2 **GradEps** !MLEARN

$< real = 10^{-2} >$

Specifies the stopping criterion w.r.t to the gradient norm in the tuning of hyper parameters. If ML training is performed after an ADGA PES construction, this keyword has only an effect if #2 SaveTrainData and #2 SaveDispGeom is present.

#2 **Hopt** !MLEARN

$< boolean = true >$

Specifies if the hyper parameters in e.g. GPR should be optimized. The default is true, but with a good set of hyper parameters you might want to avoid this step.

#2 **IoLevel** !MLEARN

$< integer = 0 > [< integer >]$

Indicates the IoLevel (print level) for Machine Learning modules.

#2 **Kernel** !MLEARN

$< string >$

Specifies the Kernel used in the Machine Learning step performed. The kernel starting with SC (specific coordinate) use a specific length scale l_i parameter for each coordinate. These kernel can better adapt to the different nature of internal coordinates, but the hyper parameter optimization is usually more difficult and takes longer.

- SCEXP

$$k_{\mathbf{xy}} = \sigma_n^2 \exp\left(-0.5 \sum_i^d \frac{(x_i - y_i)^2}{l_i^2}\right)$$

- SC52

$$k_{\mathbf{xy}} = \sigma_n^2 \left(1 + \sqrt{5} \sum_i^d \frac{|x_i - y_i|}{l_i} + 5 \sum_i^d \frac{(x_i - y_i)^2}{3l_i^2}\right)$$
$$\times \exp\left(-\sqrt{5} \sum_i^d \frac{|x_i - y_i|}{l_i}\right)$$

- SC32

$$k_{\mathbf{xy}} = \sigma_n^2 \left(1 + \sqrt{3} \sum_i^d \frac{|x_i - y_i|}{l_i}\right) \exp\left(-\sqrt{3} \sum_i^d \frac{|x_i - y_i|}{l_i}\right)$$

- SQEXP

$$k_{\mathbf{xy}} = \sigma_n^2 \exp\left(-0.5 \frac{\|\mathbf{x} - \mathbf{y}\|^2}{l^2}\right)$$

- MATERN52

$$k_{\mathbf{xy}} = \sigma_n^2 \left(1 + \frac{\sqrt{5} \|\mathbf{x} - \mathbf{y}\|}{l} + \frac{5 \|\mathbf{x} - \mathbf{y}\|^2}{3l^2}\right) \exp\left(-\sqrt{5} \frac{\|\mathbf{x} - \mathbf{y}\|}{l}\right)$$

- MATERN32

$$k_{\mathbf{xy}} = \sigma_n^2 \left(1 + \frac{\sqrt{3} \|\mathbf{x} - \mathbf{y}\|}{l}\right) \exp\left(-\sqrt{3} \frac{\|\mathbf{x} - \mathbf{y}\|}{l}\right)$$

- OUEXP

$$k_{\mathbf{xy}} = \sigma_n^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|}{l}\right)$$

- POLYNOM

$$k_{\mathbf{xy}} = \sigma_n^2 \left(\alpha \cdot \mathbf{x} \cdot \mathbf{y} + c\right))^d$$

- PERIODIC

$$k_{\mathbf{xy}} = \sigma_n^2 \exp\left(-2 \frac{\sin^2\left(\pi \frac{\|\mathbf{x} - \mathbf{y}\|^2}{\theta}\right)}{l^2}\right)$$

#2 MaxIter !MLEARN

$< integer = 200 >$

Specifies the number of iterations used in optimizing hyper parameters in GPR. If you use this option in combination with a PES construction it only an effect if #2 SaveTrainData and #2 SaveDispGeom is present.

#2 Name !MLEARN

$< string >$

Specifies a name for the ML task you want to perform and which can be used as identifier to reference to this tasks from other parts of the program

#2 Noise
$< real = 1.0 \cdot 10^{-8} >$

Specifies the noise imposed on the input data. The noise acts as regularization term and can generate a smoother predictor.

#2 OnlyICoord

Activates that only internal coordinates should be computed and nothing else. This option is mainly meant for analysis and debugging.

#2 PredictMode
$< integer = 0 >$

Specifies if the derivative order the GPR should try to predict. For 0 it only predicts energies, for 1 energies and gradients and for 2 energies, gradients and Hessians. Note that no higher order derivatives are supported and that this option only works with a squared exponential kernel.

#2 ReadCoVar
$< string >$

Specifies the location of a (decomposed) Co-Variance matrix, which can be used in GPR to bypass the construction and inversion of it. This is avoids recomputation of the Co-variance matrix for each single point is therefore the efficient choice. However only simple sanity checks are performed to verify that it is a valid Co-variance matrix for this case.

#2 ReadIcoordDef
$< string = 0 >$

Specifies a file for reading a set of definition for internal coordinates, which should be used by `MidasCpp`. This option avoids the automatic detection algorithm for defining the internal coordinates. An example for such a definition file is given below. The itype specifies the type of the coordinate. 1 bond length, 2 bond angle, 3 out of plane bending, 4 dihedral. val gives an initial value for the coordinate, which will later be updated. iat specifies the atom position and the vector atoms which atoms are involved in forming the internal coordinate.

- For the bond angle the sequence $i\ j\ k$ indicates that i is the central atom connected to j and k
- For the out of plane bending the list $i\ j\ k\ l$ describes for example the bending of i out of the plane spanned by $j\ k\ l$ where i is connected to l.

– For the dihedral angle the sequence i j k l describes the angle between the planes i j k and j k l.

Example for ReadIcoordDef

```
itype 1 val  1.876647e+00 iat 1 atoms 0 1
itype 1 val  1.876647e+00 iat 2 atoms 0 2
itype 2 val  2.297390e+00 iat 2 atoms 0 1 2
itype 1 val  2.601879e+00 iat 3 atoms 0 3
itype 2 val  1.992898e+00 iat 3 atoms 0 1 3
itype 3 val -3.141593e+00 iat 3 atoms 0 2 3 1
```

#2 SaveCoVar   !MLEARN

< string >

Specifies the location to save a (decomposed) Co-Variance matrix, which can later be reused using the #2 ReadCoVar keyword.

#2 GPR   !MLEARN

Activates that Gaussian Process Regression (GPR) should be performed for the coordinates given in #2 CoordFile. The GPR is trained using the data specified in #2 Database

Chapter 24

ODE-integration options

This section describes the input options for the `MidasCpp` ODE library. This does not function as a separate input block, but may be used at different input levels in other blocks (hence the $< N >$ in the keywords).

24.1 General keywords

#**<N> Type**
> $< string\ vector\ =\ \mathrm{MIDAS} <\mathrm{stepper}> >$ [`MIDAS DOPR853`] [`MIDAS DOPR5`] [`GSL <gsl-stepper>`]

> Set type of ODE driver and stepping routine. If only one argument is provided, the MIDAS driver is used with the given stepper. For GSL, only the steppers that do not require the Jacobian are available. Moreover, the GSL integrators are (of course) only available if MidasCpp is linked with GSL.
> (Use 'MIDAS DOPR853' as a good general-purpose integrator.)

#**<N> Name**
> $< string >$

> Assign a name to the integrator.

