# MidasCpp

**M**olecular **I**nteractions **D**ynamics **A**nd **S**imulation
**C++**
**C**hemistry **P**rogram **P**ackage
**C**ompletely **P**athetic **P**rogramming

2026.10.0

# User's Guide

# Contents

**Preface** i
0.1 Distribution and License . . . i
0.2 Standard Disclaimer . . . i
0.3 Citation . . . i
0.4 Authors . . . i

**I MidasCpp User's Guide** 1

**1 Getting started** 2
1.1 Installation . . . 2
1.2 Running `MidasCpp` . . . 3
1.3 Input . . . 3
1.4 Getting started with the example suite . . . 4

**2 Tutorial** 6
2.1 Running a VSCF calculation . . . 6
2.2 Constructing a potential energy or molecular property surface . . . 7

**3 More detailed information** 12
3.1 Advanced installations . . . 12
3.1.1 Compiling with `MPI` and `OpenMP` . . . 12
3.1.2 External libraries . . . 13
3.1.3 Vector inputs . . . 13
3.2 Potential energy or molecular property surface calculations . . . 14
3.2.1 Electronic structure programs . . . 14
3.2.2 Folder structure . . . 14
3.2.3 Commencing a calculation . . . 16
3.3 Running calculations with general curvilinear coordinates . . . 17
3.3.1 Polyspherical coordinates . . . 17
3.3.2 Taylor expansion-based KEO variants . . . 23
3.3.3 $n$-mode expansion-based KEO variants . . . 23
3.3.4 Sequential ADGA workflow . . . 24

**II Theory** 28

**4 Coordinates** 30
4.1 Normal-Coordinates and harmonic oscillator treatment of Molecular vibrations . . . 30

4.1.1 Classical coordinate transformation approach . . . 30
4.1.2 Quantization of the vibrational energy . . . 33
4.2 Frequency scaled normal coordinates . . . 33
4.3 Semi-numerical gradient-based frequency analysis . . . 34
4.4 Optimized coordinates . . . 37
4.4.1 Optimized Coordinates . . . 37
4.4.2 On PES forms appropriate for Optimization of Coordinates . . . 38
4.4.3 State-average Optimized Coordinates . . . 38
4.4.4 Hybrid Optimized and Localized Coordinates . . . 38
4.5 FALCON: Flexible Adaptation of Local Coordinates Of Nuclei . . . 39

**5 Kinetic energy operators** **40**
5.1 The Watson kinetic energy operator in normal coordinates . . . 40
5.1.1 Non-linear molecules . . . 40
5.1.2 Linear molecules . . . 41

**6 Potential energy surfaces** **42**
6.1 Potential energy surfaces - general aspects . . . 42
6.2 The potential energy surface as a function of many variable . . . 43
6.3 Approximate representation of potential energy surfaces . . . 44
6.3.1 Taylor expansion of the potential . . . 45
6.3.2 Restricted mode coupling . . . 46
6.3.3 Building the potential through its values on a grid . . . 49
6.3.4 Midas PES construction methods . . . 50
6.3.5 Taylor expansion . . . 50
6.3.6 Static grids expansion . . . 50
6.3.7 Double incremental expansion . . . 51
6.3.8 The Adaptive density guided approach . . . 52
6.3.9 The Time-Dependent Adaptive density guided approach . . . 54

**7 Wave-functions for nuclear motion for systems with many degrees of freedom: An overview** **55**
7.1 Introduction, notation, and the scaling problem . . . 55
7.2 The vibrational self consistent field method . . . 56
7.3 Basis set expansion for VSCF . . . 58
7.4 Correlated Vibrational Wave functions : basic aspects . . . 59
7.4.1 A few comments . . . 59
7.4.2 Vibrational Configuration Interaction . . . 59
7.4.3 Vibrational Møller-Plesset perturbation theory . . . 61
7.4.4 Vibrational coupled cluster theory . . . 62

**8 Many-mode wave-functions in MidasCpp** **65**
8.1 Vibrational Self-Consistent Field . . . 65
8.2 Vibrational Møller-Plesset perturbation theory . . . 65
8.3 Vibrational Coupled Cluster . . . 66
8.3.1 Ground-state Vibrational Coupled Cluster . . . 66
8.3.2 VCC response theory . . . 66
8.4 Vibrational Configuation Interaction . . . 67

8.5 Tensor-Decomposition Methods . . . 68

**9 Time-dependent wave-function propagation in MidasCpp 69**
9.1 Time-dependent Hartree (TDH) . . . 69
9.2 Multiconfiguration Time-dependent Hartree (MCTDH) . . . 69
9.3 Time-dependent Vibrational Coupled-Cluster (TDVCC) . . . 71

**10 Primitive basis sets and integrals 72**
10.1 B-spline basis functions . . . 72
10.1.1 B-spline basis function definition . . . 72
10.1.2 Integrals of B-spline basis functions . . . 73
10.1.3 Why B-spline basis sets are useful . . . 73
10.1.4 Defining the B-spline basis in MidasCpp . . . 74
10.1.5 B-spline basis sets literature and citations . . . 74
10.2 Harmonic oscillator basis functions . . . 74
10.3 Distributed Gaussians . . . 74

**11 Machine Learning Algorithms 75**
11.1 Gaussian Process Regression . . . 75
11.1.1 Representing the potential . . . 76
11.1.2 Kernel functions . . . 76
11.1.3 Optimizing the hyper parameters . . . 77
11.2 Using derivative information in GPR . . . 79
11.2.1 Squared exponential kernel . . . 80
11.3 Descriptors . . . 81

**III MidasCpp Input/Output Reference Manual 83**

**12 Midas Input Basics 84**
12.1 Input structure and keyword syntax . . . 84
12.2 Rules for operator inputs (Midas Operator) . . . 84
12.2.1 Basic syntax for operator input . . . 84
12.2.2 Mode labels and order . . . 85
12.2.3 Non-multi-state one-mode operators . . . 86
12.2.4 Multi-state operators . . . 87
12.3 Rules for function inputs (Midas Operator) . . . 88
12.4 General keywords . . . 89

**13 General input options 90**
13.1 General keywords . . . 90
13.2 Advanced keywords . . . 91

**14 Singlepoint input options 93**
14.1 General keywords . . . 93
14.2 Advanced keywords . . . 99

**15 Vib input options 100**
15.1 General keywords . . . 100
15.2 Advanced keywords . . . 101

**16 Basis input options** **102**
16.1 General keywords . . . 102
16.2 Advanced keywords . . . 104

**17 Operator input options** **106**
17.1 General keywords . . . 106
17.2 Advanced keywords . . . 111

**18 VSCF input options** **113**
18.1 General keywords . . . 113
18.2 Advanced keywords . . . 115

**19 VCC input options** **117**
19.1 General keywords . . . 117
19.2 Advanced keywords . . . 124
19.3 Tables . . . 136

**20 Response input options** **137**
20.1 General keywords . . . 137
20.2 Advanced keywords . . . 142

**21 TDH input options** **143**
21.1 General keywords . . . 143
21.2 Advanced keywords . . . 146

**22 MCTDH input options** **149**
22.1 General keywords . . . 149
22.2 Advanced keywords . . . 152

**23 TDVCC input options** **156**
23.1 General keywords . . . 156
23.2 Advanced keywords . . . 161

**24 Pes input options** **168**
24.1 General keywords . . . 168
24.2 Advanced keywords . . . 184

**25 FitBasis input options** **190**
25.1 General keywords . . . 190
25.2 Advanced keywords . . . 192

**26 FitBasisDef and .mfitbas input options** **194**
26.1 General keywords . . . 194
26.2 Advanced keywords . . . 195

**27 System input options** **197**
27.1 General keywords . . . 197
27.2 Advanced keywords . . . 199

**28 Falcon input options** **200**
28.1 General keywords . . . 200
28.2 Advanced keywords . . . 202

**29 FreqAna input options** **203**
29.1 General keywords . . . 203

**30 Normal Coordinates input options** **206**
30.1 General keywords . . . 206

**31 Local Bond Stretch Coordinates input options** **208**
31.1 General keywords . . . 208

**32 RotCoord input options** **209**
32.1 General keywords . . . 209

**33 Machine Learning options** **213**
33.1 General keywords . . . 213

**34 Additional Machine Learning options** **220**
34.1 General keywords . . . 220

**35 ODE-integration options** **221**
35.1 General keywords . . . 221
35.2 Advanced keywords . . . 222

**36 Tensor-decompostion options (stand-alone module)** **224**
36.1 General keywords . . . 224

**37 Tensor-decomposition options** **230**
37.1 General keywords . . . 230
37.2 Advanced keywords . . . 235

**38 Laplace-decomposition options** **241**
38.1 General keywords . . . 241
38.2 Advanced keywords . . . 242

**39 File conversion options** **243**
39.1 General keywords . . . 243

**40 Operator Reader Input Options** **246**
40.1 General keywords . . . 246
40.2 Advanced keywords . . . 248
40.3 Alternative operator file inputs . . . 249
40.3.1 Alternative Midas OPerator format (Historical) . . . 249
40.3.2 Spectroscopic format . . . 250

**41 Midas Molecule Input Options** **251**
41.1 General keywords . . . 252

**42 Midas Metamolecule Input Options** **255**
42.1 General keywords . . . . . 255

**A Additional input description** **257**
A.1 Example for a multi-state operator . . . . . 257
A.2 Automatic differentiation of general functions . . . . . 258
A.3 Vector Input Unfolding . . . . . 258

**B Output file description** **259**
B.1 Vibrational wave function calculation output files . . . . . 259
B.1.1 Output files from VSCF calculations . . . . . 259
B.1.2 Output files from correlated vibrational wave-function calculations . . . . . 260
B.1.3 Output files from time-dependent wave-function calculations . . . . . 264
B.2 Surface calculation output files . . . . . 265
B.2.1 Output files in **analysis** sub-directory . . . . . 265
B.2.2 Output files in **savedir** sub-directory . . . . . 266

# Preface

This is the documentation for MidasCpp version 2026.10.0

## 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 https://source.coderefinery.org/midascpp/midascpp, which is *the* source of distribution including updates and communication.

## 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,

**Consider to cite:**

1. O. Christiansen, D. G. Artiukhin, F. Bader, I. H. Godtliebsen, E. M. Gras, W. Győrffy, M. B. Hansen, M. B. Hansen, M. G. Højlund, N. M. Høyer, R. B. Jensen, A. B. Jensen, E. L. Klinting, J. Kongsted, C. König, S. A. Losilla, D. Madsen, N. K. Madsen, M. Majland, K. Monrad, A. S. Lykke-Møller, L. H. Olsen, K. Rüther, G. Schmitz, P. Seidler, K. Sneskov, M. Sparta, B. Thomsen, S. R. Thomsen, D. Toffoli, A. Zoccante, *MidasCpp*, version 2026.10.0

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 G. Artiukhin
- Frederik Bader
- **Ove Christiansen**
- Ian Heide Godtliebsen
- Eduard Matito Gras
- Werner Győrffy
- Mikkel Bo Hansen
- Mads Bøttger Hansen
- Mads Greisen Højlund
- Nicolai Machholdt Høyer
- Rasmus Berg Jensen
- Andreas Buchgraitz Jensen
- Emil Lund Klinting
- Jacob Kongsted
- Carolin König
- Sergio Alberto Losilla
- Diana Madsen
- Niels Kristian Madsen
- Marco Majland
- Kasper Monrad
- August Smart Lykke-Møller
- Lars Henrik Olsen
- Katharina Rüther
- Gunnar Schmitz
- Peter Seidler
- Kristian Sneskov
- Manuel Sparta
- Bo Thomsen
- Sebastian Riis Thomsen
- Daniele Toffoli
- Alberto Zoccante

# References

## Midas Related Publications

[1] O. Christiansen, D. G. Artiukhin, F. Bader, I. H. Godtliebsen, E. M. Gras, W. Győrffy, M. B. Hansen, M. B. Hansen, M. G. Højlund, N. M. Høyer, R. B. Jensen, A. B. Jensen, E. L. Klinting, J. Kongsted, C. König, S. A. Losilla, D. Madsen, N. K. Madsen, M. Majland, K. Monrad, A. S. Lykke-Møller, L. H. Olsen, K. Rüther, G. Schmitz, P. Seidler, K. Sneskov, M. Sparta, B. Thomsen, S. R. Thomsen, D. Toffoli, A. Zoccante, *MidasCpp*, version 2026.10.0.

# Part I

# MidasCpp User's Guide

# Chapter 1

# Getting started

In the following sections, you will find the basic instructions how to install and run MidasCpp. Installation is explained in 1.1 and how to run in 1.2 followed by a short description on how to get started using our example suite in 1.4. A small tutorial for basic calculations is given in chapter 2.

## 1.1 Installation

To get hold of the source code either navigate to https://source.coderefinery.org/midascpp/midascpp/-/release
and download a release tarball, or use a commandline tool like wget to download a specific release:

```
wget https://source.coderefinery.org/midascpp/midascpp/-/archive/2023.10.0/midascpp-2023.10.0.tar.
```

It is also possible to clone the git repository containing the MidasCpp source code:

```
git clone git@source.coderefinery.org:midascpp/midascpp.git
```

For compilation you need an up-to-date C++ compiler which complies to the C++20 standard - we recomment using gcc version 11.0.0 or above. Further requirements on libraries or tested compilers are listed in Table 1.1.

To configure the compilation run

```
./configure --prefix=/path/to/install
```

where /path/to/install is the path you want MidasCpp binaries and other files to be installed to after compilation. Running the configure script this way sets up Makefiles etc. for a standard build without parallelization or other features. See 3.1 for further hints on configuring or tweaking installations. The source code can now be compiled by typing

```
make complete
```

Using the option complete the source code is compiled and linked, and afterwards MidasCpp binaries etc. are installed in the prefix specified when configured. To be able to access MidasCpp from the command line append the bin directory of the MidasCpp installation to your PATH. This can by automated by adding the following two lines to your .bashrc:

```
export MIDASINSTALL=/path/to/install
export PATH=$MIDASINSTALL/bin:$PATH
```

To verify the correctness of your MidasCpp installation it is strongly recommended to run some examples from the test_suite. To do this go to test_suite and run ./TEST installshort. This will test the most important parts of MidasCpp in a few seconds.

For a more elaborate test run ./TEST installlong, but do note this may take hours. For a more complete overview of additional test we refer to the info in the test_suite itself.

Table 1.1: Overview of required compilers, libraries and tools to compile, test and run `MidasCpp`. In some cases different possible tested options are given.

| **Compilers** | | |
|---|---|---|
| C++ | | |
| **Name** | **Version** | **Comments** |
| g++ | 11.4.0 | Our default. |
| clang++ | 18.1.5 | Also tested. |
| Fortran | | |
| **Name** | **Version** | **Comments** |
| gfortran | 11.4.0 | Needed for Netlib LAPACK. F90 standard required. |
| **Linux/Unix Tools** | | |
| **Name** | **Version** | **Comments** |
| CMake | 3.27.7 | |
| GNU Make | 4.3 | |
| | | The tools below are mainly used for the test and example suite. |
| bash | 5.1.16 | |
| awk | | |
| bc | | |

## 1.2 Running `MidasCpp`

To run `MidasCpp` in the most basic way without any parallelization or other features run:

```
midascpp <midas input file>
```

So for example

```
midascpp midas.minp
```

will run `MidasCpp` using the input file `midas.minp` in the current directory. This will generate an output file called `midas.mout`. Output is always directed to a file with the same name as the input file with extension `.mout`. Additional command line options can be found via

```
midascpp --help
```

Some calculations also produces output in terms of extra data files, scripts and figures. To get an overview including a small description of all files that can be generated by `MidasCpp` we refer to section B in the appendix.

Some `MidasCpp` calculations assume a special folder structure to be present to read in additional input files or to put additional output files at specific places.

- In section 3.2.2 you find a description on the folder structure for PES calculations.

This is mostly the case when running PES type calculations.

## 1.3 Input

In the this section a general overview about the structure of `MidasCpp` input files is presented, but for specific example to getting started we refer to the example suite.

Input should reside in a file called `Midas.minp` in the main directory used for the calculation. `MidasCpp` first reads and checks this input file and thereafter calls the various sub-modules. The input is structured in levels. The keyword level is indicated by #1, #2 etc. `MidasCpp` interprets keywords case insensitive and blank characters are removed in the pre processing step. A particular example of an input for a calculation doing actually nothing is the following:

```
#0 Midas input                    // Everything before this is ignored
#1 General                        // The general input section
#2 IoLevel                        // Sets the general level of output
2
#0 Midas Input End                // Everything after this is ignored
```

This short input example simply sets the general `IoLevel`, dictating the amount of output generated. This can be done at a general level or for particular functionalities.

Everything occurring after // is considered a comment, as are all lines between an initial /∗ and a final ∗/ (similar to C++ programming conventions). These can be used to make comments for input files. In addition all lines beginning with ! are ignored, which makes it quick to remove a keyword from an input, without deleting it.

In some sections, the input can define that information from other files are used, such as definitions of operators etc. These files must then also be copied to the directory where the calculation takes place.

To get a feel for the hierarchical structure here is a longer template for input.