#**<N> IoLevel**
> $< integer\ =\ 1\ >$

> IO level of the integrator. Goes from no output to detailed flow of the steps taken.

#**<N> TimeInterval**
> $< real\ vector\ =\ 0\ 1\ >$

> Set the time interval for integration. Note that the code has only been tested for intervals starting at zero.

#**<N> MaxSteps**
> $< integer\ =\ -1\ (\mathrm{no\ limit})\ >$

Maximum number of steps in the integrator. -1 means no limit.

#<**N**> **Tolerance**
      < *real vector* = 1.e-16 1.e-16 >

Set absolute and relative error tolerance.
(The default is too strict for most cases.)

#<**N**> **OutputPoints**
      < *integer* = 20 >

Number of equidistant time points where specific properties (such as auto-correlation functions in TDH) are calculated. If the argument is -1, output will be performed at all steps, i.e. the points will NOT be equidistant!

#<**N**> **InitialStepSize**
      < *real* >

Initial step size for the integration. If this is not set, the integrator will make a qualified guess.
(The integrator's guess is good enough in most cases. Only set this keyword if you run into problems with the default.)

#<**N**> **DatabaseSaveVectors**

Save the $y(t)$ vectors (in memory) for all equidistant steps for later analysis.
(Only use this if you need the vectors for analysis at the end of the calculation.)

#<**N**> **FixedStepSize**
      < *real* = 1.e-2 > [AUTO] [<step-size>]

Use a fixed step size for the integration. If 'AUTO' is given as argument, the integrator will guess a 'suitable' step size (which is not always good).

#<**N**> **SaveSteps**

Save info on step sizes and scaled errors and write this to disk at the end of the integration.

24.2   Advanced keywords

#<**N**> **StepSizeControl**
      < *real vector* = Individual defaults set for each stepper >

Set constants for step-size control. Reads in 5 numbers: α, β, $safe$, $minscale$, $maxscale$. For

step-size control, the step size is scaled by: $\min(maxscale, \max(minscale, safe \times err_n^{-\alpha} \times err_{n-1}^{-\beta}))$. Here err is the scaled error. Note that there is more going on than this in the step-size controller. E.g. special action is taken after rejected steps, etc.

(Just use the defaults.)

#<N> **MinStepSize**
$< \; real \; = \; 0 \; >$

Minimum allowed step size. Integrator will throw an error if a smaller step is taken.

#<N> **MidasOdeDriverMaxFailedSteps**
$< \; integer \; = \; 3 \; >$

Number of times in a row a step can fail (the new $y(t)$ vector is not accepted) before the integrator throws an error.

#<N> **ErrorType**
$< \; string \; = \; \mathrm{MEAN} \; >$ [MEAN] [MAX] [SCALEDMAX]

Type of error estimate. This is only used for the 'MIDAS' driver.

(Do not use 'SCALEDMAX' (max error divided by number of equations) unless you have good reason for doing so.)

#<N> **GSLErrorScaling**
$< \; real \; vector \; = \; 1 \; 0 \; >$

Weight of errors in $y(t)$ and $y'(t)$, respectively. See the GSL `odeiv2` documentation for more information.

#<N> **NoFSAL**

Disable FSAL property (for the integrators that use FSAL), i.e. always calculate all derivatives.

(Only use this as a debug option.)

Chapter 25

Tensor-decomposition options

This section describes the input options for the `MidasCpp` tensor-decomposition library. This does not function as a separate input block, but may be used at different input levels in other blocks (hence the $< N >$ in the keywords).

25.1   General keywords

#<N> Type

$< string\ vector\ =\ \mathrm{NOPREPROCESSOR} <\mathrm{driver}> <\mathrm{fitter}> >$ [NOPREPROCESSOR/C2T FIXEDRANKCP/FINDBESTCP ALS/ASD/NCG/NOFITTER]

Get the names of the preprocessor, the driver, and the fitter used for tensor decomposition and recompression. The preprocessor is turned off as default, so only two arguments are required.
(Use 'C2T FINDBESTCP ALS' for fast recompressions with error control.)

#<N> Name

$< string >$

Assign a name to the decomposer.

#<N> IoLevel

$< integer\ =\ 1\ >$

IO level for decomposer.
(The default is fine unless you want detailed output.)

#<N> CpRank

$< integer\ =\ 1\ >$

Set rank for FIXEDRANKCP driver.

#<N> CpAlsThreshold

$< real\ =\ 1.e\text{-}12\ >$

Set threshold for CP-ALS algorithm.

#<N> **CpAlsRelativeThreshold**
   $< real = 0 \text{ (turned off) } >$

   Set threshold for CP-ALS algorithm relative to requested accuracy of the CP decomposition.
   (This option makes little sense with FIXEDRANKCP driver, since the rank cannot be adjusted to obtain a specific accuracy of the fit.)

#<N> **CpAlsMaxIter**
   $< integer = 100 >$

   Maximum iterations for CP-ALS algorithm.

#<N> **FindBestCpMaxRank**
   $< integer = 5 >$

   Maximum allowed rank for FINDBESTCP driver.

#<N> **FindBestCpStartingRank**
   $< integer = 1 >$

   Start FINDBESTCP driver at this rank.

#<N> **FindBestCpRankIncr**
   $< integer = 1 >$

   Rank increment in FINDBESTCP driver for fitting a tensor to a given accuracy.
   (Set it low for finding the lowest possible rank. However, this may be inefficient.)

#<N> **FindBestCpAdaptiveRankIncr**

   Set the rank increment in FINDBESTCP driver for a tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ to $\Delta R = \min_d I_d(D-2)$. This choice gives good performance in the CP-VCC transformer.
   (This works fine as an automatic guess - especially if the C2T preprocessor is used.)

#<N> **FindBestCpResThreshold**
   $< real = 1.\text{e-}2 >$

   Requested absolute accuracy of CP fit in FINDBESTCP driver.

#<N> **Guesser**
   $< string = \text{RANDOM} >$ [RANDOM] [CONSTVAL] [SVD] [SCA]

   Set the policy for generating starting guesses for CP decompositions.
   (Use RANDOM initialization.)

#<**N**> **GuessIncrementer**
$< string = \text{FITDIFF} >$ [RANDOM] [REPLACE] [CONSTVAL] [SCA] [FITDIFF] [STEPWISEFIT]

Set the policy for updating a tensor to a higher rank during CP decompositions.
(FITDIFF is recommended in most cases.)

#<**N**> **CpNcgThreshold**
$< real = \text{1.e-10} >$

Set threshold for CP-NCG algorithm.

#<**N**> **CpNcgRelativeThreshold**
$< real = 0 \text{ (turned off)} >$

Set threshold for CP-NCG algorithm relative to requested accuracy of the CP decomposition.
(This option makes little sense with FIXEDRANKCP driver, since the rank cannot be adjusted to obtain a specific accuracy of the fit.)

#<**N**> **CpNcgMaxIter**
$< integer = 100 >$

Maximum iterations for CP-NCG algorithm.

#<**N**> **C2TSvdThreshold**
$< real = \text{1.e-10} >$

Set accuracy of intermediate Tucker representation used in the C2T preprocessor.

#<**N**> **CpNcgType**
$< string = \text{HS} >$ [HS (Hestenes-Stiefel)] [FR (Flecher-Reeves)] [PR (Polak-Ribiere)] [FP (FR-PR)] [DY (Dai-Yuan)] [HZ (Hager-Zhang)]

Set type of NCG method.
(HS or HZ are usually the best for CP decomposition.)

#<**N**> **CpNcgLineSearch**
$< string = \text{WOLFE} >$ [WOLFE] [EXACT] [HZ]

Set type of NCG line search.
(EXACT is often the best.)

#<**N**> **CpAsdThreshold**
$< real = \text{1.e-10} >$

Set threshold for CP-(P)ASD algorithm.

#<N> **CpAsdRelativeThreshold**
  $< \ real \ = \ 0 \ (\text{turned off}) \ >$

  Set threshold for CP-(P)ASD algorithm relative to requested accuracy of the CP decomposition.
  (This option makes little sense with FIXEDRANKCP driver, since the rank cannot be adjusted to obtain a specific accuracy of the fit.)

#<N> **CpAsdMaxIter**
  $< \ integer \ = \ 500 \ >$

  Maximum iterations for CP-(P)ASD algorithm.

#<N> **CpAsdDiagonalPreconditioner**

  Use diagonal preconditioner with CP-(P)ASD algorithm.
  (Highly recommended.)

#<N> **PivotisedCpAsd**

  Use the CP-PASD algorithm instead of CP-ASD.
  (Highly recommended as CP-ASD does not really work.)

#<N> **CpNcgDiagonalPreconditioner**

  Use diagonal preconditioner with CP-NCG algorithm.
  (This can give quite nice results.)

#<N> **NestedCpResThreshold**
  $< \ real \ = \ 1.e\text{-}2 \ >$

  Requested absolute accuracy of CP fit in NESTEDCP driver.

#<N> **NestedCpMaxRank**
  $< \ integer \ = \ 5 \ >$

  Maximum allowed rank for NESTEDCP driver.

#<N> **NestedCpRankIncr**
  $< \ integer \ = \ 1 \ >$

  Rank increment in NESTEDCP driver for fitting a tensor to a given accuracy.

#<N> **NestedCpBalanceModeVectors**

  Balance the norms of the mode vectors of the updates in NESTEDCP driver.

#$<$**N**$>$ **NestedCpAdaptiveRankIncr**

Set the rank increment in NESTEDCP driver for a tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ to $\Delta R = \min_d I_d(D - 2)$.

#$<$**N**$>$ **NestedCpAllowRefit**

Refit the whole tensor if the NESTEDCP driver is not able to obtain a good fit within an order of magnitude.

#$<$**N**$>$ **CpAlsNoDistanceConvCheck**

Use error change instead of relative distance decrease for determining CP-ALS convergence. (Use this if low ranks are more important than speed.)

#$<$**N**$>$ **CpAlsBalanceResult**

Balance mode-vector norms of result after running CP-ALS.
(May improve numerical stability.)

#$<$**N**$>$ **Preset**
$< string >$ [VCCSOLVER] [VCCTRF]

Use a preset tailored for a given application.

25.2   Advanced keywords

#$<$**N**$>$ **CpAlsRCond**
$< real\ =\ 1.\text{e-}12\ >$

Set threshold for truncating the SVD in the linear-least-squares solver used in the CP-ALS algorithm.

#$<$**N**$>$ **CpAlsUseFitNormChange**

Check convergence of CP-ALS based on norm change in the fit.

#$<$**N**$>$ **CpAlsTikhonov**
$< real\ vector\ =\ 0.1\ 0.0\ (\text{turned off})\ >$

Read two parameters α and q. α is the Tikhonov regularization parameter. The regularization can be turned off during the iterations by multiplying α with q in each iteration.

#<N> **CpAlsRals**
> < *real vector* = 0.1 0.0 (turned off) >

Read the same input as in #<N> CpAlsTikhonov. In addition to Tikhonov regularization, use RALS where the right-hand-size of the linear system is also regularized.

#<N> **FindBestCpScaleRankIncr**

Set the rank increment in FINDBESTCP driver for a tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ to $\Delta R = \Delta R_{in}(D-2)$, where ΔR_{in} is set using #<N> FindBestCpRankIncr.
(Use #<N> FindBestCpAdaptiveRankIncr instead.)

#<N> **MatrixSvdRelThresh**
> < *real* = 1.e-2 >

Accuracy of decomposing matrices using SVD relative to requested accuracy of CP decompositions.

#<N> **LowRankSVD**

Use truncated SVD for decomposition of matrices (this is the default).

#<N> **FullRankSVD**

Do not truncate the SVD when decomposing matrices to CP format.
(Only use this if you want matrices to be represented exactly.)

#<N> **CheckSVD**

Check accuracy of SVD matrix fits.
(Only use this as a debug option.)

#<N> **DecompToOrder**

Only decompose tensors of order greater than or equal to this.

#<N> **MaxRankMaxIter**
> < *integer* = -1 (turned off) >

Set maximum iterations for CP-ALS algorithm for the fit to the highest allowed rank. This allows for spending more iterations on the final fit.

#<N> **CheckIter**
> < *integer* = -1 (turned off) >

Check the error of the fit at this iteration in CP-ALS, CP-(P)ASD, or CP-NCG. If the absolute error at this point is larger than the value given in #<N> CheckThresh the fitting is aborted.

(Does not make sense to use with FIXEDRANKCP driver.)

#<N> **CheckThresh**
    < *real* = 5.0 >

For CP-ALS, CP-(P)ASD, or CP-NCG check the error of the fit at the iteration given in #<N> CheckIter. If the error at this point is larger than the this threshold the fitting is aborted.

(Does not make sense to use with FIXEDRANKCP driver.)

#<N> **NoGuessNormScaling**

Disable scaling of starting guess by the norm of the target tensor. Also disable scaling of some types of guess updates (RANDOM and CONSTVAL).

#<N> **ResidualOverlapScaling**

Scale RANDOM and CONSTVAL updates by their overlap with the residual tensor before adding to the fitting tensor.

#<N> **NoResidualNormScaling**

Do not scale RANDOM and CONSTVAL updates by the norm of the residual tensor (use the target norm instead).

#<N> **FitDiffRelativeThreshold**
    < *real* = 1.e-2 >

Relative threshold for fitting the residual when using FITDIFF or STEPWISEFIT guess incrementers.

#<N> **FindBestCpMaxErrorIncr**
    < *integer* = 1 >

Set maximum number of times the error of the fit is allowed to increase when going to a higher rank.

#<N> **CpNcgAlphaMax**
    < *real* = 1.e5 >

Maximum step length for WOLFE line-search algorithm.

#**<N>** **CpNcgAlphaResolution**
  < real = 5.e-2 >

  Resolution of step length for WOLFE line-search algorithm.

#**<N>** **CpNcgWolfeC1**
  < real = 1.e-4 >

  C1 parameter for WOLFE line search (strong Wolfe conditions).