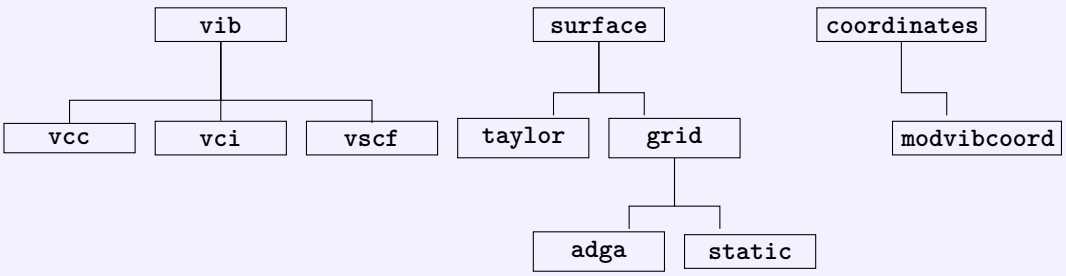
```
#0 Midas input                    // Everything before this is ignored
#1 Vib                            // Now comes vibrational input section
#2 Operat                         // Section defining operator
#3 ....                           // various operator options
....
#2 Basis                          // Section defining one-mode basis
#3 ....                           // various basis options
....
#2 Vscf                           // Section defining VSCF calculations
#3 .......                        // various vscf options
#2 Vcc                            // Section defining correlation calculations:
                                  // VCC, VCI and VMP
#3 .......                        // various vcc options
....
#3 Rsp                            // Response calculations for each VCC state
#4 .......                        // various response options
....
#3 .......                        // back to various vcc options
....
#0 Midas Input End                // Everything after this is ignored
```

## 1.4 Getting started with the example suite

We outline in this introduction only some very general aspects of the input to `MidasCpp`. We refer to the example suite https://gitlab.com/midascpp/midascpp_examples for illustrative examples and section III in the reference manual III for a more detailed overview of the available keywords. It is highly recommended to use the example suite for getting started with `MidasCpp`.

The example suite is hierarchical build

**Folder structure of the example suite**




to give a natural structure to search for the kind of `MidasCpp` calculation you are interested in. We mainly distinguish between the construction of various surfaces (surface) e.g. potential energy surface (PES) or dipole surfaces, vibrational wavefunction calculations (vib) and the generation of specialized vibrational coordinates (coordinates). In a typical application scenario first a PES has to be generated, which is then used in a subsequent vibrational structure calculation. We recommend to follow the same steps, when looking at the examples.

It has to be noted that the PES construction requires external electronic structure (ES) programs. `MidasCpp` has a generic interface and can be used in combination with any ES program, but in the example suite you find examples using Gaussian, MolPro, ORCA, TURBOMOLE, Dalton and Cfour.

Further instructions can be found in `README.md` of the example suite. Each example has a `README.md` itself, which appears nicely highlighted on GitLab. We recommend using the GitLab page to browse the example suite.

# Chapter 2

# Tutorial

In the following we will go through a few examples in the example suite in more detail.

## 2.1 Running a VSCF calculation

First, we will take a look at the example in `vib/vscf/h2o_energy`. If you run the example, you will also generate the following input. For information consider the comments in the input given below.

```
#0 MidasInput
   #1 General           // General input
      #2 IoLevel        // Set general (minimum) IO level
         5

   #1 Vib               // Vibrational input
      #2 Operator       // Operator input
         #3 OperFile    // Operator file (contains only the PES here,
            h2o_h0.mop  // but kinetic energy is added as default)

      #2 Basis          // Basis input
          #3 HoBasis    // Harmonic oscillator-basis up to
             10         // quantum number n=10

      #2 Vscf           // VSCF input (if only one basis and operator
                        // are defined, VSCF uses those as default)
#0 MidasInputEnd
```

It is assumed that the operator file `h2o_h0.mop` containing an analytical representation of the potential is located in the same folder. Such an operator file is usually generated during a PES construction and we refer to more details on this issue to Sec. 3.2. To run the example execute the command

```
midascpp h2o_energy.minp
```

in your terminal and an output file called `h2o_energy.mout` will be generated. In the file you will find information on the convergence

```
 Iter. 0 E_vscf = 2.1465343912674632E-02
 Iter. 1 E_vscf = 2.1067423815696335E-02 (decrease:  3.9792009697829644E-04,rel=1.8887933354329974E-02)
```

```
Iter. 2 E_vscf = 2.1059036014945329E-02 (decrease:  8.3878007510057584E-06,rel=3.9829936873905542E-04)
Iter. 3 E_vscf = 2.1059032750299238E-02 (decrease:  3.2646460916552211E-09,rel=1.5502355356794972E-07)
Iter. 4 E_vscf = 2.1059032748838649E-02 (decrease:  1.4605885945151442E-12,rel=6.9356869896870827E-11)
Iter. 5 E_vscf = 2.1059032748837983E-02 (decrease:  6.6613381477509392E-16,rel=3.1631738395575200E-14)
Iter. 6 E_vscf = 2.1059032748837990E-02 (decrease:  -6.9388939039072284E-18,rel=-3.2949727495390822E-16)
Vscf scf00_0.0                converged!  E = 2.1059032748837990E-02 in N_it = 6
Iter. 6 E_vscf = 2.1059032748837993E-02 (decrease:  -3.4694469519536142E-18,rel=-1.6474863747695408E-16)
```

and the VSCF energy given in several units.

```
Calculations ordered after operator/basis/name/increasing energy:
+-------------------------------------------------------------------+
VSCF: scf00_0.0               E =  2.1059032748837990E-02 Occ: [0,0,0] au


Units is converted with 2.1947463137049999E+05 corresponding to au to cm-1


Calculations ordered after operator/basis/name/increasing energy:
+-------------------------------------------------------------------+
VSCF: scf00_0.0               E =  4.6219234495705050E+03 Occ: [0,0,0] cm-1


Units is converted with 2.7211384500000001E+01 corresponding to au to eV


Calculations ordered after operator/basis/name/increasing energy:
+-------------------------------------------------------------------+
VSCF: scf00_0.0               E =  5.7304543732672253E-01 Occ: [0,0,0] eV
```

If `MidasCpp` ended properly (without crash) the last thing it will print out is

```
Midas ended at Tue Jan 22 11:37:38 2019
```

where the date of course varies.

## 2.2 Constructing a potential energy or molecular property surface

The first example, which showcased a VSCF energy calculation, was a bit unusual given that a predefined potential was used. The molecular potential usually have to be constructed first, before a vibrational structure calculation can be performed. `MidasCpp` offers several different methods of surface construction via the PES module, which will be detailed in subsequent parts of the manual, see chapter 6. In the example, the ADGA will be used in order to construct a potential energy surface for water with single point calculations at the CCSD(F12*)(T)/cc-pVDZ-F12 level of theory by running the TURBOMOLE program. This is similar to the example `surface/grid/adga/std_h2o_turbomole_mp2dip`, as found in the example_suite of `MidasCpp`. The input file looks something like the following:

```
#0 MidasInput

   #1 General
      #2 MainDir
         <pwd>

   #1 SinglePoint
```

```
      #2 Name
         generic_turbomole_spc
      #2 Type
         SP_GENERIC
      #2 InputCreatorScript
         InputCreatorScript.sh
      #2 RunScript
         RunScript.sh

   #1 System
      #2 MoleculeFile
         Midas
         <pwd>/std_h2o_turbomole_mp2dip.mmol

   #1 Pes
      #2 AdgaInfo
         2
      #2 MultilevelPes
         generic_turbomole_spc 2
      #2 AnalyzeStates
         [3*4]

   #1 Vib
      #2 Operator
         #3 OperFile
            <pwd>/savedir/prop_no_1.mop
         #3 KineticEnergy
            Simple

      #2 Basis
         #3 BSplineBasis
            20
         #3 PrimBasisDensity
            0.8
         #3 ScalBounds
            1.5 20.0

      #2 Vscf

   #1 Analysis

#0 MidasInputEnd
```

Additional files will be needed in order to perform the surface construction calculation, which are described in section 3.2. The purpose here is to describe the different blocks of input. The first input block specifies the location of the main directory, where the calculation will take place. In this example, it is set to the default path.

```
#1 General
   #2 MainDir
      <pwd>
```

The next input block is used to define how the single point calculations should be carried out, (consult chapter 14 for additional references). In this example a generic style single point calculation is choosen, which means that the user will have to provide scripts to allows for interfacing `MidasCpp` with a quantum chemistry program of his or her choosing. Two scripts are required, the input creator script which creates the individual single point input file and a runscript which runs the electronic structure program input file and extracts the required information, e.g. energy or other molecular properties.

```
#1 SinglePoint
   #2 Name
      generic_turbomole_spc
   #2 Type
      SP_GENERIC
   #2 InputCreatorScript
      InputCreatorScript.sh
   #2 RunScript
      RunScript.sh
```

The runscript communicates the final result of each single point calculation to `MidasCpp` by means of the property info file (needs to be created in the `InterfaceFiles` directory), which labels the molecular properties. In the present case potential energy and dipole surfaces are requested and the property info have the following structure:

```
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)
```

A property is here identified by the associated tensorial order, a label, how the element should be rotated and the which element of a tensor it is. The two latter options only need to be set of the tensorial order to higher than one. **Task:** Try to figure out how the generation of this file is realized in the RunScript (`RunScript.sh`), which you can find in the folder `InterfaceFiles`. The next input block is used to define the molecular system for which the surfaces is to be constructed. A file with the molecule information is all that is needed here, containing e.g. reference structure in Cartesian coordinates, harmonic frequencies, vibrational coordinates.

```
#1 System
   #2 MoleculeFile
      Midas
      <pwd>/std_h2o_turbomole_mp2dip.mmol
```

The next input block indicates directly in which directories, (see section 3.2) and how the surface construction should be performed. The use of the ADGA as surface generation strategy is indicated, using the electronic structure method previously defined. Furthermore it is indicated that the surfaces should include up to pair-mode couplings, which in most cases will provide reliable surfaces. The ADGA uses change in the vibrational density as a guide to where single points should be placed

for maximum coverage and it is indicated that each of the three vibrational modes in water should have a basis of four modals when performing VSCF calculations for determining the vibrational density.

```
#1 Pes
   #2 AdgaInfo
      2
   #2 MultilevelPes
      generic_turbomole_spc 2
   #2 AnalyzeStates
      [3*4]
```

The next input block specifies how the VSCF calculation should be performed. The VSCF module needs to know where it can find the current potential energy operator, (which is constructed by the ADGA in each iteration). The VSCF also needs a primitive basis in which to expand the modals and in this case a set of B-spline functions with a fixed density is chosen. Furthermore the wave function is allowed to extend beyond the potential grid boundaries, in order for the ADGA to investigate if significant vibrational density is found outside of the potential grid, which can then be expanded.

```
#1 Vib
   #2 Operator
      #3 OperFile
         <pwd>/savedir/prop_no_1.mop
      #3 KineticEnergy
         Simple

   #2 Basis
      #3 BSplineBasis
         20
      #3 PrimBasisDensity
         0.8
      #3 ScalBounds
         1.5 20.0

   #2 Vscf

#1 Analysis
```

The calculation itself might take some time, as something around 1000 single point calculations are needed in order to converge the surface. After the calculation is completed you can check the output file. At the bottom you find the VSCF energy obtained for the final potential. If you scroll up a bit, you should find a block like

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                            C
C Checking Adga convergence for 2-mode grids                                 C
C                                                                            C
C Number of converged MCs    : 3                                             C
C Number of non-converged MCs: 0                                             C
C                                                                            C
C !!! Adga procedure for 2-mode grids is converged !!!                       C
C                                                                            C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

telling you that the ADGA procedure converged. The objective of this calculation was to obtain analytical representations of the potential energy and dipole surfaces. They are stored in the folder `FinalSurfaces/savedir`. Here you can find the files

- `prop_no_1.mop` : Analytical representation for the PES
- `prop_no_2.mop` : Analytical representation for the $x$ component of the dipole surface.
- `prop_no_3.mop` : Analytical representation for the $y$ component of the dipole surface.
- `prop_no_4.mop` : Analytical representation for the $z$ component of the dipole surface.

**Task:** After performing this example, make a similar example with a electronic structure method and a QCP of your choice. For this purpose take a look at the scripts in the folder `InterfaceFiles`.

# Chapter 3

# More detailed information

## 3.1 Advanced installations

There are many ways to configure MidasCpp to accommodate many different systems and architectures. This is handled by the configure script, which sets up an appropriate Makefile.config for the system in question. For a comprehensive overview of the many options for configuring the building of MidasCpp executables, run

```
./configure --help
```

### 3.1.1 Compiling with MPI and OpenMP

To compile and run MidasCpp using MPI add --enable=mpi=openmpi to commandline when calling the configure script

```
./configure --prefix=/path/to/install --enable-mpi=openmpi
```

Note that OpenMPI must be available on your system in this case. We recommend to compile and run MidasCpp using OpenMPI version 4.0.0 or above. It is possible to configure and run with other MPI implementations, but the use of these has not been extensively tested. After configuration MidasCpp can be compiled and installed as usual by running

```
make complete
```

To run using MPI run midascpp with the mpirun wrapper.

```
mpirun <mpi_args> --mca mpi_warn_on_fork 0 midascpp <minp> --maindir <maindir>
```

MidasCpp must always be run with the --mca mpi_warn_on_fork 0 option when running with MPI. Also note that PES calculations run with MPI should always be run using an ethernet interface, and not Infiniband or any other fast network if such are available on the system.

Some parts of MidasCpp has been also parallelized with OpenMP. To enable this optimization, configure with --enable-openmp before compiling.

```
./configure --prefix=/path/to/install --enable-openmp
```

Note that using either MPI or OpenMP will make the code run non-determisticly (like any efficient numeric code running in parallel), which means that you will not get the exact same result with two consecutive runs. Of course results are still equally good w.r.t. to numeric thresholds provided to the calculation.

### 3.1.2 External libraries

`MidasCpp` also relies on different external libraries, including some optional math libraries that can be toggled on or off as needed. The `MidasCpp` build system can determine which libraries are present, and download and build any missing libraries automatically. Controlling which libraries are downloaded, and which versions to download is done by configuring with `--enable-<library>=<version>`, where providing a version number with `<version>` is optional. When not providing a version number a suitable default version will be downloaded. E.g. to download, build and link against a newer version of LAPACK than the default, configure with

```
./configure --prefix=/path/to/install --enable-lapack=<version>
```

Linking against all libraries is turned on by default, but the use of all optional libraries can be turned off using `--disable-<library>`. This of course disables all features utilizing the disabled libraries (trying to use a disabled feature will result in a program error). Required and optional math libraries are listed in 3.1 including minimum recommended versions.

Table 3.1: External libraries

| **Package** | **URL** | **Version** |
|---|---|---|
| **Required** | | |
| lapack | http://www.netlib.org/lapack/ | 3.8.0 |
| **Optional** | | |
| fftw | http://www.fftw.org/ | 3.3.4 |
| gsl | https://www.gnu.org/software/gsl/ | 1.16 |
| boost | https://www.boost.org/ | 1.68.0 |
| Tana-Tnum | https://github.com/lauvergn/Tnum-Tana/ | 0.8 |

The default URL for downloading each external package can also be overwritten using the `--with-<library>-url` configure options. With this option a mirror can be used to download if the default site is not responding. With this option one can also provide a path to a local file, which has been downloaded previously. To build `MidasCpp` with *e.g.*, LAPACK supplied from a local pre-downloaded file, configure with:

```
./configure --enable-lapack=<version> --with-lapack-url=/path/to/lapack-<version>.tar.gz
```

### 3.1.3 Vector inputs

Some keywords prompt the user to give a vector as input. `MidasCpp` has implemented a few shortcuts for defining vectors which might come in handy to avoid typing out long vectors. Usually the vectors are meant to be space separated unless otherwise stated for the specific keyword, *i.e.* the string "1 2 3 5" is read as the vector (1, 2, 3, 5). Multiple separators in a row will be ignored, *i.e.* the string "1–2" is read as the vector (1, 2) when '-' is used as separator.

The following characters will be treated special, namely ',', '(', '[' and '\'. To input these characters, or the separator, use '\' combined with the special character or separator, *i.e.* "1 2 \( 3" would be read as the vector (1, 2, (, 3).

To further tricks can be useful in generating simple input for complex tasks. The first is the range which is implemented to transform expressions of the type $[i..N;s]$ to a vector, $i$ is the first element, $N$ is the end of the range and $s$ is the stride. This results in the string "[1..5;2]" expands to the vector (1,3,5). Alternatively the stride, $s$, can be left out and will then by default be 1, resulting in the string "[1..5]" is converted to the vector (1, 2, 3, 4, 5). The second is the expansion

which takes the following form $[n * number]$ where $n$ is the number of repetitions. This allows the user to input the following string "[2*5]" which the program translates to (5, 5). The user can also use a nested range for repetition "[2*[1..5]]" will result in the vector (1, 2, 3, 4, 5, 1, 2, 3, 4, 5).

## 3.2 Potential energy or molecular property surface calculations

There are two main strategies towards surface generation available in MidasCpp, namely via Taylor expansion, see keyword #2 TaylorInfo, or by construction of a grid of single points. In the latter case the grid can be specified by the user prior to a calculation, i.e. a static grid approach, see #2 StaticInfo, or it can be dynamically constructed with the adaptive density-guided approach (ADGA), see #2 AdgaInfo.

It is important to carefully consider the mode combination level, (sometimes referred to as coupling order), before initializing a calculation with any of these surface generation strategies as this will define the dimensionality of the resulting surfaces. This, in turn, will define the accuracy and overall computational cost of the surface.

### 3.2.1 Electronic structure programs

The choice of surface generation strategy does not limit the choice of electronic structure method or program that can be used in any way. Indeed, `MidasCpp` is not an electronic structure program but interfaces with other such programs in order to obtain the electronic energy and/or properties for individual single points. This means that interfacing to different electronic structure programs is entirely independent of the surface generation strategy and will only need to be changed if another electronic structure method or program is desired. The `MidasCpp` example suite currently features the use of some standard electronic structure programs but this is by no means a limiting list.

**Electronic structure programs featured in the `MidasCpp` example suite**

- CFOUR
- DALTON
- Gaussian
- Orca
- MolPro
- Turbomole

### 3.2.2 Folder structure

The folder structure used for surface calculations is organized in two levels, first after the number of subsystems and then after the number of different electronic structure method employed. This means, that each surface calculation will effectively be a subsystem multilevel surface calculation, even though only the system might only consist of a single subsystem and only a single electronic structure method is used.