#**<N>** **CpNcgWolfeC2**
  < real = 1.e-1 >

  C2 parameter for WOLFE line search (strong Wolfe conditions).

#**<N>** **CpPasdMaxNormPivot**

  Use max norm to determine the pivot in CP-PASD algorithm.

#**<N>** **CpPasdBalanceModeVectors**

  Balance norms of mode vectors after each CP-PASD update.

#**<N>** **PivotisedCpAls**

  Use the pivotised version of CP-ALS.

#**<N>** **CpAlsGelsd**
  < integer = 20 >

  Use GELSD instead of GELSS for solving the least-squares problem in CP-ALS when the rank is larger than this value.
  (The default is usually fine.)

#**<N>** **CpAlsNewton**

  Use the Newton version of CP-ALS where the update to the mode matrix is computed (instead of the new matrix itself).

#**<N>** **CpAlsLineSearch**

  Use line searches together with the Newton version of CP-ALS, i.e. the updates are scaled by a factor > 1 that depends on the iteration number and tensor order.

#<N> **CpNcgGeneralSolver**

Use the general NCG implementation for CP-NCG instead of the specialized one.
(Only for testing purposes.)

#<N> **CpNcgMaxElementConvCheck**

Converge CP-NCG based on the maximum element of the gradient.

#<N> **CpNcgBalanceModeVectors**

Balance the mode-vector norms after each CP-NCG step.

#<N> **C2TSafeSvd**

Use GESVD instead of GESDD in C2T algorithm.
(Only for testing purposes.)

#<N> **DefaultResidualFitter**

Always use CP-ALS for fitting residuals for FITDIFF and STEPWISEFIT guess updates.
(Can be an improvement when using CP-PASD or CP-NCG as these methods may perform
poorly for low-rank fits.)

#<N> **HagerZhangTheta**
$< real = 2.0 >$

Theta parameter for Hager-Zhang NCG method.
(theta=2.0 is the CG_DESCENT method)

#<N> **NestedCpSafeResidualNorm**

Calculate residual norms the safe (and expensive) way in NESTEDCP driver.

#<N> **CpNcg3PGRelThreshold**
$< real = 1.e-3 >$

Set relative threshold for EXACT (3-PG) line-search algorithm used in CP-NCG.

#<N> **CpNcgRestart**
$< real = 1.e-1 >$

Restart NCG algorithm if cosine to angle between old and new gradient is larger than this
threshold (i.e. if gradients are far from orthogonal).

#<N> **CpAlsConditionCheck**

Check conditioning of CP-ALS fit.
(This is a debug option.)

#<N> **NestedCpScaleRankIncr**

Set the rank increment in FINDBESTCP driver for a tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ to $\Delta R = \Delta R_{in}(D-2)$, where ΔR_{in} is set using #<N> NestedCpRankIncr.

#<N> **CpAlsNormRegularization**
$< \text{ real } = \ 0.0 \text{ (no regularization) } >$

Regularization parameter for CP norm regularization.

#<N> **CpAlsScaleLSSystem**

Scale LS system in CP-ALS by norm of target tensor.

#<N> **CpAlsScaleFitTensor**

Scale fitting tensor to the norm of the target before running CP-ALS.

#<N> **CpAlsInitialErrorCalculation**

Calculate error before starting the CP-ALS iterations.
(May be a good idea if you want to be able to stop the iterations early.)

#<N> **EffectiveLowOrderSVD**

Use SVD to decompose tensors that are effectively of order 2 (other extents are 1).

#<N> **NoDecomposeCanonical**

Do not let this decomposer recompress CanonicalTensor.

#<N> **NoDecomposeDirProd**

Do not let this decomposer recompress TensorDirectProduct.

#<N> **NoDecomposeSum**

Do not let this decomposer recompress TensorSum.

Chapter 26

Laplace-decomposition options

26.1 General keywords

#<**N**> **Type**

    $< string >$ [NOOPT] [FIXLAP] [BESTLAP]

    Set the type of Laplace quadrature for approximating energy denominators. Options are 'NOOPT' (use pre-optimized data), 'FIXLAP' (optimize fixed number of points) , and 'BESTLAP' (optimize to given accuracy, allow the algorithm to increase the number of points).
    (Use 'NOOPT' for CP-VCC calculations where the denominators are only used for updates in the equation solver.)

#<**N**> **Name**

    $< string >$

    Assign a name to the quadrature.

#<**N**> **IoLevel**

    $< integer\ =\ 1\ >$

    IO level for Laplace quadrature.

#<**N**> **NumPoints**

    $< integer\ =\ 4\ >$

    Set the number of points for types 'NOOPT' and 'FIXLAP'.

#<**N**> **MaxPoints**

    $< integer\ =\ 20\ >$

    Set the maximum number of points for type 'BESTLAP'.

#<**N**> **MinPoints**

    $< integer\ =\ 2\ >$

Set the minimum number of points for type 'BESTLAP'.

#**<N>** **LapConv**
   $<$ *real* $=$ 1.0e-4 $>$

Set the requested accuracy for type 'BESTLAP'.

26.2 Advanced keywords

#**<N>** **HistogramDelta**
   $<$ *real* $=$ 1.0e-1 $>$

Set the spacing for constructing an energy histogram used for optimizing the quadrature.

Chapter 27

File conversion options

This section describes the file conversion features of MidasCpp. There are two main uses of this keyword:

(1) Conversion between molecular file formats, currently the conversion between MIDAS, MOLDEN and SINDO formats are currently supported. The latter two are mainly supported for the sake of using the programs for visualization, and they are not recommended for use in MIDAS inputs. Conversions from TURBOMOLE and ORCA to the MIDAS format can also be performed.

(2) The conversion of a set of potential energy files to `C++` code that can be turned into a program or dynamic linked library. Note that this option also requires the #2 MoleculeInput keyword.

The default behaviour is (1), so if one wish to utilize behaviour (2) either #2 Midasdynlib or #2 Midaspot should be present in the input.

27.1 General keywords

#1 Fileconversion

Read input for fileconversion until next #1 input key.

#2 Description
  $< string = $ N/A $ >$

Gives a description of the MidasPot which can be output by the generated program.

#2 Midasdynlib

If set, the potential `C++` file generated will be for creating a dynamically linked library.

#2 Midaspot

If set, the potential `C++` file generated will be for creating a stand alone program.

#2 **MoleculeInput**

$< string >$ [MIDAS] [MOLDEN] [TURBOMOLE] [ORCAHESS] [SINDO]
$< string >$

The first string indicates in which format the molecule file is given, while the second string indicates the path and name of the file in which molecular structure information is provided. Information given in the molecule file depends on the actual calculation at hand but usually involdes the reference geometry in Cartesian coordinates, vibrational frequencies and vibrational coordinates.
(For more information on the MIDAS molecule format see A.1.)

Example for MoleculeInput

```
#2 MoleculeInput
   MIDAS                // Read input in MidasCpp molecule format (.mmol).
   mymolecule.mmol      // Read file 'mymolecule.mmol' in main-directory.
```

#2 **Moleculeoutput**

$< string >$ [MIDAS] [MOLDEN] [SINDO]
$< string >$

The first string indicates in which format the output molecule file will be written in, while the second string indicates the path and name of the file to which the information will be written.
(Note that the standard setting for all purposes of surface generation should be MIDAS. The option to generate other file formats are mainly for interfacing with visualization programs and should be used with care for calculation purposes. For more information on the MIDAS molecule format see A.1.)

Example for Moleculeoutput

```
#2 MoleculeOutput
   MIDAS                // Write in MidasCpp molecule format (.mmol).
   mymolecule.mmol      // To the file 'mymolecule.mmol' in main-directory.
```

#2 **Operatorinput**

$< string = N/A >$
$< string = N/A >$
$< string = N/A > [< string >]$
$[< string = N/A >]$

Takes an operator name, describtion, filename of the operator file, optionally a reference value index and optionally the file name of an one_mode_grid_bounds.mbounds file. These are then parsed into C++ code according to the MidasPot procedure.

#2 Referencevalues
$< string\ =\ N/A\ >$

Give a filename from which the operator reference values should be read from. Reference values are by default assigned based on input order, i.e. first operator gets first value, second operator gets second value, etc. This behaviour can be overwritten by #2 Operatorinput.

Appendix A

Input file description

A.1 The molecule file

Standard input for molecular structure information is stored in a file named Molecule.mol.

Input for rectilinear vibrational coordinates is

#0 MOLECULEINPUT

  # 1 XYZ

    <number of atoms> <unit>

    comment line
    <element atom 1>   <x atom 1>   <y atom 1>   <z atom 1>
    <element atom 2>   <x atom 2>   <y atom 2>   <z atom 2>
    <element atom 3>   <x atom 3>   <y atom 3>   <z atom 3>
    ⋮

  #1 FREQ

    <number of modes> <unit>

    <frequency mode 1>

    <frequency mode 2>

    <frequency mode 3>

    ⋮

  #1 VIBCOORD

    <unit>

    #2 COORDINATE

      < x atom 1 mode 1 > < y atom 1 mode 1 > < z atom 1 mode 1 >
      < x atom 2 mode 1 > < y atom 2 mode 1 > < z atom 2 mode 1 >
      < x atom 3 mode 1 > < y atom 3 mode 1 > < z atom 3 mode 1 >
      ⋮

    #2 COORDINATE

      < x atom 1 mode 2 > < y atom 1 mode 2 > < z atom 1 mode 2 >
      < x atom 2 mode 2 > < y atom 2 mode 2 > < z atom 2 mode 2 >
      < x atom 3 mode 2 > < y atom 3 mode 2 > < z atom 3 mode 2 >
      ⋮

    ⋮

#0 MOLECULEINPUTEND

If the coordinates are multiplied entry-wise with $\sqrt{(m_i)}$, where m_i is the mass of the respective atom, they will be orthonormal.

Appendix B

Output file description

B.1 Vibrational wave function calculation output files

This section gives a non-exhaustive overview of the output files generated by the various vibrational wave-function methods implemented in `MidasCpp`.

Many output files contain a `<FileIndex>` which is used if the data is distributed over multiple files. Usually, this is not the case and the index is simply 0.

B.1.1 Output files from VSCF calculations

All VSCF calculations have a name called `<VscfName>` in the following. This consists of the prefix given under the `#3 Name` keyword followed by a state label (default is `_0.0`, i.e. the ground state). Note that the default VSCF prefix is `<Operator>_<Basis>`.

- `<VscfName>_Modals_<FileIndex>`   !VSCF

  Binary file containing the VSCF modals for the given VSCF calculation. The modals are saved vector by vector, i.e. the eigen-vector matrices are saved in column-major format. Note that for VSCF calculations used as reference for VCC, VscfName is replaced by VccName.

- `<VscfName>_OccModals_<FileIndex>`   !VSCF

  Binary file containing the occupied VSCF modal for each mode for the given VSCF calculation. Note that for VSCF calculations used as reference for VCC, VscfName is replaced by VccName.

- `<VscfName>_EigVal_<FileIndex>`   !VSCF

  Binary file containing the VSCF modal energies (relative to the occupied modal) for the given VSCF calculation. The energies are saved with the occupied-modal energy (always zero) first and then the rest in ascending order. Note that for VSCF calculations used as reference for VCC, VscfName is replaced by VccName.

- `<Operator>_<Basis>_OneModeInt_<FileIndex>`   !VSCF

Binary file containing the one-mode integrals of a given operator in a given primitive basis.

- `analysis/Vscf_ss_Info_Energy` `!VSCF`

  Summary of energies obtained from state-specific VSCF calculations. Used for later simulation of temperature effects.

- `analysis/Vscf_ss_Info_Prop` `!VSCF`

  Summary of the number of states and properties. Used for later simulation of temperature effects.

- `analysis/Vscf_ss_Info_Property_<PropNumber>` `!VSCF`

  Summary of a given property for all calculated states. Used for later simulation of temperature effects.

B.1.2 Output files from correlated vibrational wave-function calculations

All correlated vibrational wave-function calculations have a name called `<VccName>` in the following. This consists of the prefix given under the `#3 Name` keyword followed by a state label (as in VSCF), the excitation level (EXC), and the interaction-space order (ISO). Thus the name of a ground-state VCC[2] calculation with no limitations in the interaction space will have the name `<VccPrefix>_0.0_EXC_2_ISO_0`. Note that the VSCF calculation used as reference in a given correlated calculation uses the `<VccName>` as `<VscfName>` in its output files.

- `<VccName>_Vci_vec_<Root>_<FileIndex>` `!VCI`

  Binary file containing the VCI eigen vector for a given root. The data is stored in the same order as the mode-combination range with the reference coefficient first followed by the one-mode coefficients, two-mode coefficients, etc. For storage of matrices and higher-order tensors, the last (right) indices are leading.

- `<VccName>_Vcc_vec_0_<FileIndex>` `!VCC`

  Binary file containing the VCC solution vector. The data is stored in the same order as the mode-combination range.

- `<VccName>_Vcc_vec_tensor_0_0` `!VCC` `!TENSOR`

  Binary file containing the VCC solution vector in tensor format (can be either full or CP depending on the calculation).

- `<VccName>_Vcc_resid_tensor_0_0` `!VCC` `!TENSOR`

  Binary file containing the VCC residual vector in tensor format (can be either full or CP depending on the calculation).

- `<VccName>_Vcc_decomp_threshold` `!VCC` `!TENSOR`

  The latest decomposition threshold for the VCC amplitudes used in for CP-VCC ground-state calculations.

- `<VccName>_Vcc_<Algorithm>_<VecType>_<VecIndex>` `!VCC` `!TENSOR`

  Residuals and trials for CROP or DIIS subspaces used for restarts. $<Algorithm>$ is either CROP or DIIS. $<VecType>$ is either resid or trial.

- `<VccName>_Vmp_sum_wf_<FileIndex>` `!VMP`

  Binary file containing the VMP wave function.

- `<VccName>_rsp_eigvec_<Root>_<FileIndex>` `!VCC` `!VCI` `!RSP`

  Binary file containing the response eigen vector for a given root. The data is stored in the same order as the mode-combination range.