In the main directory, which is where a surface calculation is started there will be three directories present, named **`FinalSurfaces`**, **`InterfaceFiles`** and **`System`**. There will additionally be two soft-link directories present named **`savedir`** and **`analysis`**. The **`FinalSurfaces`** directory will contain the finished surface(s) of the entire calculation and in case that the system is divided into subsystems, then the corresponding surfaces will have been merged before placed in the folder.

The **`System`** directory contains one or more directories named **`subsystem_j`**, $j = 0, 1, 2, \ldots$ depending on how many structures have been defined by the #2 MoleculeFile keyword. Each **`subsystem_j`** directory will contain two directories named **`Multilevels`** and **`FinalSurfaces`**.

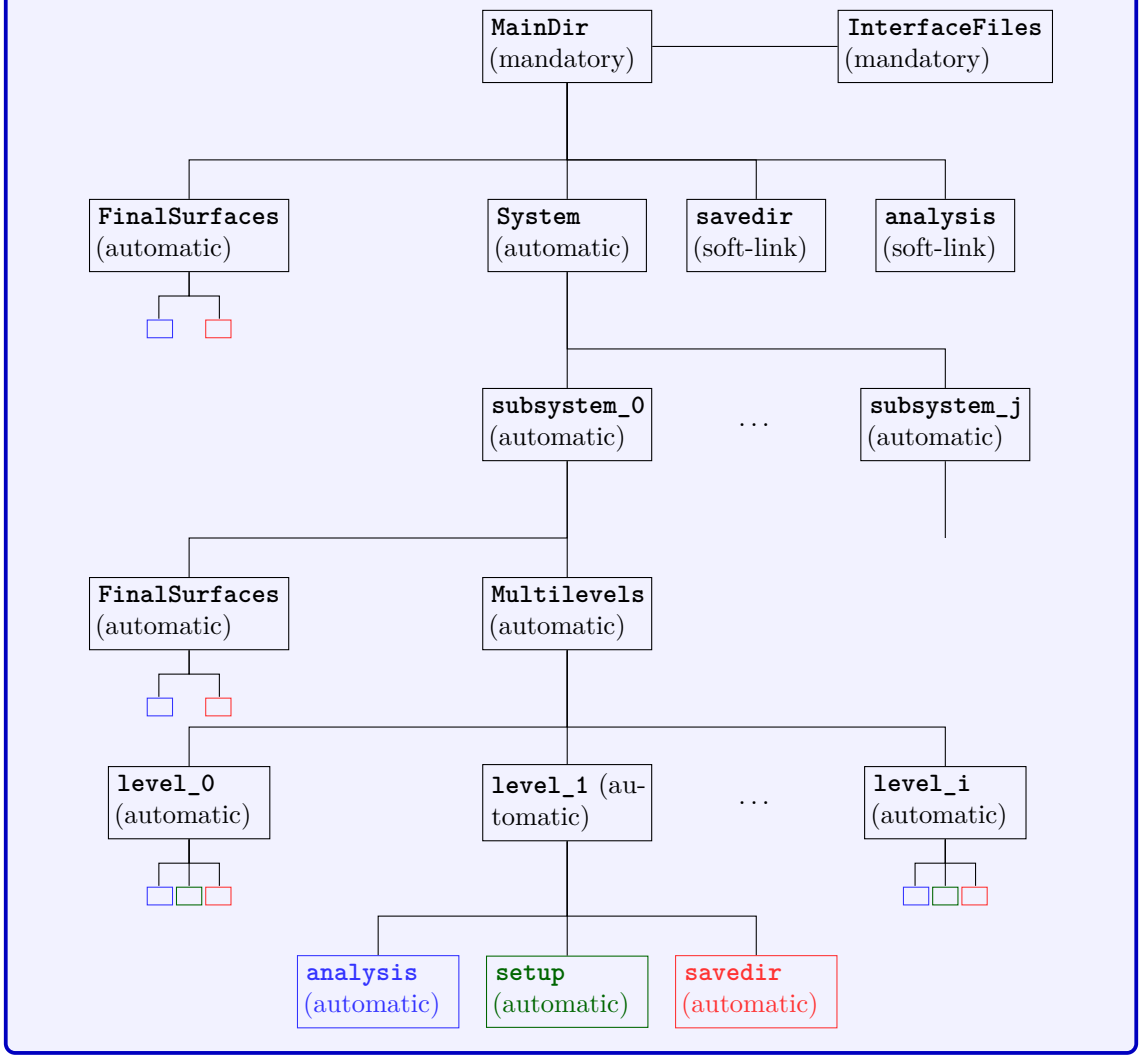
The **Multilevels** directory contains one or more directories named **level_i**, $i = 1, 2, 3, \ldots$ depending on how many and in which order electronic structure methods are specified under the #2 MultiLevelPes keyword. Note that a directory named **level_0** will be present if the #2 BoundariesPreOpt keyword is in use.

Each **level_j**, $j = 0, 1, 2, \ldots$ directory will contain three sub-directories named **savedir**, **analysis** and **setup**. The **savedir** sub-directory contains all collected information related to the calculated single points, the fitted and/or interpolated surfaces, restart information and properties related to the molecular structure, e.g. moment of inertia. The **analysis** sub-directory contains plots of the fitted and/or interpolated surface, information of the grid boundaries and plots of the mean or maximum density if an ADGA towards surface generation is in use. The **setup** sub-directory contains the interface files between **MidasCpp** and the external electronic structure program, i.e. a script for generating input, a script for running and a script for validating the electronic structure program, as indicated through the #2 InputCreatorScript, #2 RunScript and #2 ValidationScript keywords respectively. There should additionally be a file present that indicates the property or properties that are calculated by the electronic structure program and should be treated by **MidasCpp**. This is indicated by the #2 PropertyInfo keyword.

The **FinalSurfaces** directory will contain two sub-directories, **savedir** and **analysis**, which will contain the final multi-level surface(s) and grid dimensions respectively. If more than one electronic structure method are in use, then these final files will be merges of the corresponding files found in the **level_i** sub-directories according to the specifications given under the #2 MultiLevelPes keyword. If only one electronic structure method is specified, then the files found in the **FinalSurfaces** directory will be identical to those found in the **level_1** directory.

The soft-link directories found in the main directory will point to different **savedir** and **analysis** sub-directories in the **level_j** directories depending on which electronic structure method is used to generate the surface at a given moment.

**Surface calculation folder structure overview**




### 3.2.3 Commencing a calculation

`MidasCpp` will in general create all the directories described above but unless the internal electronic structure input options, specified under the #2 DaltonInput keywords is used, then the **`MainDir`** and **`InterfaceFiles`** directories will need to be created before commencing a calculations.

The **`MainDir`** directory should contain an input file with the keywords relevant for the desired calculation in the format detailed in part III of this manual and a file containing information about the molecular structure, as indicated by the #2 MoleculeFile keyword. The **`InterfaceFiles`** directory should contain four files for each **`level_j`** directory specified via the keywords

- #2 InputCreatorScript
- #2 RunScript
- #2 ValidationScript
- #2 PropertyInfo

as indicated above, which will be copied automatically to the associated **`setup`** sub-directory. Note that the validation script is not strictly mandatory but generally recommendable to include.

Note that MidasCpp will automatically create all the additional folder structure and the soft-link sub-directories if this is not created by the user. Should these directories already be present, then they will not be deleted or overwritten by MidasCpp but used as they are with what they contain, e.g. for restart purposes. This furthermore means that for restart calculations, where the folder structure is already in place, the InterfaceFiles directory is not strictly required.

## 3.3 Running calculations with general curvilinear coordinates

The deformational KEO in a general (curvilinear) coordinate system can be stated as

$$\hat{T}_{\text{def}} = -\frac{1}{2}\sum_{i,j=1}^{3N-6}\frac{1}{\rho(\mathbf{Q})}\frac{\partial}{\partial Q_i}\rho(\mathbf{Q})G_{ij}(\mathbf{Q})\frac{\partial}{\partial Q_j} + V_{\text{ep}}(\mathbf{Q}) \quad , \tag{3.1}$$

where $\mathbf{Q}$ denotes the set of $3N-6$ internal deformational coordinates, $G_{ij}$ are the contravariant components of the metric tensor and $V_{\text{ep}}$ is the scalar extrapotential, stemming from the normalization convention. The function $\rho$ describes the selected normalization convention, corresponding to the volume element $d\tau_{\text{def}} = \rho(\mathbf{Q})dQ_1dQ_2\ldots dQ_{3N-6}$. When using general coordinates in MidasCpp $\rho$ is always chosen to be one.

A key aspect of employing general coordinate systems in MidasCpp is to ensure that the corresponding kinetic energy operator always displays a sum-of-products form. To this end, MidasCpp can work with variants of the general coordinate KEOs that are always characterized accordingly, in particular approximations that are based on Taylor and $n$-mode expansions of $G_{ij}$ and $V_{\text{ep}}$. In addition to that, there is a particular class of curvilinear coordinates for which the full analytical KEO can always be set up in a sum-of-products form and thus be used in MidasCpp directly. These are the polyspherical coordinates. A further prerequisite for using general coordinates in MidasCpp is an interface to the TNUM/TANA software package of David Lauvergnat and coworkers [2–4], which is used to perform coordinate transformations between curvilinear and Cartesian coordinates as well as to calculate the G matrix, extrapotential and the full KEO (the latter only for polyspherical coordinates). TNUM/TANA can be installed by adding `--enable-tana` to the command line when calling the `configure` script (see Section 3.1) during the MidasCpp installation process.

All general coordinate KEO variants can be used in vibrationally correlated calculations with an existing PES, but also in PES constructions with the ADGA. For the use in vibrationally correlated calculations, the relevant information to build the KEO has to be provided at input stage, in terms of coefficient files (Taylor) or `.mop` files (n-mode, full). Running PES and molecular property constructions in general curvilinear coordinates requires a starting KEO (Taylor, n-mode or full), too. Furthermore, a TNUM/TANA script is needed that handles the coordinate transformations between Cartesian and internal coordinates as well as (optionally) the calculation of the G matrix and extrapotential at the displaced structures (keywords #2 CoordTrafoScript, #2 CalcGMatrix and #2 CalcExtraPotential under #1 Pes). As of now, the application of general coordinate system is restricted to coordinates that are expressed in terms of distances, angles and dihedral angles.

### 3.3.1 Polyspherical coordinates

When using polyspherical coordinates, a sum-of-products representation of the analytical KEO can be generated before starting MidasCpp. To this end, a reference structure and a suitable coordinate parametrization have to be defined in a script that is run with the `Tnum90_MidasCpp.exe` executable from the TNUM/TANA distribution. A corresponding bash script could look like the following

```
#!/bin/bash

# script with Tana input:
# weight function for normalization is 1
# total angular momentum J=0

Tnum90_MidasCpp.exe << ** > tana.out
 &variables
   nrho=1
   JJ=0
   Old_Qtransfo=f
   nb_Qtransfo=3
   Tana=t
   MidasCppForm=t
 /

 &Coord_transfo name_transfo='bunch' inTOout=f nb_vect=2 / Define bond vectors
   O H H
   1 2 (R1 = O->H1)
   1 3 (R2 = O->H2)

 &Coord_transfo name_transfo='poly' / Transformation to polyspherical coordinates
   &vector nb_vect=1 Frame=t cos_th=f / Vector defining z-axis in BF frame
      &vector nb_vect=0 Frame=f cos_th=f /

 &Coord_transfo name_transfo='active' / All internal coordinates are active
   1   1 1

 &minimum Read_nameQ=t read_Qsym0=f read_xyz0=t unit='bohr' / Reference geom.
 O  0.000000000000E+00  0.000000000000E+00 -7.394405002748E-01
 H -1.431340452709E+00  0.000000000000E+00  3.697202501374E-01
 H  1.431340452709E+00  0.000000000000E+00  3.697202501374E-01

**
```

Running this script produces a `MidasCpp_KEO.mop` file that contains the full analytical KEO in a sum-of-products form as well as a `Molecule.mmol` file with the reference structure that are directly readable by `MidasCpp`. In addition to that, it also produces Taylor expansions of the G matrix and extrapotential (`Taylor_G.keo`, `Taylor_Vep.keo`) at the reference structure (which can be used to set up a Taylor expansion-based KEO in `MidasCpp`).

For correlated calculations with a preexisting PES, only the `.mop` files for the KEO and PES have to be provided under the #2 Operator keyword and thus these files are sufficient to run `MidasCpp`:

```
#0 MidasInput
...

#1 Vib
```

```
   #2 Operator
      #3 Name
         h0
      #3 OperFile                // Potential Energy Operator
         <path to potential energy surface operator file >/<filename_pes>.mop
      #3 KineticEnergy           // Kinetic Energy Operator
         Polyspherical
         #4 KeoFile
            <path to kinetic energy operator file>/<filename_keo>.mop
   #2 Basis
      ...
   #2 Vscf
      ...
   #2 Vcc
      ...

#0 MidasInputEnd
```

When constructing a potential energy (or molecular property) surface, further coordinate-specific information is required under the #1 Pes and #2 Operator keywords. An exemplary `MidasCpp` input illustrating this for polyspherical coordinates is

```
#0 MidasInput
...

#1 System
   #2 MoleculeFile
      Midas
      <path to mmol file specifying the geometry>/<filename>.mmol

#1 Pes
   #2 CoordinateType
      Curvilinear
   #2 CoordTrafoScript
      <name of the TANA runscript placed in the InterfaceFiles directory>
   #2 PscMaxGridBounds
      +0.0 -0.02 -0.02
   #2 CalcGMatrix
      True
   #2 CalcExtraPotential
      True
   #2 MultiLevelPes
      model_spc 2
   #2 AdgaInfo
      2
      vscf_adga
   #2 AnalyzeStates
      [3*4]
```

```
   #2 DynamicAdgaExt
   #2 ScalCoordInFit
      Off
   #2 PolyInterpolationPoints
      12

#1 Vib
   #2 Operator
      #3 Name
         h0
      #3 OperFile                // Potential Energy Operator
         <path to potential energy surface operator file >/<filename_pes>.mop
      #3 KineticEnergy           // Kinetic Energy Operator
         Polyspherical
         #4 KeoFile
            <path to kinetic energy operator file>/<filename_keo>.mop
   #2 Basis
      ...

   #2 Vscf
      ...

#0 MidasInputEnd
```

Under #2 MoleculeFile the molecular structure is specified. For this, the `.mmol` file should contain the Cartesian coordinates of the investigated molecule as well as the internal (polyspherical) coordinates (#1 InternalCoord). An example for a calculation of a water molecule in valence coordinates could look like the following:

```
#0 MidasMolecule

#1 XYZ
3 au
xyz coordinates
 O  0.000000000000E+00  0.000000000000E+00 -7.394405002748E-01
 H -1.431340452709E+00  0.000000000000E+00  3.697202501374E-01
 H  1.431340452709E+00  0.000000000000E+00  3.697202501374E-01

#1 INTERNALCOORD
3 au rad
 1.8107934895553828 R1 Q0
 1.8107934895553828 R2 Q1
 1.8230843491285216 V2 Q2

#0 MidasMoleculeEnd
```

Moreover, under #2 CoordTrafoScript and #4 KeoFile the name of the runscript communicating between `MidasCpp` and TANA and the path and name of the previously generated kinetic energy

operator file are specified, respectively. The runscript is very similar to the KEO generating script depicted above, except that it additionally contains the displaced coordinates (in the curvilinear coordinates) for the current molecular geometry sampled by the ADGA. The corresponding information is produced within `MidasCpp` and added to the TNUM/TANA runscript under `&newQ`. In order to calculate the G matrix and extrapotential at the current displaced geometry, TNUM/TANA sets up a Taylor expansion of these quantities at the current structure. This is activated by specifying the `&newQ` keyword with the options `Taylor`, `GTaylor_order` and/or `VepTaylor_order`. Then, TNUM/TANA is executed, yielding output from which the file `XYZfromPSC` is generated, which is read by `MidasCpp`. For calculations of the G matrix and extrapotential, two additional files with Taylor expansion coefficients (`Taylor_G.keo` and `Taylor_Vep.keo`) are generated, from which `MidasCpp` reads the coefficients. An example for a runscript is given below.

```
#!/bin/bash

# Function for running Tana calculation with internal displacements supplied by Midas
run_tana()
{
   # Obtain displaced internal coordinates from a file written by MidasCpp
   displacements=""
   while IFS= read -r LINE
   do
      if ! [[ "$LINE" = "#" ]]
      then
         displacements+=$LINE$" "
      fi
   done < "DisplacedStruct"

   # Run Tana/Tnum with displaced internal coordinates
   Tnum90_MidasCpp.exe << ** > tana.out
&variables
  nrho=1
  JJ=0
  Old_Qtransfo=f
  nb_Qtransfo=3
  Tana=t
  MidasCppForm=t
/

&Coord_transfo name_transfo='bunch' inTOout=f nb_vect=2 / Define bond vectors
  O H H
  1 2 (R1 = O->H1)
  1 3 (R2 = O->H2)

&Coord_transfo name_transfo='poly' / Transformation to polyspherical coord.
  &vector nb_vect=1 Frame=t cos_th=f / Vector defining z-axis in BF frame
     &vector nb_vect=0 Frame=f cos_th=f /
```

```
&Coord_transfo name_transfo='active' / All internal coordinates are active
  1   1 1

&minimum Read_nameQ=t read_Qsym0=f read_xyz0=t unit='bohr' / Reference geom.
O  0.000000000000E+00  0.000000000000E+00 -7.394405002748E-01
H -1.431340452709E+00  0.000000000000E+00  3.697202501374E-01
H  1.431340452709E+00  0.000000000000E+00  3.697202501374E-01

&newQ  Taylor=t  GTaylor_order=0  VepTaylor_order=0 /
$displacements

**
}

# Function for extracting new (displaced) Cartesian coordinates from
# Tana output for MidasCpp
extract_new_xyz()
{
   N_ATOMS=`grep -A1 "XYZ format (bohr)" tana.out | tail -n1`

   echo -e "# XYZ coordinates are given with the following values: " > XYZfromPsC

   for i in $(seq 1 $N_ATOMS)
   do
      j=$(echo "$i + 2" | bc)
      XYZ_COORD_X=`grep -A$j "XYZ format (bohr)" tana.out | tail -n1 | awk '{print $2}'`
      XYZ_COORD_Y=`grep -A$j "XYZ format (bohr)" tana.out | tail -n1 | awk '{print $3}'`
      XYZ_COORD_Z=`grep -A$j "XYZ format (bohr)" tana.out | tail -n1 | awk '{print $4}'`

      echo $XYZ_COORD_X $XYZ_COORD_Y $XYZ_COORD_Z >> XYZfromPsC
   done
}

# Change directory to the scratch directory before invoking any functions
cd $1

run_tana
echo "Done running the Tana program"

extract_new_xyz
echo " Done extracting the new Cartesian coordinates from Tana"
```

Running the `MidasCpp` input for the ADGA calculation in polyspherical coordinates produces a potential energy surface operator file (typically named `prop_no_1.mop`), which together with the polyspherical KEO file can be used in subsequent VSCF or correlated calculations. If the keywords #2 CalcGMatrix and #2 CalcExtraPotential are specified (and the TNUM/TANA runscript produces Taylor expansions of $G$ and $V_{\mathrm{ep}}$), additional `prop_no_X.mop` files are generated that

contain $n \cdot (n+1)/2$ n-mode expansions for the (symmetric) G matrix and one n-mode expansion for the (scalar) extrapotential. Specific to the calculations in polyspherical coordinates is also the keyword #2 PscMaxGridBounds, which gives control over the extent of the displacements from equilibrium for the curvilinear coordinates. Further examples of calculations in polyspherical coordinates are given in the `MidasCpp` example suite.