- `<VccName>_rsp_eigvec_tensor_<Root>` `!VCC` `!RSP` `!TENSOR`

  VCC response eigenvector for tensor-based calculations.

- `<VccName>_rsp_eigval_<FileIndex>` `!VCC` `!VCI` `!RSP`

  Binary file containing the response eigenvalues (excitation energies).

- `<VccName>_rsp_eigval_<ReIm>_tensor_0` `!VCC` `!RSP` `!TENSOR`

  Binary file containing the real or imaginary parts of tensor-based VCC response eigenvalues (excitation energies).

- `<VccName>_rsp_eigvec_left_<Root>_<FileIndex>` `!VCC` `!RSP`

  Binary file containing the left response eigen vector for a given root. The data is stored in the same order as the mode-combination range.

- `<VccName>_rsp_eigvec_left_tensor_<Root>` !VCC !RSP !TENSOR

  Left VCC response eigenvector for tensor-based calculations.

- `<VccName>_rsp_eigval_left_<FileIndex>` !VCC !RSP

  Binary file containing the left response eigenvalues (excitation energies).

- `<VccName>_rsp_eigval_<ReIm>_left_tensor_0` !VCC !RSP !TENSOR

  Binary file containing the real or imaginary parts of tensor-based left VCC response eigenvalues (excitation energies).

- `<VccName>_rsp_vci_eta_<Operator>_<FileIndex>` !VCI !RSP

  Auxiliary vector used in VCI response calculations.

- `<VccName>_eta_vec_<Operator>_<FileIndex>` !VCC !VCI !RSP

  Auxiliary vector used in VCC and VCI response calculations.

- `<VccName>_eta0_vec_<FileIndex>` !VCC !RSP

  Auxiliary vector used in VCC response calculations.

- `<VccName>_mul0_vec_<Root>_<FileIndex>` !VCC !RSP

  Zeroth-order multipliers for a given root. Used in VCC response calculations.

- `<VccName>_rsp_vcc_fr_<Root>_<FileIndex>` !VCC !RSP

  Auxiliary vector used in VCC response calculations.

- `<VccName>_rsp_vcc_m_<Root>_<FileIndex>` !VCC !RSP

  Auxiliary vector used in VCC response calculations.

- `<VccName>_xi_vec_<Operator>_<FileIndex>` !VCC !RSP

  Auxiliary vector used in VCC response calculations.

- `RspTargetVector_<Root>_<FileIndex>` !VCC !VCI !RSP

  Target vectors generated by the default eigenvalue solver.

- `target_<Root>` !VCC !RSP !TENSOR

  Target vectors used by the new eigenvalue solvers, e.g. for tensor-based VCC response calculations.

- `analysis/RspTargetVectors<Extension>` !VCC !VCI !RSP

  Auxiliary files for targeting procedure.

- `analysis/Target_Overlaps` !VCC !VCI !RSP

  Auxiliary file for targeting procedure.

- `analysis/TargSolvecS.mat` !VCC !VCI !RSP

  Auxiliary file for targeting procedure.

- `analysis/SolvecE.mat` !VCC !VCI !RSP

  Auxiliary file for targeting procedure.

- `analysis/VCCRESTART.INFO` !VCC !VCI

  Info file for restarting correlated wave-function calculations.

- `analysis/VCCRSPRESTART.INFO` !VCC !VCI !RSP

  Info file for restarting response calculations.

- `analysis/TENSORVCCRESTART.INFO` !VCC !TENSOR

  Info file for restarting VCC calculations using TensorNlSolver (this includes all CP-VCC calculations).

- `analysis/<Method>_ss_Info_Energy` !VCC !VCI !VMP

  Summary of energies obtained from state-specific correlated wave-function calculations. Used for later simulation of temperature effects.

- `analysis/<Method>_ss_Info_Prop` !VCC !VCI !VMP

  Summary of the number of states and properties. Used for later simulation of temperature effects.

- `analysis/<Method>_ss_Info_Property_<PropNumber>` !VCC !VCI !VMP

  Summary of a given property for all calculated states. Used for later simulation of temperature effects.

B.1.3 Output files from time-dependent wave-function calculations

- `analysis/<TdHName>_energy.dat` !TDH

  Energy of the TDH wave function at all steps during the integration.

- `analysis/<TdHName>_phase.dat` !TDH

  Phase factor ($F(t)$ in $|\bar{\Phi}\rangle = e^{-iF(t)}|\tilde{\Phi}\rangle$) of the TDH wave function at all steps during the integration.

- `analysis/<TdHName>_ac.dat` !TDH

  Auto-correlation function calculated at equidistant time points.

- `analysis/<TdHName>_t12ac.dat` !TDH

  Auto-correlation function calculated at equidistant time points using the wave function at half time.

- `analysis/<TdHName>_<CorrelationFunction>_spectrum.dat` !TDH

  The spectrum of the TDH wave packet obtained as a Fourier transform of a given correlation function.

- `analysis/<TdHName>_mean_<Operator>.dat` !TDH

  Expectation value of operator calculated at all steps during the integration. The file also contains expectation values where only active terms are included for each mode.

- `analysis/<TdHName>_kappa_norms.dat` !TDH

The norms of the kappa vectors in X-TDH calculated at all steps during the integration.

- `analysis/<TdHName>_wf.dat` !TDH

  The TDH wave function (the time-dependent modals) calculated at equidistant time points. Note that the phase factor is multiplied on all modals so the total wave function is not a product of the modals!

- `analysis/<TdHName>_density.dat` !TDH

  The TDH one-mode wave-function density calculated at equidistant time points.

- `analysis/<TdHName>_pair_density_<Mode1>_<Mode2>.dat` !TDH

  The TDH two-mode wave-function density for a given pair of modes calculated at equidistant time points.

B.2 Surface calculation output files

This section gives a non-exhausting list of output files generated over the course of a potential energy or molecular property surface calculation.

B.2.1 Output files in analysis sub-directory

- `PointQ_Prop<PropNr>_<ModeNr>.mplot` !ADGA

  File holding the fitted displacement vibrational coordinate value and associated bar-potential value for use with plotting tools. Note that this file will always apply to the current iteration, while files applicable for previous iterations will have the suffix `_it<IterNr>`.

- `Potential_Prop<PropNr>_Q<ModeNr>.mplot` !ADGA

  File holding evaluated values of the analytic form of the surface at a uniform grid of plot points for use with plotting tools. Note that this file will always apply to the current iteration, while files applicable for previous iterations will have the suffix `_it<IterNr>`.

- `MaxDens_Prop<PropNr>_Q<ModeNr>.mplot` !ADGA

  File holding the evaluated values for the maximum vibrational density at a uniform grid of plot points for use with plotting tools. Note that this file will always apply to the current iteration, while files applicable for previous iterations will have the suffix `_it<IterNr>`.

- `MeanDens_Prop<PropNr>_Q<ModeNr>.mplot` `!ADGA`

  File holding the evaluated values for the mean vibrational density at a uniform grid of plot points for use with plotting tools. Note that this file will always apply to the current iteration, while files applicable for previous iterations will have the suffix `_it<IterNr>`.

- `one_mode_grid_bounds.mbounds` `!ADGA` `!Static`

  File holding information on the boundaries for the individual modes, i.e. the dimensions of the grid of single points.

- `EsPoints_Prop<PropNr>_Q<ModeNr>.mplot` `!ADGA` `!Static`

  File holding the non-fitted property values in terms of displacement vibrational coordinates for use with plotting tools. Note that this file will always apply to the current iteration, while files applicable for previous iterations will have the suffix `_it<IterNr>`.

- `VscfDensFiles` `!ADGA`

  File holding information on where the `MaxDens_Q<ModeNr>.mplot` or `MeanDens_Q<ModeNr>.mplot` files are stored.

B.2.2 Output files in savedir sub-directory

- `prop_no_<PropNr>` `!ADGA` `!Static` `!Taylor`

  File holding the single point calculation number, single point identification code and the associated calculated property value at each line. The property values are the values obtained directly from an electronic structure calculation.

- `prop_no_<PropNr>.bar` `!ADGA` `!Static`

  File holding the displacement vibrational coordinate value and the associated property value resulting from evaluating the analytic form of the surface, as seen in the `prop_no_<ModeNr>.mop` file. This means that it is the so-called bar-potential or bar-property values that can be found in this file.

- `prop_no_<PropNr>_shifted` `!ADGA` `!Static` `!Taylor`

  File holding the single point calculation number, single point identification code and the associated calculated property value subtracted by the reference value as given in the `Reference_values` file, at each line. The property values are the values obtained directly from an electronic structure calculation.

- `prop_no_<PropNr>.mop` !ADGA !Static !Taylor

  A Midas Operator file (.mop) holding the analytical surface in terms of the fit-basis functions, which results from fitting and/or interpolating the grid of single points.

- `Harmonic_Frequencies` !ADGA !Static !Taylor

  File holding the harmonic frequencies for each vibrational mode if it is possible to obtain these over the coarse of the calculation.

- `Inertia_Tensor` !ADGA !Static !Taylor

  File holding the inertia tensor for the reference structure.

- `offsets_to_bar_files` !ADGA !Static

  File holding information on which blocks of the `prop_no_<Propnr>.bar` file belongs to which mode or mode combination.

- `Reference_values` !ADGA !Static !Taylor

  File holding the property/properties value/values for the reference structure.

- `sp_restart_<PropNr>` !ADGA !Static !Taylor

  File holding restart information on each calculated single point. Single point identification code is followed by the number of properties calculated for it. The following lines contain the rotation group, a label and the property value.

- `These_Properties_Have_Been_Calculated` !ADGA !Static !Taylor

  File holding information on the property number, the electronic state it belongs to and a label for the particular multi-level it was calculated for. Note that the properties will be numbered based on which line they appear in the `midasifc.propinfo` file of the associated **setup** sub-directory.

Bibliography

[1] O. Christiansen, D. Artiukhin, I. H. Godtliebsen, E. M. Gras, W. Győrffy, M. B. Hansen, M. B. Hansen, E. L. Klinting, J. Kongsted, C. König, D. Madsen, N. K. Madsen, K. Monrad, G. Schmitz, P. Seidler, K. Sneskov, M. Sparta, B. Thomsen, D. Toffoli, A. Zoccante, *MidasCpp*, version 2019.04.0.

[2] B. Thomsen, K. Yagi, O. Christiansen, "Optimized coordinates in vibrational coupled cluster calculations", *Journal of Chemical Physics* **2014**, *140*, 154102–124101–15.

[3] B. Thomsen, K. Yagi, O. Christiansen, "A simple state-average procedure determining optimal coordinates for anharmonic vibrational calculations", *Chemical Physics Letters* **2014**, *610–611*, 288–297.

[4] E. L. Klinting, C. König, O. Christiansen, "Hybrid Optimized and Localized Vibrational Coordinates", *The Journal of Physical Chemistry A* **Nov. 2015**, *119*, 11007–11021.

[5] C. König, M. B. Hansen, I. H. Godtliebsen, O. Christiansen, "FALCON: A method for flexible adaptation of local coordinates of nuclei", *Journal of Chemical Physics* **2016**, *144*, 074108.

[6] J. Kongsted, O. Christiansen, "Automatic generation of force fields and property surfaces for use in variational vibrational calculations of anharmonic vibrational energies and zero-point vibrational averaged properties", *J. Chem. Phys.* **2006**, *125*, 124108–16.

[7] D. Toffoli, J. Kongsted, O. Christiansen, "Automatic generation of potential energy and property surfaces of polyatomic molecules in normal coordinates", *J. Chem. Phys.* **2007**, *127*, 204106–14.

[8] E. Matito, D. Toffoli, O. Christiansen, "A hierarchy of potential energy surfaces constructed from energies and energy derivatives calculated on grids", *J. Chem. Phys.* **2009**, *130*, 1341041–13.

[9] C. König, O. Christiansen, "Linear-scaling generation of potential energy surfaces using a double incremental expansion", *J. Chem. Phys.* **2016**, *145*, 064105.

[10] M. Sparta, D. Toffoli, O. Christiansen, "An Adaptive Density-Guided Approach for the generation of potential energy surfaces of polyatomic molecules", *Theor. Chem. Acc.* **2009**, *123*, 413 –429.

[11] M. Sparta, D. T. I.-M. Høyvik, O. Christiansen, "Potential energy surfaces for vibrational structure calculations from a multiresolution adaptive density-guided approach: implementation and test calculations", *J. Phys. Chem. A* **2009**, *113*, 8712–8723.

[12] E. L. Klinting, B. Thomsen, I. H. Godtliebsen, O. Christiansen, "Employing general fit-bases for construction of potential energy surfaces with an adaptive density-guided approach", *The Journal of Chemical Physics* **Feb. 2018**, *148*, 064113.

[13] O. Christiansen, "Møller-Plesset perturbation theory for vibrational wave functions", *J. Chem. Phys.* **2003**, *119*, 5773–5781.

[14] J. M. Bowman, "Self-consistent field energies and wavefunctions for coupled oscillators", *J. Chem. Phys.* **1978**, *68*, 608–610.

[15] R. B. Gerber, M. A. Ratner, "self-consistent field review", *Adv. Chem. Phys.* **1988**, *70*, 97.

[16] M. B. Hansen, M. Sparta, P. Seidler, O. Christiansen, D. Toffoli, "A new formulation and implementation of vibrational self-consistent field (VSCF) theory", *J. Chem. Theo. and Comp.* **2010**, *6*, 235–248.

[17] O. Christiansen, "A second quantization formulation of multi-mode dynamics", *J. Chem. Phys.* **2004**, *120*, 2140–2148.

[18] J. O. Jung, R. B. Gerber, "Vibrational wave functions and spectroscopy of $(H_2O)_n$, n=2,3,4,5: Vibrational self-consistent field with correlation corrections", *J. Chem. Phys.* **1996**, *105*, 10332–10348.

[19] L. S. Norris, M. A. Ratner, A. E. Roitberg, R. B. Gerber, "Møller-Plesset perturbation theory applied to vibrational problems", *J. Chem. Phys.* **1996**, *105*, 11261–11267.