### 3.3.2 Taylor expansion-based KEO variants

The Taylor expansion-based KEO variants are constructed from a TNUM/TANA-generated file that contains the expansion coefficients of the $G_{ij}$ matrix elements at the reference configuration. Optionally, a second file containing the coefficients for the expansion of the extrapotential term can be supplied. The files with the required Taylor expansions can be generated before starting `MidasCpp` from only the reference configuration and a suitable coordinate parametrization (see first TNUM/TANA runscript in Section 3.3.1). The mandatory input for a calculation using a Taylor expansion-based general coordinate KEO is specified under #2 Operator. An example of the minimum required input is

```
#1 Vib
   #2 Operator
      #3 Name
         h0
      #3 OperFile                // Potential Energy Operator
         <path to potential energy surface operator file >/<filename_pes>.mop
      #3 KineticEnergy           // Kinetic Energy Operator
         General
         #4 KeoExpansionType
            Taylor
         #4 GMatrixExpansion
            <order of Taylor expansion of G matrix>
            <path to Taylor exp. coeff. file of G>/<filename, exp. coeff. of G>

   #2 Basis
   ...
```

Further keywords under #3 KineticEnergy related to the Taylor expanded KEO variants are #4 AddExtrapotential, #4 ExtrapotentialExpansion, #4 KeoScreening, and #4 KeoScreeningThreshold, which are applicable to n-mode expansion-based general-coordinate KEOs as well, and #4 TaylorModeTruncApprox and #4 TaylorKeoGmatDiag, which are applicable only to Taylor expansion-based KEOs. The Taylor expansion-based KEOs can be used in potential energy surface constructions analogously to the full polyspherical KEOs (see Section 3.3.1 above).

### 3.3.3 $n$-mode expansion-based KEO variants

To use $n$-mode expansion-based KEO variants, `.mop` files containing the $n$-mode expansions of the $M \cdot (M+1)/2$ ($M$: number of modes) upper triangle $G_{ij}$ matrix elements and (optionally) the extrapotential have to be provided. The minimum required input under #3 KineticEnergy is

```
#1 Vib
   #2 Operator
      #3 Name
```

```
      h0
   #3 OperFile                // Potential Energy Operator
      <path to potential energy surface operator file >/<filename_pes>.mop
   #3 KineticEnergy           // Kinetic Energy Operator
      General
      #4 KeoExpansionType
         NMode
      #4 GMatrixExpansion
         <number of active modes>
         <path to n-mode exp. of G_{11} .mop file>/<name of G_{11} .mop file>

#2 Basis
...
```

The names of the .mop files containing the upper triangle of the $G$ matrix have to consist of a common basename and a consistently increasing index **combined with an underscore** (_), for instance **`Gmat_2.mop, Gmat_3.mop, Gmat_4.mop, ...`**. Under #4 GMatrixExpansion only the path and name of the first element of the $G$ matrix is given (alongside the number of modes). The `.mop` files for the $G$ matrix elements and the extrapotential can be generated with `MidasCpp` and TNUM/TANA in a standalone way using the static-grid facilities of the former, if the relevant coordinate domains are known in advance. Alternatively, a workflow of concatenating ADGA runs can be deployed to generate $n$-mode expansion-based PESs and KEOs simultaneously and without prior knowledge of the coordinate domains (see Section 3.3.4). In both cases, a TNUM/TANA runscript has to be provided to do the coordinate transformations from displaced internal coordinates to Cartesian coordinates. This runscript has to include the keyword `&newQ` with options `Taylor=t GTaylor_order=0 VepTaylor_order=0` to calculate the G matrix and extrapotential at the displaced geometries. In addition to that, the keywords #2 CalcGMatrix and #2 CalcExtraPotential have to be provided in the PES definitions. Additional keywords under #3 KineticEnergy related to the $n$-mode expanded KEO variants are #4 AddExtrapotential, #4 ExtrapotentialExpansion, #4 KeoScreening #4 KeoScreeningThreshold (all of them are applicable to the Taylor expansion-based KEO schemes as well).

### 3.3.4 Sequential ADGA workflow

For some general coordinate systems and applications, analytical representations of the full KEO are not available, while simple Taylor expansion-based KEO variants are not applicable. In these cases, a more flexible $n$-mode expansion-based KEO variant could be used. The main obstacle for this is the knowledge of the relevant coordinate domains, which is not necessarily obvious in advance. To circumvent this issue, `MidasCpp` provides a sequential ADGA workflow, which allows to construct high-level $n$-mode PES and KEO representations simultaneously, starting from a Taylor expansion-based KEO variant. The general approach in the sequential ADGA workflow is to concatenate multiple ADGA runs with step-wise increasing maximum mode coupling level. Each ADGA run uses $n$-mode representations of the KEO that are constructed in the preceding step, while the PES can be constructed either from scratch or starting from the data points obtained in the previous ADGA runs (default). Concerning the currently implemented interface to `MidasCpp`, the sequential ADGA workflow is invoked by defining a regular ADGA calculation for a single PES and providing the #2 SeqAdgaKeoBuild (and optionally #2 SeqAdgaStepFullRestart) under #1 Pes. The argument of #2 SeqAdgaKeoBuild is a vector containing the names of the operators

that are used in the different ADGA runs. These operators have to be defined in the `MidasCpp` input under #2 Operator. Corresponding PES and operator definitions are shown below. The PES definition is

```
...
#1 Pes
   #2 Name
      H2O        // specify a base name for the PESs for seq. ADGA runs
   #2 CoordinateType
      Curvilinear
   #2 CoordTrafoScript
      TanaRunScript.sh
   #2 CalcGMatrix
      True
   #2 CalcExtraPotential
      True
   #2 FitBasis
      #3 Name
         PBase1
      #3 FitBasisMaxOrder
      ...
   #2 UseFitBasis
      PBase1
   #2 MultiLevelPes
      generic_orca_spc 3
   #2 AdgaInfo
      3                // specify the final target mode coupling level
      vscf_adga
   #2 AnalyzeStates
   ...
   #2 SeqAdgaKeoBuild  // sequence of 3 ADGA runs using operators h0, h1 and h2, respectively
      h0  h1  h2
   #2 ScalCoordInFit // turn off frequency scaling in fits for general coordinates
      Off
...
```

In a corresponding operator input, a new PES (file) is defined under each #2 Operator keyword, while the KEOs (except the initial one) use the n-mode expansions obtained in the previous ADGA run. (Because the reference values for the G matrix elements are required for the KEO definition, the `.mop` files from the subsystem directories should be read.) The operator input could look like this:

```
...
#1 Vib
   // operator for first ADGA run
   #2 Operator
      #3 Name
         h0
```

```
      #3 OperFile                // Potential Energy Operator
         $MAINDIR/H2O_seqAdga_step_1/prop_no_1.mop
      #3 KineticEnergy           // Kinetic Energy Operator
         General
         #4 KeoExpansionType
            Taylor
         #4 GMatrixExpansion
            <0, 1 or 2>          // Order of Taylor expansion of G matrix and extrapotential
            <path to and name of coeffcient file>

   // operator for second ADGA run
   #2 Operator
      #3 Name
         h1
      #3 OperFile                // Potential Energy Operator
         $MAINDIR/H2O_seqAdga_step_2/System/subsystem_0/FinalSurfaces/savedir/prop_no_1.mop
      #3 KineticEnergy           // Kinetic Energy Operator
         General
         #4 KeoExpansionType
            NMode
         #4 GMatrixExpansion
            3                    // number of active modes and first G matrix .mop file
            $MAINDIR/H2O_seqAdga_step_1/System/subsystem_0/FinalSurfaces/savedir/prop_no_2.mop

   // operator for final ADGA run
   #2 Operator
      #3 Name
         h2
      #3 OperFile                // Potential Energy Operator
         $MAINDIR/H2O_seqAdga_step_3/System/subsystem_0/FinalSurfaces/savedir/prop_no_1.mop

      #3 KineticEnergy           // Kinetic Energy Operator
         General
         #4 KeoExpansionType
            NMode
         #4 GMatrixExpansion
            3
            $MAINDIR/H2O_seqAdga_step_2/System/subsystem_0/FinalSurfaces/savedir/prop_no_2.mop
...
```

There are a few **limitations** for the application of the sequential ADGA workflow:

1. As of now, the sequential ADGA workflow is not implemented for multilevel calculations!
2. Similarly, using different fitting bases for the different ADGA runs in the workflow is **not** tested!
3. The mode extension scheme is also **not** tested for the sequential ADGA workflow.

4. The names of and paths to the operator files should be provided explicitly, i.e. the absolute path to the file including the file name must be given!

5. After all ADGA calculations in the workflow are finished, a VSCF calculation with the respective operator is run for each of them. Additionally, another VSCF calculation is run for the VSCF definition provided in the input file.

# 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 4

# 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.

## 4.1 Normal-Coordinates and harmonic oscillator treatment of Molecular vibrations

### 4.1.1 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

$$\mathbf{R}_P = \mathbf{R}_P^0 + \mathbf{D}_P \tag{4.1}$$

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 $\mathbf{D}_P$

$$\dot{\mathbf{R}}_P = \dot{\mathbf{D}}_P \tag{4.2}$$

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

$$\mathbf{D}^T = (d_{1X}, d_{1Y}, d_{1Z}, d_{2X}, d_{2Y}, d_{2Z}, ....., d_{NX}, d_{NY}, d_{NZ}) \tag{4.3}$$

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

$$\mathbf{G} = \begin{pmatrix} M_1^{-1} & 0 & . & . & . & . & . & . & 0 \\ 0 & M_1^{-1} & & & & & & & . \\ . & & M_1^{-1} & & & & & & . \\ . & & & M_2^{-1} & & & & & \\ . & & & & M_2^{-1} & & & & \\ . & & & & & M_2^{-1} & & & \\ & & & & & & & & . \\ & & & & & & & & . \\ . & & & & & & & & . \\ 0 & . & . & & & & & . & M_N^{-1} \end{pmatrix} \tag{4.4}$$

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

$$T_n = \frac{1}{2}\sum_P (\dot{\mathbf{R}}_P)^T M_P \dot{\mathbf{R}}_P = \frac{1}{2}(\dot{\mathbf{D}})^T \mathbf{G}^{-1}\dot{\mathbf{D}} \tag{4.5}$$

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

$$\mathbf{D}_M = \mathbf{G}^{-\frac{1}{2}}\mathbf{D} \tag{4.6}$$

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

$$T_n = \frac{1}{2}(\dot{\mathbf{D}}_M)^T \dot{\mathbf{D}}_M \tag{4.7}$$

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

$$\mathbf{Q} = \mathbf{L}^T \mathbf{D}_M \tag{4.8}$$

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.(4.7). 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

$$\mathbf{L}^T\mathbf{L} = \mathbf{1} \tag{4.9}$$

$$\mathbf{L}\mathbf{L}^T = \mathbf{1} \tag{4.10}$$

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

$$\mathbf{D}_M = \mathbf{L}\mathbf{Q} \tag{4.11}$$

The kinetic energy maintain the simple form in Eq.(4.7) now in terms of the $\mathbf{Q}$ coordinates

$$T_n = \frac{1}{2}(\dot{\mathbf{Q}})^T \dot{\mathbf{Q}} \tag{4.12}$$

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)

$$V(\mathbf{D}) = V(\mathbf{0}) + \sum_{P=1}^{N_n}\sum_{\alpha}\frac{dV}{dD_{P\alpha}}|_{\mathbf{0}}D_{P\alpha} + \frac{1}{2}\sum_{P,Q=1}^{N_n}\sum_{\alpha,\beta}\frac{d^2V}{dD_{P\alpha}dD_{Q\beta}}|_{\mathbf{0}}D_{P\alpha}D_{Q\beta} + .... \tag{4.13}$$

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

$$F_{P\alpha Q\beta} = \frac{\partial^2 V}{\partial D_{P\alpha}\partial D_{Q\beta}}|_{\mathbf{0}} \tag{4.14}$$

and neglecting terms higher than second order we obtain

$$V(\mathbf{D}) - V(\mathbf{0}) \approx \frac{1}{2}(\mathbf{D})^T\mathbf{F}\mathbf{D} = \frac{1}{2}(\mathbf{D}_M)^T(\mathbf{G}^T)^{\frac{1}{2}}\mathbf{F}\mathbf{G}^{\frac{1}{2}}\mathbf{D}_M = \frac{1}{2}(\mathbf{Q})^T\mathbf{L}^T\mathbf{G}^{\frac{1}{2}}\mathbf{F}\mathbf{G}^{\frac{1}{2}}\mathbf{L}\mathbf{Q} = \frac{1}{2}(\mathbf{Q})^T\mathbf{\Lambda}\mathbf{Q} \tag{4.15}$$

In Eq.(4.15) we have made use of the fact that the first derivatives vanishes at equilibrium. We have also used the coordinate transformations in Eqs.(4.6,4.8) and that since $\mathbf{G}$ is diagonal $(\mathbf{G}^T)^{\frac{1}{2}} = \mathbf{G}^{\frac{1}{2}}$. We have in Eq.(4.15) introduced the matrix

$$\mathbf{\Lambda} = \mathbf{L}^T\mathbf{G}^{\frac{1}{2}}\mathbf{F}\mathbf{G}^{\frac{1}{2}}\mathbf{L} \tag{4.16}$$

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

$$\mathbf{G}^{\frac{1}{2}}\mathbf{F}\mathbf{G}^{\frac{1}{2}}\mathbf{L} = \mathbf{L}\mathbf{\Lambda} \tag{4.17}$$

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.(4.17) 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. (4.17) 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 $Q_k$

$$V(\mathbf{D}) - V(\mathbf{0}) \approx \frac{1}{2}\sum_k \lambda_k Q_k^2 \tag{4.18}$$

Often, the eigenvalue problem is formulated as

$$\mathbf{G}\mathbf{F}(\mathbf{G}^{\frac{1}{2}}\mathbf{L}) = (\mathbf{G}^{\frac{1}{2}}\mathbf{L})\mathbf{\Lambda} \tag{4.19}$$

which is easily obtained from Eq.(4.17) by multiplication with $\mathbf{G}^{\frac{1}{2}}$. Diagonalization of $\mathbf{G}\mathbf{F}$ thus determines the eigenvalues $\lambda_k$ 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

$$E_{HO} = \frac{1}{2}\sum_{k=1}^{3N_n-6}(\dot{Q}_k^2 + \lambda_k Q_k^2) \tag{4.20}$$

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.

### 4.1.2 Quantization of the vibrational energy

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

$$E_{HO} = \frac{1}{2}\sum_{k=1}^{M}(P_k^2 + \lambda_k Q_k^2) \tag{4.21}$$

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

$$H = \sum_{k=1}^{M} H_k \tag{4.22}$$

where

$$H_k = -\frac{1}{2}\hbar^2\frac{\partial^2}{\partial Q_k^2} + \frac{1}{2}\lambda_k Q_k^2 \tag{4.23}$$

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

$$H_k\psi_{n_k}^k(Q_k) = \hbar\omega_k(n_k + \frac{1}{2})\psi_{n_k}^k(Q_k) \tag{4.24}$$

with

$$\omega_k = \sqrt{\lambda_k} \tag{4.25}$$

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

$$\Psi(Q_1,,,,,Q_M) = \prod_{k=1}^{M}\psi^k(Q_k) \tag{4.26}$$

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.