[20] E Matito, J. M. Barroso, E. Besalú, O Christiansen, J. M. Luis, "The vibrational auto-adjusting perturbation theory", *Theor. Chem. Acc.* **2009**, *123*, 41–49.

[21] O. Christiansen, "Vibrational coupled cluster theory", *J. Chem. Phys.* **2004**, *120*, 2149–2159.

[22] P. Seidler, M. B. Hansen, O. Christiansen, "Towards fast computations of correlated vibrational wave functions: Vibrational coupled cluster response excitation energies at the two-mode coupling level", *J. Chem. Phys.* **2008**, *128*, 154113–12.

[23] A. Zoccante, P. Seidler, O. Christiansen, "Computation of expectation values from vibrational coupled-cluster at the two-mode coupling level", *J. Chem. Phys.* **2011**, *134*, 154101–154109.

[24] P. Seidler, O. Christiansen, "Automatic derivation and evaluation of vibrational coupled cluster theory equations", *J. Chem. Phys.* **2009**, *131*, 234109–15.

[25] P. Seidler, E. Matito, O. Christiansen, "Vibrational coupled cluster theory with full two-mode and approximate three-mode couplings: The VCC[2pt3] model", *J. Chem. Phys.* **2009**, *131*, 034115–12.

[26] A. Zoccante, P. Seidler, M. B. Hansen, O. Christiansen, "Approximate inclusion of four-mode couplings in vibrational coupled-cluster theory", English, *Journal of Chemical Physics* **2012**, *136*, 204118–204118–12.

[27] N. K. Madsen, I. H. Godtliebsen, O. Christiansen, "Efficient algorithms for solving the non-linear vibrational coupled-cluster equations using full and decomposed tensors", *The Journal of Chemical Physics* **2017**, *146*, 134110.

[28] O. Christiansen, "Response theory for vibrational wave functions", *J. Chem. Phys.* **2005**, *122*, 194105.

[29] P. Seidler, O. Christiansen, "Vibrational excitation energies from vibrational coupled cluster response theory", *J. Chem. Phys.* **2007**, *126*, 204101.

[30] P. Seidler, M. Sparta, O. Christiansen, "Vibrational coupled cluster response theory: A general implementation", *J. Chem. Phys.* **2011**, *134*, 054119–15.

[31] W. Györffy, P. Seidler, O. Christiansen, "Solving the eigenvalue equations of correlated vibrational structure methods: preconditioning and targeting strategies", *J. Chem. Phys.* **2009**, *131*, 024108–16.

[32] B. Thomsen, M. B. Hansen, P. Seidler, O. Christiansen, "Vibrational absorption spectra from vibrational coupled cluster damped linear response functions calculated using an asymmetric Lanczos algorithm", English, *Journal of Chemical Physics* **2012**, *136*, 124101–124101–17.

[33] I. H. Godtliebsen, O. Christiansen, "A band Lanczos approach for calculation of vibrational coupled cluster response functions: simultaneous calculation of IR and Raman anharmonic spectra for the complex of pyridine and a silver cation", en, *Physical Chemistry Chemical Physics* **2013**, DOI 10.1039/C3CP50283J.

[34] I. H. Godtliebsen, O. Christiansen, "Calculating vibrational spectra without determining excited eigenstates: Solving the complex linear equations of damped response theory for vibrational configuration interaction and vibrational coupled cluster states", en, *The Journal of Chemical Physics* **Oct. 2015**, *143*, 134108.

[35] O. Christiansen, J. Kongsted, M. J. Paterson, J. M. Luis, "Linear Response functions for a vibrational configuration interaction state", *J. Chem. Phys.* **2006**, *125*, 214309.

[36] M. B. Hansen, O. Christiansen, C. Hättig, "Automated calculation of anharmonic vibrational contributions to first hyperpolarizabilities: Quadratic response functions from vibrational configuration interaction wave functions", *J. Chem. Phys.* **2009**, *131*, 154101.

[37] M. B. Hansen, O. Christiansen, "Vibrational contributions to cubic response functions from vibrational configuration interaction response theory", *Journal of Chemical Physics* **2011**, *135*, 154107–154107–14.

[38] P. Seidler, M. B. Hansen, W. Györffy, D. Toffoli, O. Christiansen, "Vibrational absorption spectra calculated from vibrational configuration interaction response theory using the Lanczos method", *J. Chem. Phys.* **2010**, *132*, 164105–15.

[39] I. H. Godtliebsen, B. Thomsen, O. Christiansen, "Tensor Decomposition and Vibrational Coupled Cluster Theory", *The Journal of Physical Chemistry A* **2013**, *117*, 7267–7279.

[40] I. H. Godtliebsen, M. B. Hansen, O. Christiansen, "Tensor decomposition techniques in the solution of vibrational coupled cluster response theory eigenvalue equations", *The Journal of Chemical Physics* **2015**, *142*, 024105.

[41] N. K. Madsen, I. H. Godtliebsen, S. A. Losilla, O. Christiansen, "Tensor-decomposed vibrational coupled-cluster theory: Enabling large-scale, highly accurate vibrational-structure calculations", *The Journal of Chemical Physics* **Jan. 2018**, *148*, 024103.

[42] C. d. Boor, *A Practical Guide to Splines*, Springer Verlag, New York, **1978**.

[43] D. Toffoli, M. Sparta, O. Christiansen, "Accurate multimode vibrational calculations using a B-spline basis: theory, tests and application to dioxirane and diazirinone", *Mol. Phys.* **2011**, *109*, 673–685.

[44] T. L. Fletcher, P. L. A. Popelier, "Multipolar Electrostatic Energy Prediction for all 20 Natural Amino Acids Using Kriging Machine Learning", *Journal of Chemical Theory and Computation* **2016**, *12*, 2742–2751.

[45] S. M. Kandathil, T. L. Fletcher, Y. Yuan, J. Knowles, P. L. A. Popelier, "Accuracy and tractability of a kriging model of intramolecular polarizable multipolar electrostatics and its application to histidine", *Journal of Computational Chemistry* **2013**, *34*, 1850–1861.

[46] M. J. L. Mills, P. L. A. Popelier, "Polarisable multipolar electrostatics from the machine learning method Kriging: an application to alanine", *Theoretical Chemistry Accounts* **2012**, *131*, 1137.

[47] C. E. Rasmussen, C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*, The MIT Press, **2005**.

[48] M. Riedmiller, H. Braun, RPROP - A Fast Adaptive Learning Algorithm, tech. rep., Proc. of ISCIS VII), Universität Karlsruhe, **1992**.

[49] C. Igel, M. Hüsken in Proceedings of the Second International Symposium on Neural Computation, **2000**, pp. 115–121.

[50] G. Schmitz, O. Christiansen, "Gaussian process regression to accelerate geometry optimizations relying on numerical differentiation", *The Journal of Chemical Physics* **2018**, *148*, 241704.

[51] G. Schmitz, D. G. Artiukhin, O. Christiansen, "Approximate high mode coupling potentials using Gaussian process regression and adaptive density guided sampling", *The Journal of Chemical Physics* **2019**, *150*, 131102.