## 4.2 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:

$$\tilde{Q}_k = \sqrt{\frac{\omega_k}{\hbar}}Q_k \tag{4.27}$$

Some basic results:

$$\frac{\partial^2}{\partial Q_k^2} = \frac{\omega_k}{\hbar}\frac{\partial^2}{\partial \tilde{Q}_k^2} \tag{4.28}$$

$$H_k = -\frac{1}{2}\hbar^2\frac{\omega_k}{\hbar}\frac{\partial^2}{\partial \tilde{Q}_k^2} + \frac{1}{2}\omega_k^2\frac{\hbar}{\omega_k}\tilde{Q}_k^2 \tag{4.29}$$

$$H_k = \hbar\omega_k\left(-\frac{1}{2}\frac{\partial^2}{\partial \tilde{Q}_k^2} + \frac{1}{2}\tilde{Q}_k^2\right) \tag{4.30}$$

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:

$$V(r - r_e) = D_e\left(1 - \exp\left(\alpha(r - r_e)\right)\right)^2 \tag{4.31}$$

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

$$V(\tilde{Q}_k) = D_e\left(1 - \exp\left(\tilde{\alpha}\tilde{Q}_k\right)\right)^2 \tag{4.32}$$

where

$$\alpha = \tilde{\alpha}\sqrt{\omega} \tag{4.33}$$

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.

## 4.3 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, $\alpha, \beta$, 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 $\mathbf{\Sigma}$ may be calculated (approximately) as

$$\sigma_{p\alpha,j} = \sum_{k\sigma} h_{p\alpha,k\sigma} b_{k\sigma,j} \tag{4.34}$$

$$= \sum_{k\sigma} b_{k\sigma,j} \frac{\partial}{\partial d_{k\sigma}} \frac{\partial V}{\partial d_{p\alpha}} \tag{4.35}$$

$$= \nabla_{\mathbf{b}_j} \left( \frac{\partial V}{\partial d_{p\alpha}} \right) \tag{4.36}$$

$$\approx \frac{\left(\frac{\partial V}{\partial d_{p\alpha}}\right)_{\mathbf{r}_0 + \delta_{\text{eff}} \mathbf{b}_j} - \left(\frac{\partial V}{\partial d_{p\alpha}}\right)_{\mathbf{r}_0 - \delta_{\text{eff}} \mathbf{b}_j}}{2\delta_{\text{eff}}}, \tag{4.37}$$

where $\nabla_{\mathbf{v}}(f)$ is the derivative of a function $f$ in the direction of $\mathbf{v}$, and $\delta_{\text{eff}}$ is an effective displacement factor $\delta_{\text{eff}} = \delta / \|\mathbf{b}_j\|$ that accounts for the possibility of $\mathbf{b}_j$ not being normalized. ($\delta$ is set by #3 DispFactor.)

What (4.37) 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 (4.34) 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 (4.37) 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 (4.34). If calculating *all* the columns of $\mathbf{\Sigma}$ 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 (4.37) 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 $\delta_{\text{eff}}$ 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 #3 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 #3 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

$$\mathbf{B}^{(k)} = (\mathbf{b}_1, \ldots, \mathbf{b}_k) \tag{4.38}$$

$$\mathbf{\Sigma}^{(k)} = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_k) \tag{4.39}$$

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

$$\tilde{\mathbf{H}}^{(k)} = \mathbf{B}^{(k)\dagger} \mathbf{\Sigma}^{(k)}, \tag{4.40}$$

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

$$\tilde{\mathbf{C}}^{(k)\dagger}\tilde{\mathbf{H}}^{(k)}\tilde{\mathbf{C}}^{(k)} = \tilde{\boldsymbol{\Lambda}}^{(k)} = \mathrm{diag}(\tilde{\lambda}_1, \ldots, \tilde{\lambda}_k), \tag{4.41}$$

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 (4.37). By default the reduced Hessian is made symmetric, but this can be switched off by #3 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 $n_{\mathrm{eq}}$ 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 $n_{\mathrm{eq}}$ most desirable ones are chosen for further study, $\{\tilde{\mathbf{c}}_1^{(k)}, \ldots, \tilde{\mathbf{c}}_{n_{\mathrm{eq}}}^{(k)}\}$. 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

$$\mathbf{r}_j^{(k)} = \mathbf{H}\mathbf{c}_j^{(k)} - \tilde{\lambda}_j\mathbf{c}_j^{(k)} = (\boldsymbol{\Sigma}^{(k)} - \tilde{\lambda}_j^{(k)}\mathbf{B}^{(k)})\tilde{\mathbf{c}}_j^{(k)}, \qquad j = 1, \ldots, n_{\mathrm{eq}} \tag{4.42}$$

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;

$$\epsilon_{\mathrm{res}} \geq \left\| \mathbf{r}_j^{(k)} \right\|, \qquad \text{for } j = 1, \ldots, n_{\mathrm{eq}} \tag{4.43}$$

$$\epsilon_{\mathrm{energy}} \geq \left| \frac{\lambda_j^{(k)} - \lambda_j^{(k-n_{\mathrm{eq}})}}{\lambda_j^{(k-n_{\mathrm{eq}})}} \right|, \quad \text{for } j = 1, \ldots, n_{\mathrm{eq}}, \tag{4.44}$$

where $\epsilon_{\mathrm{res}}$ (#3 ItEqResThr) and $\epsilon_{\mathrm{energy}}$ (#3 ItEqEnerThr) are predefined thresholds and $(k - n_{\mathrm{eq}})$ signifies the values obtained in the previous iteration.

If all equations in (4.43) and (4.44) 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,

$$s_{ij}^{(k)} = \langle \boldsymbol{\tau}_i | \mathbf{c}_j^{(k)} \rangle = \langle \boldsymbol{\tau}_i | \mathbf{B}^{(k)}\tilde{\mathbf{c}}_j^{(k)} \rangle = \langle \tilde{\mathbf{o}}_i | \tilde{\mathbf{c}}_j^{(k)} \rangle \,, \tag{4.45}$$

where

$$\tilde{o}_{li} = \langle \mathbf{b}_l | \boldsymbol{\tau}_i \rangle \tag{4.46}$$

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

$$\tilde{\mathbf{S}}^{(k)} = (\tilde{\mathbf{O}}^{(k)})^{\dagger} \tilde{\mathbf{C}}^{(k)}. \tag{4.47}$$

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

$$S_i^{(k)} = \sum_{j=1}^{n_{\text{overlap},i}} \left( s_{ij}^{(k)} \right)^2. \tag{4.48}$$

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, $S_i^{(k)} \geq \epsilon_{\text{overlap}}$ (#3 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, $n_{\text{overlap,max}}$ (#3 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.

## 4.4 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.

### 4.4.1 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. [5, 6]

**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

### 4.4.2 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.

### 4.4.3 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.[6]

**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

### 4.4.4 Hybrid Optimized and Localized Coordinates

Another possible set of coordinates are hybrid optimized and localized coordinates as described in Ref.[7]. 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* **2015**, *119*, 11007–11021

## 4.5 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. 6.3.7).

**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 5

# Kinetic energy operators

## 5.1 The Watson kinetic energy operator in normal coordinates

### 5.1.1 Non-linear molecules

For a non-rotating non-linear molecule, the Watson kinetic energy operator[9] (or Watsonian) is given by

$$T = \frac{1}{2}\sum_{k=1}^{M}\frac{\partial^2}{\partial Q_k^2} + \frac{1}{2}\sum_{\alpha\beta} j_\alpha \mu_{\alpha\beta} j_\beta - \frac{\hbar^2}{8}\sum_{\alpha}\mu_{\alpha\alpha}. \tag{5.1}$$

Here, $M$ denotes the number of vibrational modes and the second and third sums run over the Cartesian components $x$, $y$ and $z$. The terms will be denoted as the simple kinetic energy term, the Coriolis coupling term and the extra-potential (or pseudo-potential) term, respectively. The Coriolis coupling term contains the so-called vibrational angular momentum operators,

$$\begin{aligned} j_\alpha &= -i\hbar j'_\alpha \\ &= -i\hbar\sum_{kl} Q_k \zeta^\alpha_{kl}\frac{\partial}{\partial Q_l} \\ &= -i\hbar\sum_{k>l}\zeta^\alpha_{kl}\Big(Q_k\frac{\partial}{\partial Q_l} - Q_l\frac{\partial}{\partial Q_k}\Big). \end{aligned} \tag{5.2}$$

The last equality follows from the anti-symmetry of the Coriolis coupling matrices $\boldsymbol{\zeta}^\alpha$. The inverse effective moment of inertia tensor $\boldsymbol{\mu}$ is given by

$$\boldsymbol{\mu} = (\mathbf{I}')^{-1} \tag{5.3}$$

with

$$I'_{\alpha\beta} = I_{\alpha\beta} - (\boldsymbol{\zeta}^\alpha\mathbf{Q})^{\mathsf{T}}(\boldsymbol{\zeta}^\beta\mathbf{Q}). \tag{5.4}$$

Here, $I_{\alpha\beta}$ are the ordinary (nuclear) moment of inertia elements and $\mathbf{Q}$ is simply the vector containing the displacements along the normal coordinates. It should be noted that $\boldsymbol{\mu}$ generally depends on the coordinates, i.e. $\boldsymbol{\mu} = \boldsymbol{\mu}(\mathbf{Q})$. If an expansion of $\boldsymbol{\mu}$ is available, we may use Eq. (5.1) as it stands (we refer to this as the *expanded* Watson operator). A simpler approach (which we call the *equilibrium* Watson operator) is to approximate the $\boldsymbol{\mu}$ tensor by its equilibrium value (or, more generally, its value at the reference geometry), i.e.

$$\boldsymbol{\mu} \approx \boldsymbol{\mu}^0 = (\mathbf{I}'^0)^{-1} = (\mathbf{I}^0)^{-1}. \tag{5.5}$$

This approach requires only the equilibrium moment of inertia tensor $\mathbf{I}^0$ and is often sufficient[10].

Unless the molecule in question has very small moments of inertia (and thus large $\boldsymbol{\mu}$ elements), one may reasonably neglect the Coriolis coupling and extra-potential terms completely. This yields the *simple* kinetic energy operator.

### 5.1.2 Linear molecules

For linear molecules, the situation is more complicated due to the singular moment of inertia tensor. Watson showed[11] how to handle this complication. In addition, he showed how an extra transformation of the kinetic energy operator produces an expression that is similar to Eq. (5.1) while leaving the spectrum unchanged. For a non-rotating linear molecule, this operator (which is sometimes called the Hougen-Watson operator[11, 12]) is given by

$$T = \frac{1}{2}\sum_{k=1}^{M}\frac{\partial^2}{\partial Q_k^2} + \frac{1}{2}\sum_{\alpha\beta} j_\alpha \mu_{\alpha\beta} j_\beta \tag{5.6}$$

where the absence of an extra-potential (pseudo-potential) term should be noted. The vibrational angular momentum operators are defined as in Eq. (5.2). The Coriolis coupling term simplifies considerably since for a linear molecule

$$\boldsymbol{\mu} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{5.7}$$

$$[j_\alpha, \mu] = 0, \tag{5.8}$$

allowing us to write

$$\begin{aligned} T &= \frac{1}{2}\sum_{k=1}^{M}\frac{\partial^2}{\partial Q_k^2} + \frac{1}{2}(j_x \mu j_x + j_y \mu j_y) \\ &= \frac{1}{2}\sum_{k=1}^{M}\frac{\partial^2}{\partial Q_k^2} + \frac{1}{2}\mu(j_x^2 + j_y^2). \end{aligned} \tag{5.9}$$

As in the non-linear case, one may approximate or neglect the second term in reasonable cases.

# Chapter 6

# Potential energy surfaces

## 6.1 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 6.1 a model potential energy surface as function of two variables is given. Throughout we shall return to this figure and explain various features.

Figure 6.1: A model potential energy surface illustrating important stationary points and curves. (from Jack Simons:Introduction to theoretical chemistry)

## 6.2 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.

$$\hat{H}^{el}\Phi_I^{el}(\mathbf{r};\mathbf{R}) = E_I^{el}(\mathbf{R})\Phi_I^{el}(\mathbf{r};\mathbf{R}) \tag{6.1}$$

$$\hat{H}^{el} = \hat{T}_e + \hat{V}_{ee} + \hat{V}_{en} + \hat{V}_{nn} \tag{6.2}$$

- 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.

## 6.3 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

$$\hat{V} = V(q_1, q_2, ..., q_M) \tag{6.3}$$

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.

### 6.3.1 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 $N_t$

$$V(\mathbf{q}) = V(\mathbf{0}) + \sum_{n=1}^{N_t} \frac{1}{n!} \sum_{i_1 i_2, ... i_n} \frac{\partial^n V}{\partial q_{i_1} \partial q_{i_2} ... \partial q_{i_n}} q_{i_1} q_{i_2} ... q_{i_n} \tag{6.4}$$

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

$$|\Phi_{\mathbf{r}}\rangle = \prod_{m=1}^{M} \phi_{r^m}^m(q_m) \tag{6.5}$$

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

$$\begin{aligned} \langle \Phi_{\mathbf{r}} | V | \Phi_{\mathbf{s}} \rangle &= V(\mathbf{0}) \langle \Phi_{\mathbf{r}} | \Phi_{\mathbf{s}} \rangle + \sum_{n=1}^{N_t} \frac{1}{n!} \sum_{i_1 i_2, ... i_n} \frac{\partial^n V}{\partial q_{i_1} \partial q_{i_2} ... \partial q_{i_n}} \langle \Phi_{\mathbf{r}} | q_{i_1} q_{i_2} ... q_{i_n} | \Phi_{\mathbf{s}} \rangle \\ &= V(\mathbf{0}) \delta_{\mathbf{rs}} + \sum_{n=1}^{N_t} \frac{1}{n!} \sum_{i_1 i_2, ... i_n} \frac{\partial^n V}{\partial q_{i_1} \partial q_{i_2} ... \partial q_{i_n}} \prod_{m=1}^{M} \langle \phi_{r^m}^m(q_m) | q_m^{\sum_{j=1}^{n} \delta_{i_j m}} | \phi_{s^m}^m(q_m) \rangle \end{aligned} \tag{6.6}$$

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.

$$\hat{H} = \sum_{p=1}^{P} c_p \prod_{m=1}^{M} \hat{h}_m^p \tag{6.7}$$

giving a similar simple structure of the matrix elements.

$$\langle \Phi_{\mathbf{r}} | \hat{H} | \Phi_{\mathbf{s}} \rangle = \sum_{p=1}^{P} c_p \prod_{m=1}^{M} \langle \phi_{r^m}^m(q_m) | \hat{h}_m^p | \phi_{s^m}^m(q_m) \rangle \tag{6.8}$$

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. 6.7 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.6.8 is crucial for efficient wave function calculations.

### 6.3.2 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

$$V^{m_1} = V(0, ..., 0, q_{m_1}, 0, ..., 0) \tag{6.9}$$

$$V^{m_1 m_2} = V(0, ..., 0, q_{m_1}, 0, ..., 0, q_{m_2}, 0, ..., 0) \tag{6.10}$$

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

$$V^{m_1 m_2, ... m_M} = V(\mathbf{q}) \tag{6.11}$$

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, $\mathbf{m} = \{m_0, m_1, \ldots\}$ 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

$$V^{m_1 m_2}(q_{m_1}, q_{m_2}) = V^{m_2 m_1}(q_{m_2}, q_{m_1}) \tag{6.12}$$

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

$$\sum_{m_1<m_2=1}^{M} V^{m_1 m_2}(q_{m_1}, q_{m_2}=0) = \sum_{m_1} V^{m_1}(M-m_1) \tag{6.13}$$

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

$$\sum_{\mathbf{m}_n}{}'' = \sum_{m_1}\sum_{m_2>m_1} .... \sum_{m_n>m_{n-1}} \tag{6.14}$$

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

$$\sum_{\mathbf{m}_n}{}' = \sum_{m_1}\sum_{m_2\neq m_1} .... \sum_{m_n\neq\{m_{n-1},m_{n-2},...,m_1\}} \tag{6.15}$$

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

$$\bar{V}^{\mathbf{m}_n}(..., q_i=0, ....) = 0 \tag{6.16}$$

For example,

$$\bar{V}^{m_1}(0) = 0 \tag{6.17}$$

$$\bar{V}^{m_1 m_2}(0, q_{m_2}) = \bar{V}^{m_1 m_2}(q_{m_1}, 0) = 0 \tag{6.18}$$

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

$$V^{\mathbf{m}_n} = \bar{V}^{\mathbf{m}_n} + \sum_{n'=1}^{n-1}\sum_{\mathbf{m}_{n'}} C^{\mathbf{m}_{n'}} \bar{V}^{\mathbf{m}_{n'}} \tag{6.19}$$

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.(6.16). We obtain in this way

$$V^{m_1} = \bar{V}^{m_1} \tag{6.20}$$

$$V^{m_1 m_2} = \bar{V}^{m_1 m_2} + \bar{V}^{m_1} + \bar{V}^{m_2} = S^{m_1 m_2}(\bar{V}^{m_1 m_2} + 2\bar{V}^{m_1}) \tag{6.21}$$

$$\begin{aligned} V^{m_1 m_2 m_3} &= \bar{V}^{m_1 m_2 m_3} + \bar{V}^{m_1 m_2} + \bar{V}^{m_1 m_3} + \bar{V}^{m_2 m_3} + \bar{V}^{m_1} + \bar{V}^{m_2} + \bar{V}^{m_3} \\ &= S^{m_1 m_2 m_3}(\bar{V}^{m_1 m_2 m_3} + 3\bar{V}^{m_1 m_2} + 3\bar{V}^{m_1}) \end{aligned} \tag{6.22}$$

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

$$S^{m_1 m_2} F(m_1, m_2) = \frac{1}{2}(F(m_1, m_2) + F(m_2, m_1)) \tag{6.23}$$

In general

$$V^{\mathbf{m}_n} = S^{\mathbf{m}_n} \sum_{n'=1}^{n} \binom{n}{n'} \bar{V}^{\mathbf{m}_{n'}} \tag{6.24}$$

Eq.(6.24) 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 $n' = 1, 2, .., n$. 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 $\left( \begin{array}{c} n \\ n' \end{array} \right)$ ways. The operator $S^{\mathbf{m}_n}$ generates the different permutations of indices necessary. The examples above exemplify the formulae.

One can prove that

$$\bar{V}^{\mathbf{m}_n} = S^{\mathbf{m}_n} \sum_{n'=1}^{n} (-1)^{n-n'} \left( \begin{array}{c} n \\ n' \end{array} \right) V^{\mathbf{m}_{n'}} \tag{6.25}$$

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

$$\Delta V^{(n)} = \sum_{\mathbf{m}_n}{}'' \bar{V}^{\mathbf{m}_n} \tag{6.26}$$

We have for example

$$\Delta V^{(1)} = \sum_{m_1} \bar{V}^{m_1} \tag{6.27}$$

$$\Delta V^{(2)} = \sum_{m_1} \sum_{m_2 > m_1} \bar{V}^{m_1 m_2} \tag{6.28}$$

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.(6.9-6.11) 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

$$V^{(k)} = \sum_{n=1}^{k} \Delta V^{(n)} \tag{6.29}$$

In this manner we have defined the sequence of potentials

$$V^{(1)}, V^{(2)}, V^{(3)}, ...., V^{(M)} \tag{6.30}$$

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 $\Delta V^{(k)}$

$$\Delta V^{(1)} = V^{(1)} \tag{6.31}$$

$$\Delta V^{(k)} = V^{(k)} - V^{(k-1)} \text{ for } k > 1 \tag{6.32}$$

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.(6.26). 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,

$$V = \sum_{\mathbf{m} \in \text{MCR}[V]} \bar{V}^{\mathbf{m}} \tag{6.33}$$

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,

$$\hat{H} = \sum_{t=1}^{T} c_t \prod_{m=1}^{M} \hat{h}^{m,t} \tag{6.34}$$

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:

$$\bar{V}^{\mathbf{m}_n} = \sum_{i_1,i_2,\ldots,i_n} c_{i_1,i_2,\ldots,i_n} y_{m_1}^{i_1} \cdots y_{m_n}^{i_n} \tag{6.35}$$

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

$$H = \sum_{n} \sum_{m_0<m_1<\ldots<m_{n-1}} \sum_{o_1,o_2,\ldots,o_{n-1}} c_{m_0 m_1 \cdots m_{n-1}}^{o_0 o_1 \cdots o_{n-1}} \; h^{m_0 o_0} \; h^{m_1 o_1} \; \cdots \; h^{m_{n-1} o_{n-1}}, \tag{6.36}$$

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 $q_m^i$.

### 6.3.3 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 $g^M = \exp(Mln(g))$ 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

$$N_{points} = \sum_{k=1}^{n} \left( \begin{array}{c} M \\ k \end{array} \right) g_k^k \tag{6.37}$$

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.

### 6.3.4 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.

### 6.3.5 Taylor expansion

Following section 6.3.1 `MidasCpp` can employ arbitrary order Taylor expansions to compute potential energy and property surfaces as described in Ref. [13]

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

### 6.3.6 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.[14] and for use of energy derivatives, in Ref.[15].

**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

### 6.3.7 Double incremental expansion

The double incremental expansion exploits ideas of restricting mode–mode couplings (as described in Sec. 6.3.2) 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. 4.5), it significantly reduces the overall computational cost giving rise to the so-called double incremental expansion in Falcon coordinates (DIF) approximation [16]. 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 [16].

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

$$E = E(z_1, z_2, ..., z_N), \tag{6.38}$$

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. 6.3.2, we can define a sequence of functions

$$E_{f_1} = E(0, ..., 0, z_{f_1}, 0, ..., 0), \tag{6.39}$$

$$E_{f_1 f_2} = E(0, ..., 0, z_{f_1}, 0, ..., 0, z_{f_2}, 0, ..., 0), \tag{6.40}$$

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. (6.44), the expression for the total supersystem energy $E$ can be written as

$$E = \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \underline{E}_{\mathbf{f}_l}, \tag{6.41}$$

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

$$V(\mathbf{q}) = E(\mathbf{z}(\mathbf{q})) - E(\mathbf{r}_0), \tag{6.42}$$

where $E(\mathbf{r}_0)$ is the energy of the reference structure. By expanding both terms at the right-hand side of Eq. (6.42), we obtain

$$V(\mathbf{q}) = \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \underline{E}_{\mathbf{f}_l} - \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \underline{E}_{\mathbf{f}_l}(\mathbf{r}_{0,\mathbf{f}_l}) = \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \Delta\underline{E}_{\mathbf{f}_l}. \tag{6.43}$$

Here, $\mathbf{r}_{0,\mathbf{f}_l}$ is the reference conformation of FCs $\mathbf{f}_l$. Finally, expanding in a similar fashion a subpotential $V^{\mathbf{m}_{n'}}$ from Eq (6.25) and inserting the resulting expression into Eq. (6.44), one obtaines the doubly-incremented form of the total potential,

$$V = S^{\mathbf{m}_n} \sum_{n'=1}^{n} (-1)^{n-n'} \begin{pmatrix} n \\ n' \end{pmatrix} \sum_{\mathbf{f}_l \in \mathrm{FCR}[E]} \Delta \underline{E}^{\mathbf{m}_{n'}}_{\mathbf{f}_l}. \tag{6.44}$$

**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

### 6.3.8 The Adaptive density guided approach

The Adaptive density guided approach (ADGA) is an iterative technique to building the PES and property surfaces. The advantage of this approach is that much of the 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[17]. Extensions allows for multilevel approaches where electronic structure information at different levels are used at different levels of mode combination[18]. The most recent update to ADGA allows any fit basis to be used and various approaches to finding the grid boundaries, see Ref. [19].

Tables 6.1–6.6 show maximum deviation (MaxD), minimum deviation (MinD), and root-mean-square deviation (RMSD) of fundamental frequencies (as compared to the calculation in the last column of each table) as well as the number of single points (No. SPs) calculated for a range of different molecules and ADGA thresholds. All computations have been done using the Orca program package and the HF-3c electronic structure method. This data has originally been presented in the Supporting Information of Ref. [20]. The columns marked in blue denotes a calculation applying the default thresholds used by `MidasCpp` for each molecule. To change the thresholds when running a calculation use keywords #2 ItVDensThr ($\epsilon_{\mathrm{rel}}$), #2 ItResEnThr ($\epsilon_{\mathrm{abs}}$), and #2 ItResDensThr ($\epsilon_\rho$).

Table 6.1: ADGA-2M computations for water. For more details, see the main text. All deviations are given in $\mathrm{cm}^{-1}$.

| $\log_{10}\epsilon_{\mathrm{rel}}$ | −2 | −2 | −3 | −2 | −2 | −3 | −2 | −3 | −2 | −3 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\log_{10}\epsilon_{\mathrm{abs}}$ | −3 | −3 | −3 | −4 | −4 | −4 | −5 | −5 | −6 | −6 |
| $\log_{10}\epsilon_\rho$ | −3 | −4 | −4 | −3 | −4 | −4 | −3 | −3 | −3 | −4 |
| MaxD | −3.59 | −3.96 | −3.96 | −0.00 | 0.06 | −0.07 | 0.02 | 0.03 | −0.01 | – |
| MinD | −7.25 | −9.65 | −9.65 | −0.50 | −0.86 | −0.42 | −0.10 | −0.31 | −0.10 | – |
| RMSD | 5.62 | 6.56 | 6.56 | 0.34 | 0.50 | 0.26 | 0.07 | 0.22 | 0.07 | – |
| No. SPs | 68 | 69 | 75 | 237 | 280 | 404 | 536 | 841 | 751 | 2086 |

Table 6.2: ADGA-2M computations for formaldehyde. For more details, see the main text. All deviations are given in $cm^{-1}$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\log_{10} \epsilon_{rel}$ | −2 | −2 | −3 | −2 | −2 | −3 | −2 | −3 | −2 | −3 |
| $\log_{10} \epsilon_{abs}$ | −3 | −3 | −3 | −4 | −4 | −4 | −5 | −5 | −6 | −6 |
| $\log_{10} \epsilon_{\rho}$ | −3 | −4 | −4 | −3 | −4 | −4 | −3 | −3 | −3 | −4 |
| MaxD | 9.16 | 8.39 | 8.39 | 4.85 | 6.01 | 6.02 | 0.06 | 0.07 | 0.04 | – |
| MinD | −17.25 | −19.61 | −19.59 | −0.60 | −0.07 | −0.12 | −0.40 | −0.62 | −0.45 | – |
| RMSD | 10.40 | 11.11 | 11.10 | 2.95 | 4.28 | 4.29 | 0.22 | 0.33 | 0.25 | – |
| No. SPs | 134 | 134 | 145 | 459 | 457 | 596 | 1812 | 2766 | 2741 | 6511 |

Table 6.3: ADGA-2M computations for ethylene. For more details, see the main text. All deviations are given in $cm^{-1}$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\log_{10} \epsilon_{rel}$ | −2 | −2 | −3 | −2 | −2 | −3 | −2 | −3 | −2 | −3 |
| $\log_{10} \epsilon_{abs}$ | −3 | −3 | −3 | −4 | −4 | −4 | −5 | −5 | −6 | −6 |
| $\log_{10} \epsilon_{\rho}$ | −3 | −4 | −4 | −3 | −4 | −4 | −3 | −3 | −3 | −4 |
| MaxD | 23.35 | 25.57 | 25.57 | 19.49 | 25.87 | 25.87 | 0.87 | 0.87 | 0.36 | – |
| MinD | −19.51 | −21.88 | −21.89 | −1.60 | 0.34 | 0.36 | −0.18 | −0.16 | −0.20 | – |
| RMSD | 16.43 | 18.96 | 18.96 | 10.04 | 14.09 | 14.09 | 0.37 | 0.37 | 0.14 | – |
| No. SPs | 415 | 414 | 436 | 1346 | 1292 | 1681 | 4145 | 9616 | 6306 | 17501 |

Table 6.4: ADGA-2M computations for imidazole. For more details, see the main text. All deviations are given in $cm^{-1}$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\log_{10} \epsilon_{rel}$ | −2 | −2 | −3 | −2 | −2 | −3 | −2 | −3 | −2 | −3 |
| $\log_{10} \epsilon_{abs}$ | −3 | −3 | −3 | −4 | −4 | −4 | −5 | −5 | −6 | −6 |
| $\log_{10} \epsilon_{\rho}$ | −3 | −4 | −4 | −3 | −4 | −4 | −3 | −3 | −3 | −4 |
| MaxD | 98.77 | 46.11 | 46.11 | 33.07 | 46.19 | 46.17 | 1.22 | 1.22 | 0.15 | – |
| MinD | −41.85 | −140.22 | −140.23 | 1.12 | 0.60 | 0.55 | −1.73 | −1.73 | −0.44 | – |
| RMSD | 47.91 | 47.65 | 47.65 | 14.97 | 18.82 | 18.83 | 0.67 | 0.67 | 0.13 | – |
| No. SPs | 1106 | 1105 | 1125 | 3324 | 3319 | 3480 | 17105 | 22609 | 39297 | 85446 |

Table 6.5: ADGA-2M computations for pyrimidine. For more details, see the main text. All deviations are given in $cm^{-1}$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\log_{10} \epsilon_{rel}$ | −2 | −2 | −3 | −2 | −2 | −3 | −2 | −3 | −2 | −3 |
| $\log_{10} \epsilon_{abs}$ | −3 | −3 | −3 | −4 | −4 | −4 | −5 | −5 | −6 | −6 |
| $\log_{10} \epsilon_{\rho}$ | −3 | −4 | −4 | −3 | −4 | −4 | −3 | −3 | −3 | −4 |
| MaxD | 97.25 | 72.50 | 72.52 | 16.42 | 21.44 | 21.42 | 1.54 | 1.55 | 0.16 | – |
| MinD | −124.60 | −105.81 | −105.81 | −3.34 | −5.13 | −5.13 | −1.50 | −1.52 | −0.29 | – |
| RMSD | 54.89 | 49.12 | 49.14 | 8.09 | 10.31 | 10.32 | 0.80 | 0.80 | 0.09 | – |
| No. SPs | 1352 | 1355 | 1385 | 2939 | 3045 | 3323 | 16022 | 20756 | 33836 | 75993 |

**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* **2018**, *148*, 064113

Table 6.6: ADGA-2M computations for uracil. For more details, see the main text. All deviations are given in $cm^{-1}$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\log_{10} \epsilon_{\text{rel}}$ | $-2$ | $-2$ | $-3$ | $-2$ | $-2$ | $-3$ | $-2$ | $-3$ | $-2$ | $-3$ |
| $\log_{10} \epsilon_{\text{abs}}$ | $-3$ | $-3$ | $-3$ | $-4$ | $-4$ | $-4$ | $-5$ | $-5$ | $-6$ | $-6$ |
| $\log_{10} \epsilon_{\rho}$ | $-3$ | $-4$ | $-4$ | $-3$ | $-4$ | $-4$ | $-3$ | $-3$ | $-3$ | $-4$ |
| MaxD | – | – | – | 24.03 | 34.49 | 34.50 | 3.90 | 4.26 | 0.14 | – |
| MinD | – | – | – | $-9.32$ | $-7.03$ | $-7.06$ | $-1.34$ | $-1.34$ | $-0.26$ | – |
| RMSD | – | – | – | 10.45 | 14.83 | 14.88 | 1.43 | 1.47 | 0.07 | – |
| No. SPs | – | – | – | 4507 | 4721 | 4897 | 25377 | 32107 | 62117 | 147444 |

### 6.3.9 The Time-Dependent Adaptive density guided approach

The Time-Dependent Adaptive density guided approach (TD-ADGA) is an extension of the ADGA to construct PES's for simulations of time-dependent wave function methods. The TD-ADGA is based on the basic ADGA framework but is run together with a simulation of a time-dependent wave function. The TD-ADGA utilizes the one-mode density og the time-dependent wave function to guide the placement of new single points. During a TD-ADGA calculation the conververgence og the ADGA grid is thus repeatedly re-obtained by adding new single points as the time-dependent wave function moves around on the PES during the dynamics simulation. The final operator file(s) (.mop) produced by the TD-ADGA is (are) tailored for the dynamics of the studied system. [21]

The TD-ADGA is implemented to be used with the Time-Dependent Hartree, Multiconfiguration Time-Dependent Hartree, and the Time-Dependent Vibrational Coupled Cluster (with and without time-dependent modals) wave function methods which are also available in `MidasCpp`.

**Consider to cite:**

1. N. M. Høyer, O. Christiansen, "Quasi-direct Quantum Molecular Dynamics: The Time-Dependent Adaptive Density-Guided Approach for Potential Energy Surface Construction", *J. of Chem. Theory Comput.* **2024**, *20*, 558–579

# Chapter 7

# 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 apply to time-dependent theory.

## 7.1 Introduction, notation, and the scaling problem

The number of modes (or degrees of freedom) is denoted by $M$. We shall usually use $m, m', m''$ as indices for the modes. We use $\phi^m_{s^m}(q_m)$ 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

$$\{\phi^m_{s^m}(q_m), s^m = 1, N_m\} \tag{7.1}$$

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

$$\langle \phi^m_{r^m}(q_m) | \phi^m_{s^m}(q_m) \rangle = \delta_{r^m s^m} \tag{7.2}$$

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

$$\Phi_{\mathbf{s}}(\mathbf{q}) = \prod_{m=1}^{M} \phi^m_{s^m}(q_m) = \phi^1_{s^1}(q_1)\phi^2_{s^2}(q_2)....\phi^M_{s^M}(q_M) \tag{7.3}$$

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

$$\Psi_{\mathbf{c}}(\mathbf{q}) = \sum_{\mathbf{s}} c_{\mathbf{s}} \Phi_{\mathbf{s}} = \sum_{s^1}^{N^1} \sum_{s^2}^{N^2} ... \sum_{s^M}^{N^M} c_{s^1 s^2 ... s^M} \Phi_{s^1 s^2 ... s^M}(q_1, q_2, ..., q_M) \tag{7.4}$$

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.(7.4).

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 simplificty 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 $\phi_i$ 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.

## 7.2 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

$$\hat{H} = \hat{H}_1 + \hat{H}_r \tag{7.5}$$

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

$$\hat{H}_1 = \sum_{m=1}^{M} \hat{h}^m \tag{7.6}$$

$\hat{H}_r$ 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 $\phi^m_{s^m}(q_m)$ that are eigen-functions for the one-mode operators in $\hat{H}_1$

$$\hat{h}^m \phi^m_{s^m}(q_m) = \epsilon^m_{s^m} \phi^m_{s^m}(q_m) \tag{7.7}$$

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

$$\hat{H}_1 \Phi_{\mathbf{s}} = \sum_{m=1}^{M} \hat{h}^m \Phi_{\mathbf{s}} = (\sum_{m=1}^{M} \epsilon^m_{s^m}) \Phi_{\mathbf{s}} = E_{1,\mathbf{s}} \Phi_{\mathbf{s}} \tag{7.8}$$

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

$$E_{\mathbf{i}} = \langle \Phi_{\mathbf{i}} | \hat{H} | \Phi_{\mathbf{i}} \rangle \tag{7.9}$$

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

$$\langle \phi^m_{i^m}(q_m) | \phi^m_{i^m}(q_m) \rangle = 1 \tag{7.10}$$

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

$$\begin{aligned} \delta^m E_{\mathbf{i}} &= 2Re\{\langle \delta\phi^m_{i^m}(q_m) \prod_{m'=1,\neq m}^{M} \phi^{m'}_{i^{m'}}(q_{m'}) | \hat{H} | \prod_{m''=1}^{M} \phi^{m''}_{i^{m''}}(q_{m''}) \rangle\} \\ &= 2Re\{\langle \delta\phi^m_{i^m}(q_m) | \hat{F}^{m,\mathbf{i}} | \phi^m_{i^m}(q_m) \rangle\} \end{aligned} \tag{7.11}$$

where we have defined the effective operator

$$\hat{F}^{m,\mathbf{i}} = \langle \prod_{m'=1,\neq m}^{M} \phi^{m'}_{i^{m'}}(q_{m'}) | \hat{H} | \prod_{m''=1,\neq m}^{M} \phi^{m''}_{i^{m''}}(q_{m''}) \rangle \tag{7.12}$$

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

$$\delta\{\langle \phi^m_{i^m}(q_m) | \phi^m_{i^m}(q_m) \rangle\} = 0 \tag{7.13}$$

or

$$Re\{\langle \delta\phi^m_{i^m}(q_m) | \phi^m_{i^m}(q_m) \rangle\} = 0 \tag{7.14}$$

We may combine these constraints with Eq.(7.11) and write the variational criteria as

$$0 = Re\{\langle \delta\phi^m_{i^m}(q_m) | \hat{F}^{m,\mathbf{i}} | \phi^m_{i^m}(q_m) \rangle\} - \epsilon^m_{i^m} Re\{\langle \delta\phi^m_{i^m}(q_m) | \phi^m_{i^m}(q_m) \rangle\} \tag{7.15}$$

where $\epsilon^m_{i^m}$ is a Lagrangian multiplier. We see that $\epsilon^{m*}_{i^m} = \epsilon^m_{i^m}$ and thus $\epsilon^m_{i^m}$ is real. We may therefore write Eq.(7.15) as

$$0 = Re\{\langle \delta\phi^m_{i^m}(q_m) | \hat{F}^{m,\mathbf{i}} - \epsilon^m_{i^m} | \phi^m_{i^m}(q_m) \rangle\} \tag{7.16}$$

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

$$0 = \langle \delta\phi^m_{i^m}(q_m) | \hat{F}^{m,\mathbf{i}} - \epsilon^m_{i^m} | \phi^m_{i^m}(q_m) \rangle \tag{7.17}$$

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

$$\hat{F}^{m,\mathbf{i}}\phi^m_{i^m}(q_m) = \epsilon^m_{i^m}\phi^m_{i^m}(q_m) \tag{7.18}$$

The total energy is obtained from Eq.(7.9)

$$E_{\mathbf{i}} = \langle\Phi_{\mathbf{i}}|H|\Phi_{\mathbf{i}}\rangle \tag{7.19}$$

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

$$\epsilon^m_{i^m} = \langle\phi^m_{i^m}(q_m)|\hat{F}^{m,\mathbf{i}}|\phi^m_{i^m}(q_m)\rangle = \langle\Phi_{\mathbf{i}}|\hat{H}|\Phi_{\mathbf{i}}\rangle = E_{\mathbf{i}} \tag{7.20}$$

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.

## 7.3 Basis set expansion for VSCF

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

$$\phi^m_{s^m}(q_m) = \sum_{v^m} C^m_{v^m s^m}\chi^m_{v^m}(q_m) \tag{7.21}$$

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

$$S^m_{v^m u^m} = \langle\chi^m_{v^m}(q_m)|\chi^m_{u^m}(q_m)\rangle \tag{7.22}$$

We shall here require that the primitive basis is orthonormal

$$S^m_{v^m u^m} = \delta_{v^m u^m} \tag{7.23}$$

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

Expressed in matrix form in the primitive basis set the VSCF modal eigenvalue equations in Eq.(7.18) becomes

$$\mathbf{F}^{m,\mathbf{i}}\mathbf{C}^m = \mathbf{C}^m\epsilon^{\mathbf{m}} \tag{7.24}$$

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

$$F^{m,\mathbf{i}}_{u^m v^m} = \langle\chi^m_{u^m}(q_m)|\hat{F}^{m,\mathbf{i}}|\chi^m_{v^m}(q_m)\rangle \tag{7.25}$$

and $\epsilon^{\mathbf{m}}$ is a diagonal matrix with the $\mathbf{F}^{m,\mathbf{i}}$ eigenvalues $\epsilon^m_{r^m}$ 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 $\epsilon^m_{r^m}$ 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.(7.25) 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.

## 7.4 Correlated Vibrational Wave functions : basic aspects

### 7.4.1 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.

### 7.4.2 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

$$|\mathrm{VCI}\rangle = C_{\mathbf{i}} |\Phi_{\mathbf{i}}\rangle + \sum_{\mu} C_{\mu} \tau_{\mu} |\Phi_{\mathbf{i}}\rangle \tag{7.26}$$

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

$$\tau_\mu \left|\Phi_{\mathbf{i}}\right\rangle = \left|\mu\right\rangle \tag{7.27}$$

Thus, $\mu$ 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 $\mu$, 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 7.1.

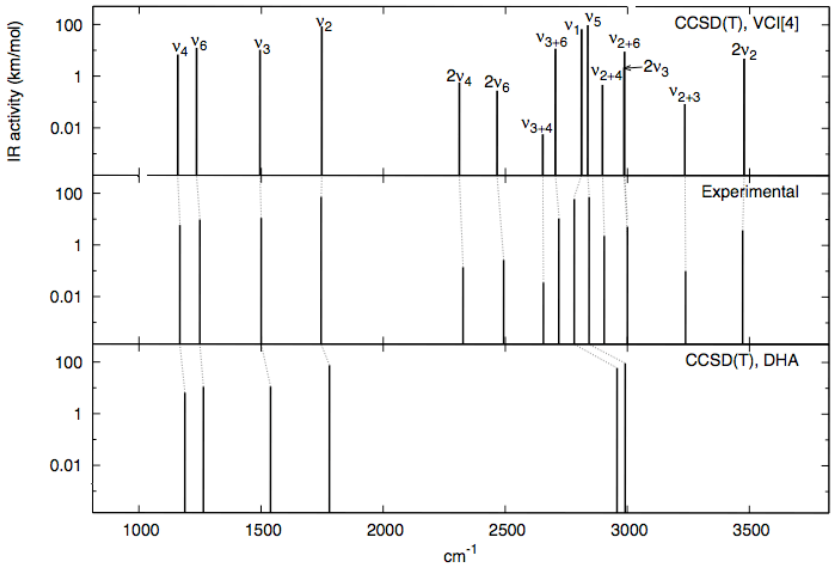



Figure 7.1: Comparison of formaldehyde vibrational excitation energies in experiment, the double harmonic approximation, and VCI[4] using a relative accurate potential) (Seidler and Christiansen, 2007).

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

$$\mathbf{HC} = E\mathbf{C} \tag{7.28}$$

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

$$H_{\mu\nu} = \langle \mu | \hat{H} | \nu \rangle \tag{7.29}$$

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.

### 7.4.3 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

$$\hat{H} = \hat{H}_0 + \hat{U} = \hat{F}^{\mathbf{i}} + \hat{U}^{\mathbf{i}} \tag{7.30}$$

$$\hat{F}^{\mathbf{i}} = \sum_m \hat{F}^{m,\mathbf{i}} \tag{7.31}$$

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 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 $\mathbf{i}$ 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

$$E_{\mathbf{i}} = \sum_{n=0}^{\infty} E_{\mathbf{i}}^{(n)} \tag{7.32}$$

$$|\Psi_{\mathbf{i}}\rangle = \sum_{n=0}^{\infty} |\Psi^{(n)_{\mathbf{i}}}\rangle \tag{7.33}$$

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.7.2. 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.

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.[22]

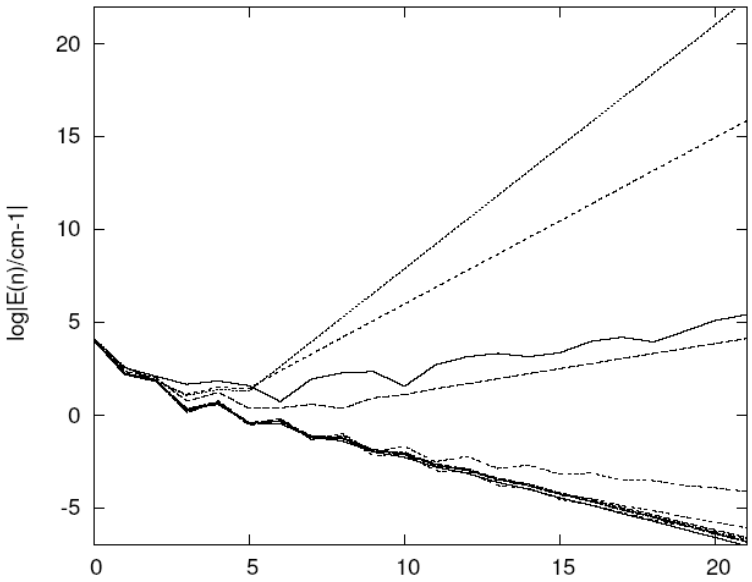



Figure 7.2: Contributions in different orders of perturbation theory in VMP for the fundamental vibrations of ethylene.

### 7.4.4 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.

$$|\mathrm{VCC}\rangle = \exp(T)|\Phi_\mathbf{i}\rangle. \tag{7.34}$$

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

$$T = \sum_\mu t_\mu \tau_\mu \tag{7.35}$$

where $t_\mu$ are the VCC cluster amplitudes while $\tau_\mu$ are the previously described excitation operators indexed by $\mu$. 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 $\mu$ 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,

$$\exp(-T)\hat{H}\exp(T)|\Phi_\mathbf{i}\rangle = E_{VCC}|\Phi_\mathbf{i}\rangle. \tag{7.36}$$

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

$$E_{VCC} = \langle\Phi_\mathbf{i}|\exp(-T)\hat{H}\exp(T)|\Phi_\mathbf{i}\rangle = \langle\Phi_\mathbf{i}|\hat{H}\exp(T)|\Phi_\mathbf{i}\rangle. \tag{7.37}$$

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

$$0 = e_\mu = \langle\mu|\exp(-T)\hat{H}\exp(T)|\Phi_\mathbf{i}\rangle. \tag{7.38}$$

These equations determines the parameters also for approximate VCC schemes where only a restricted set of excitations are included in the cluster operator Eq.(7.35) and the same set of excitations are considered in Eq.(7.38).

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 7.3 illustrate this.

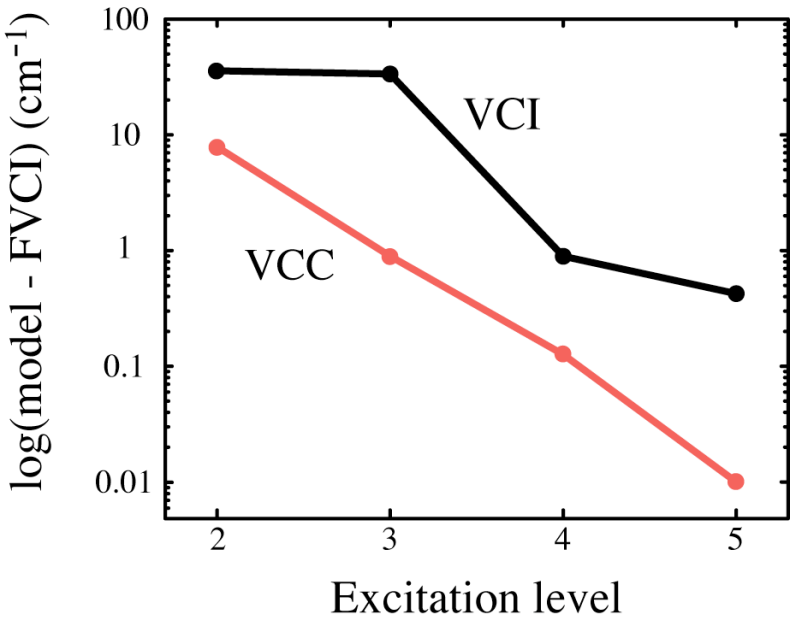



Figure 7.3: Comparison of vibrational excitation energies between VCI[n] and VCC[n]. Errors relative to FVCI in $cm^{-1}$ for the eight lowest fundamental vibrations of ethylene for different truncation of the excitation space determined by n. (Seidler and Christiansen).

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 8

# Many-mode wave-functions in MidasCpp

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

## 8.1 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.[23] and Gerber et al[24] since later 1970s.

The VSCF implementation of MidasCpp is described in Ref.[25] and stand out as based on second quantization[26], 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

## 8.2 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 [27, 28]).

The arbitrary order VMP theory and implementation (and Pade approximants) of MidasCpp is described in Ref.[22]. An alternative to VMP is vibrational auto adjusting perturbation theory[29] 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

## 8.3 Vibrational Coupled Cluster

### 8.3.1 Ground-state Vibrational Coupled Cluster

Vibrational Coupled Cluster theory was introduced in Refs.[26, 30]

An efficient VCC[2] algorithm specialized to Hamiltonians with at most two-mode couplings showed the perspective in applications to large systems[31, 32].

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.[33]. In Ref.[33] 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.[34].

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.[35].

The ground state equations are by standard solved by iterative algorithms as described by [36] 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", *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

### 8.3.2 VCC response theory

Vibrational response theory[37] 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.[38]

The general contraction algorithms of Ref.[33] was extended to general VCC response theory in Ref.[39] 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.[33, 38, 39] using for example standard Davidson or Olsen types of algorithms with some means for targeting specific states.[40]

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 [41, 42] using respectively Lanczos and band-Lanczos algorithms. Alternatively one solves complex linear-response equations to achieve the same goal as described in Ref.[43]

**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", *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", *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", *The Journal of Chemical Physics* **2015**, *143*, 134108

## 8.4 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.[33].

Special to MidasCpp is also the possibility to compute VCI spectra and vibrational properties from vibrational response theory[37] applied to VCI states[44–46]. Vibrational spectra can be computed directly from the VCI damped response functions as described in Refs.[42, 47] using respectively Lanczos and band-Lanczos algorithms. Alternatively one solves complex linear-response equations to achieve the same goal as described in Ref.[43].

**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", *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", *The Journal of Chemical Physics* **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

## 8.5 Tensor-Decomposition Methods

In the context of VCC and VCI tensor-decomposition approaches has been studied for possible gains in VCC and VCI computations [36, 48–50].

At this moment CP-VCC is available for ground state VCC computations as described in Refs.[36, 50]. 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* **2018**, *148*, 024103

# Chapter 9

# Time-dependent wave-function propagation in MidasCpp

The MidasCpp program includes an array of methods for solving the time-dependent Schrödinger equation starting from a given initial state. Among these are the *time-dependent Hartree* (TDH), *multiconfiguration time-dependent Hartree* (MCTDH), and *time-dependent vibrational coupled-cluster* (TDVCC) methods.

## 9.1 Time-dependent Hartree (TDH)

The TDH wave function is a single Hartree product of time-dependent modals written in phase-isolated form as,

$$|\bar{\Phi}\rangle = e^{-iF(t)}\,|\tilde{\Phi}\rangle = e^{-iF(t)}\prod_{m=1}^{M}\tilde{a}_{i^m}^{m\,\dagger}(t)\,|\mathrm{vac}\rangle\ , \tag{9.1}$$

where $F(t)$ is a time-dependent overall phase factor and $\tilde{a}_{i^m}^{m\,\dagger}(t)$ creates the time-dependent modal in mode $m$.

The TDH method can be applied to very large systems, but because it only accounts for correlation between modes in a mean-field manner, its accuracy is limited [51, 52].

MidasCpp includes two variants of TDH with two different parameterizations of the time-dependent modals [53]. The L-TDH method employs a linear parameterization of the form $\tilde{a}_{i^m}^{m\,\dagger}(t) = \sum_{r^m} a_{r^m}^{m\,\dagger} C_{r^m i^m}^{m}(t)$ while the X-TDH method writes the time-dependent modals using an exponential transformation $\tilde{a}_{i^m}^{m\,\dagger}(t) = e^{\kappa(t)} a_{i^m}^{m\,\dagger} e^{-\kappa(t)}$ where $\kappa(t)$ is an anti-hermitian operator. The X-TDH formulation leads to more complicated equations of motion, but is computationally much cheaper for systems with many terms in the Hamiltonian (as is often the case in accurate potential-energy surfaces).

**Consider to cite:**

1. N. K. Madsen, M. B. Hansen, A. Zoccante, K. Monrad, M. B. Hansen, O. Christiansen, "Exponential parameterization of wave functions for quantum dynamics: Time-dependent Hartree in second quantization", *J. Chem. Phys.* **2018**, *149*, 134110

## 9.2 Multiconfiguration Time-dependent Hartree (MCTDH)

The multiconfiguration time-dependent Hartree method is a flexible and highly efficient method for propagating wave packets [52, 54]. It expands the wave function in a compact time-dependent basis

as,

$$|\Psi(t)\rangle = \sum_{u^1=1}^{n^1} \cdots \sum_{u^M=1}^{n^M} C_{u^1 \ldots u^M}(t) \prod_{m=1}^{M} \tilde{a}^{m\,\dagger}_{u^m}(t) \, |\mathrm{vac}\rangle = \sum_{\mathbf{u}} C_{\mathbf{u}} \, |\tilde{\Phi}_{\mathbf{u}}\rangle \ . \tag{9.2}$$

It contains TDH and the exact propagation method as limits in the cases of $n^m = 1 \, \forall \, m$ and $n^m = N^m \, \forall \, m$, respectively.

In order to reduce the computational scaling of MCTDH (which is exponential with respect to $M$), MidasCpp includes an array of truncated MCTDH methods, the MCTDH[$n$] hierarchy[55]. The MCTDH[$n$] wave function uses a time-dependent basis (like MCTDH) and a wave function expansion based on excitations from a reference:

$$|\Psi(t)\rangle = (C_0 + C_1 + C_2 + \cdots) \, |\Phi\rangle \tag{9.3}$$

The multi-reference extension (denoted MR–MCTDH[$n$][56]) includes additional higher-order excitations involving a specific mode (or a small set of modes). Comparing MCTDH[2] and MR–MCTDH[2], we could write

$$|\Psi_{\text{MCTDH[2]}}(t)\rangle = (C_0 + C_1 + C_2) \, |\Phi\rangle \, , \tag{9.4}$$

$$|\Psi_{\text{MR–MCTDH[2]}}(t)\rangle = (C_0 + C_1 + C_2 + C_3') \, |\Phi\rangle \, , \tag{9.5}$$

where $C_3'$ contains all triple excitations that involve some special mode $m$ (as an example).

The autocorrelation function can be written as

$$S(t) = \langle \Psi(0) | \Psi(t) \rangle = \langle \Psi(0) | \hat{S} | \Psi(t) \rangle \, , \tag{9.6}$$

where the overlap operator $\hat{S}$ is a product of one-mode overlap operators:

$$\hat{S} = \prod_{m=1}^{M} \hat{S}^m . \tag{9.7}$$

Each one-mode factor is given in terms of modal overlap integrals $S^m_{p^m q^m}$:

$$\hat{S}^m = \sum_{p^m q^m} S^m_{p^m q^m} E^m_{p^m q^m} \tag{9.8}$$

$$= S^m_{i^m i^m} E^m_{i^m i^m} + \sum_{a^m} S^m_{a^m i^m} E^m_{a^m i^m} + \sum_{a^m} S^m_{i^m a^m} E^m_{i^m a^m} + \sum_{a^m b^m} S^m_{a^m b^m} E^m_{a^m b^m} \tag{9.9}$$

$$= {}^{p}\hat{S}^m + {}^{u}\hat{S}^m + {}^{d}\hat{S}^m + {}^{f}\hat{S}^m \tag{9.10}$$

The matrix elements $S^m_{p^m q^m}$ are simply the modal overlap integrals. The (MR–)MCTDH[$n$] wave function is closed under deexcitation, so the passive ($p$), down ($d$) and forward ($f$) contributions can be absorbed into the ket exactly. The up ($u$) part of $\hat{S}^m$ will generally generate excitations that lie outside the excitation space of the wave function (${}^{u}\hat{S}^m$ will for example turn double excitations into triple excitations and so on). The autocorrelation function is thus computed by sequentially absorbing the factors $\hat{S}^m$ into the ket,

$$|\Psi\rangle \leftarrow \hat{S}^m \, |\Psi\rangle \, , \tag{9.11}$$

while discarding any out-of-space terms. Although this is an approximation, it is found to work well in practice.

**Consider to cite:**

1. N. K. Madsen, M. B. Hansen, G. A. Worth, O. Christiansen, "Systematic and variational truncation of the configuration space in the multiconfiguration time-dependent Hartree method: The MCTDH[n] hierarchy", *The Journal of Chemical Physics* **2020**, *152*, 084101

2. N. K. Madsen, M. B. Hansen, G. A. Worth, O. Christiansen, "MR–MCTDH[n]: Flexible Configuration Spaces and Nonadiabatic Dynamics within the MCTDH[n] Framework", *J. Chem. Theory Comput.* **2020**, *16*, 4087–4097

## 9.3 Time-dependent Vibrational Coupled-Cluster (TDVCC)

**Consider to cite:**

1. M. B. Hansen, N. K. Madsen, A. Zoccante, O. Christiansen, "Time-dependent vibrational coupled cluster theory: Theory and implementation at the two-mode coupling level", *The Journal of Chemical Physics* **2019**, *151*, 154116

2. N. K. Madsen, A. B. Jensen, M. B. Hansen, O. Christiansen, "A general implementation of time-dependent vibrational coupled-cluster theory", *The Journal of Chemical Physics* **2020**, *153*, 234109

3. N. K. Madsen, M. B. Hansen, O. Christiansen, A. Zoccante, "Time-dependent vibrational coupled cluster with variationally optimized time-dependent basis sets", *The Journal of Chemical Physics* **2020**, *153*, 174108

# Chapter 10

# 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

$$\phi^m_{s^m}(q_m) = \sum_{v^m} C^m_{v^m s^m} \chi^m_{v^m}(q_m) \tag{10.1}$$

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 $\phi^m_{r^m}(q_m)$ basis are simple to obtain from the integrals in the $\chi^m_{v^m}(q_m)$ basis as

$$h^{m,o^t}_{r^m s^m} = \sum_{u^m v^m} (C^m_{u^m s^m})^* h^{m,o^t}_{u^m v^m} C^m_{v^m s^m}. \tag{10.2}$$

Here the primitive one-mode integrals are

$$h^{m,o^t}_{u^m v^m} = \langle \chi^m_{u^m}(q_m) | h^{m,o^t} | \chi^m_{v^m}(q_m) \rangle. \tag{10.3}$$

Note, that even if the modal basis $\phi^m_{r^m}(q_m)$ often is chosen to be orthonormal, this choice does not depend on the primitive basis $\chi^m_{v^m}(q_m)$ 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.

## 10.1 B-spline basis functions

### 10.1.1 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 $\mathbf{t}$. 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 [60], as

$$B^1_i(x) = \begin{cases} 1 & \text{if } x \in [t_i, t_{i+1}) \\ 0 & \text{elsewhere} \end{cases} \tag{10.4}$$

$$B^k_i(x) = \frac{x - t_i}{t_{i+k-1} - t_i} B^{k-1}_i(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B^{k-1}_{i+1}(x). \tag{10.5}$$

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 $\mu_i$ 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.

### 10.1.2 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).

### 10.1.3 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[25] 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.

### 10.1.4 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

$$\rho = \frac{N}{b-a} \tag{10.6}$$

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. (10.6). 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 $\rho$. 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.

### 10.1.5 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 [60].

The MidasCpp B-spline vibrational basis function implementation is described in Ref. [61].

**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

## 10.2 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.

## 10.3 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 11

# Machine Learning Algorithms

## 11.1 Gaussian Process Regression

Gaussian process regression (GPR), sometimes also referred to as kriging [62–64], 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.[65] 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

$$p(\mathbf{v}|\mathbf{m}, \mathbf{K}) = \frac{1}{(2\pi)^{n/2} |\mathbf{K}|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{v} - \mathbf{m}) \mathbf{K}^{-1} (\mathbf{v} - \mathbf{m})\right) \tag{11.1}$$

for which we use the short hand notation

$$\mathbf{v} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \quad . \tag{11.2}$$

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

$$\begin{bmatrix} \mathbf{v} \\ \mathbf{v}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m} \\ \mathbf{m}_* \end{bmatrix}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right) \quad , \tag{11.3}$$

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)

$$\mathbf{v}_*|\mathbf{v} \sim \mathcal{N}(\mu, \mathbf{\Sigma}) \tag{11.4}$$

with new mean and co-variance function

$$\mu = \mathbf{m}_* + K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} (\mathbf{v} - \mathbf{m}) \tag{11.5}$$

$$\mathbf{\Sigma} = K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}_*) \tag{11.6}$$

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 $\mu$ 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 $\sigma^2$ on the diagonal of the co-variance matrix

$$\mathbf{v} \sim \mathcal{N}(\mathbf{m}, \mathbf{K} + \sigma^2 \cdot \mathbf{I}) \quad . \tag{11.7}$$

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 #<N> Noise keyword.

### 11.1.1 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

$$\mathbf{K}\omega = (\mathbf{v} - \mathbf{m}) \quad . \tag{11.8}$$

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

$$\mathrm{RMSD}_{\mathrm{geo}} = \sqrt{\frac{1}{N_{\mathrm{coord}}} \sum_{i}^{N_{\mathrm{coord}}} \left( x_i^{(m)} - x_i^{(n)} \right)^2} \tag{11.9}$$

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

Using GPR the potential is represented as

$$V(\mathbf{x}) = \sum_k \omega_k k(\mathbf{x}, \mathbf{x}_k) \quad , \tag{11.10}$$

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.

### 11.1.2 Kernel functions

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

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_i, \mathbf{x}_i) & k(\mathbf{x}_i, \mathbf{x}_j) & \cdots & k(\mathbf{x}_i, \mathbf{x}_n) \\ k(\mathbf{x}_j, \mathbf{x}_i) & k(\mathbf{x}_j, \mathbf{x}_j) & \cdots & k(\mathbf{x}_j, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_i) & k(\mathbf{x}_n, \mathbf{x}_j) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \tag{11.11}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left( \sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{l} \right)^{\nu} K_\nu \left( \sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{l} \right) \tag{11.12}$$

where $\Gamma$ is the gamma function and $K_\nu$ the modified Bessel function. This general expression is not implemented, but for half integer values of $\nu$ 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

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \tag{11.13}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \frac{\sqrt{3}\,\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \exp\left(-\frac{\sqrt{3}\,\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \tag{11.14}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \frac{\sqrt{5}\,\|\mathbf{x} - \mathbf{x}'\|}{l} + \frac{5\,\|\mathbf{x} - \mathbf{x}'\|^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}\,\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \quad . \tag{11.15}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp - \frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2l^2} \tag{11.16}$$

is obtained. As a rule of thumb the ability to describe smoother and smoother functions increases with $\nu$.[65] 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 $\sigma$ 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. $\sigma^2$ 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.[65] 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.

### 11.1.3 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

$$\log p(\mathbf{v}|\mathbf{X}, \theta) = J^{\theta}_{vx} = -\frac{1}{2}(\mathbf{v} - \mathbf{m})^T \mathbf{K}^{-1} (\mathbf{v} - \mathbf{m}) - \frac{1}{2}\log|\mathbf{K}| - \frac{n}{2}\log 2\pi \tag{11.17}$$

with respect to the hyper parameters $\theta$. 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. 65.

The partial derivatives with respect to the hyper parameters are given as [65]

$$J^{\theta}_{vx} = \frac{\partial}{\partial \theta_i} = \frac{1}{2}\mathrm{tr}\left(\left(\omega\omega^{\mathrm{T}} - \mathbf{K}^{-1}\right)\frac{\partial \mathbf{K}}{\partial \theta_{\mathrm{i}}}\right) \quad , \tag{11.18}$$

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 [66] (Rprop) with an improvement (iRprop) described in Ref. [67]. Rprop uses adaptive update steps, which only use the sign of the gradient

$$\theta_i^{(n+1)} = \theta_i^{(n)} - \text{sign}\left(\frac{\partial J_{vx}^{\theta,(n)}}{\partial \theta_i}\right)\Delta_i^{(n)} \quad . \tag{11.19}$$

The update step is determined according to

$$\Delta_i = \begin{cases} \min(\eta^+ \cdot \Delta_i^{(n-1)}, \Delta_{\max}), & \text{if } \frac{\partial J_{vx}^{\theta,(n-1)}}{\partial \theta_i} \cdot \frac{\partial J_{vx}^{\theta,(n)}}{\partial \theta_i} > 0 \\ \max(\eta^- \cdot \Delta_i^{(n-1)}, \Delta_{\min}), & \text{if } \frac{\partial J_{vx}^{\theta,(n-1)}}{\partial \theta_i} \cdot \frac{\partial J_{vx}^{\theta,(n)}}{\partial \theta_i} < 0 \\ \Delta_i^{(n-1)}, & \text{else} \end{cases} \tag{11.20}$$

If the sign of the partial derivative with respect to $\theta_i$ did not change, the update step $\Delta_i$ is increased by a factor $\eta^+ > 1$ in order to accelerate convergence. However, if the sign changes, the update step $\Delta_i$ 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 $\Delta_0$ and bounded by $\Delta_{\min}$ and $\Delta_{\max}$. In `MidasCpp` we set $\Delta_0$ 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 $\Delta_{\min} = 10^{-6}$.[66] 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

## 11.2 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.

$$\text{cov}\left(V_p, \frac{\partial V_p}{\partial x_{q_i}}\right) = \frac{\partial k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{q_i}} \quad \text{cov}\left(\frac{\partial V_p}{\partial x_{p_i}}, \frac{\partial V_q}{\partial x_{q_j}}\right) = \frac{\partial^2 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{q_j}} \tag{11.21}$$

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

$$V(\mathbf{x}_1), \mathbf{g}(\mathbf{x}_1), \mathbf{h}(\mathbf{x}_1), ..., V(\mathbf{x}_N), \mathbf{g}(\mathbf{x}_N), \mathbf{h}(\mathbf{x}_N) \quad , \tag{11.22}$$

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 $j \leq i = 1, n$. We can then write the co-variance matrix as:

$$\mathbf{K} = \begin{pmatrix} \mathbf{k}_{1,1}^{0,0} & \mathbf{k}_{1,1}^{0,1} & \mathbf{k}_{1,1}^{0,2} & & \mathbf{k}_{1,N}^{0,0} & \mathbf{k}_{1,N}^{0,1} & \mathbf{k}_{1,N}^{0,2} \\ \mathbf{k}_{1,1}^{1,0} & \mathbf{k}_{1,1}^{1,1} & \mathbf{k}_{1,1}^{1,2} & \cdots & \mathbf{k}_{1,N}^{1,0} & \mathbf{k}_{1,N}^{1,1} & \mathbf{k}_{1,N}^{1,2} \\ \mathbf{k}_{1,1}^{2,0} & \mathbf{k}_{1,1}^{2,1} & \mathbf{k}_{1,1}^{2,2} & & \mathbf{k}_{1,N}^{2,0} & \mathbf{k}_{1,N}^{2,1} & \mathbf{k}_{1,N}^{2,2} \\ & & & & & & \\ \mathbf{k}_{N,1}^{0,0} & \mathbf{k}_{N,1}^{0,1} & \mathbf{k}_{N,1}^{0,2} & & \mathbf{k}_{N,N}^{0,0} & \mathbf{k}_{N,N}^{0,1} & \mathbf{k}_{N,N}^{0,2} \\ \mathbf{k}_{N,1}^{1,0} & \mathbf{k}_{N,1}^{1,1} & \mathbf{k}_{N,1}^{1,2} & \cdots & \mathbf{k}_{N,N}^{1,0} & \mathbf{k}_{N,N}^{1,1} & \mathbf{k}_{N,N}^{1,2} \\ \mathbf{k}_{N,1}^{2,0} & \mathbf{k}_{N,1}^{2,1} & \mathbf{k}_{N,1}^{2,2} & & \mathbf{k}_{N,N}^{2,0} & \mathbf{k}_{N,N}^{2,1} & \mathbf{k}_{N,N}^{2,2} \end{pmatrix} \tag{11.23}$$

where we introduced the following shortcuts

$$k_{p,q}^{0,0} = k\left(\mathbf{x}_p, \mathbf{x}_q\right) \tag{11.24}$$

$$(k_{p,q}^{1,0})_i = \frac{\partial k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i}} \quad i = 1, n \tag{11.25}$$

$$(k_{p,q}^{2,0})_{ij} = \frac{\partial^2 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{p_j}} \quad j \leq i = 1, n \tag{11.26}$$

$$(k_{p,q}^{1,1})_{ij} = \frac{\partial^2 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{q_j}} \quad i = 1, n \quad j = 1, n \tag{11.27}$$

$$(k_{p,q}^{2,1})_{ijk} = \frac{\partial^3 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{p_j} \partial x_{q_k}} \quad j \leq i = 1, n \quad k = 1, n \tag{11.28}$$

$$(k_{p,q}^{2,2})_{ijkl} = \frac{\partial^4 k\left(\mathbf{x}_p, \mathbf{x}_q\right)}{\partial x_{p_i} \partial x_{p_j} \partial x_{q_k} \partial x_{q_l}} \quad j \leq i = 1, n \quad l \leq k = 1, n \tag{11.29}$$

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

$$\mathbf{v}^* \left(V(\mathbf{x}^*), \mathbf{g}(\mathbf{x}^*), \mathbf{H}(\mathbf{x}^*), \right)^T \quad , \tag{11.30}$$

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

$$\mathbf{v}^* = \mathbf{K}^* \left(\mathbf{K} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{v} = \mathbf{K}^* \boldsymbol{\omega} \quad , \tag{11.31}$$

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

$$\mathbf{K}^* = \begin{pmatrix} \mathbf{k}_{*,1}^{0,0} & \mathbf{k}_{*,1}^{0,1} & \mathbf{k}_{*,1}^{0,2} & & \mathbf{k}_{*,N}^{0,0} & \mathbf{k}_{*,N}^{0,1} & \mathbf{k}_{*,N}^{0,2} \\ \mathbf{k}_{*,1}^{1,0} & \mathbf{k}_{*,1}^{1,1} & \mathbf{k}_{*,1}^{1,2} & \cdots & \mathbf{k}_{*,N}^{1,0} & \mathbf{k}_{*,N}^{1,1} & \mathbf{k}_{*,N}^{1,2} \\ \mathbf{k}_{*,1}^{2,0} & \mathbf{k}_{*,1}^{2,1} & \mathbf{k}_{*,1}^{2,2} & & \mathbf{k}_{*,N}^{2,0} & \mathbf{k}_{*,N}^{2,1} & \mathbf{k}_{*,N}^{2,2} \end{pmatrix} \tag{11.32}$$

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

### 11.2.1 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

$$k(\mathbf{x},\mathbf{x}') = \sigma^2 \exp\left(-0.5\sum_{i=k}^{d}\frac{(x_k - x_k')^2}{l_k^2}\right) \quad . \tag{11.33}$$

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.

$$\frac{\partial k(\mathbf{x},\mathbf{x}')}{\partial x_i} = -l_i^{-2}\,(x_i - x_i')\,k(\mathbf{x},\mathbf{x}') \tag{11.34}$$

$$\frac{\partial^2 k(\mathbf{x},\mathbf{x}')}{\partial x_j \partial x_i} = -\delta_{ij} l_i^{-2} k(\mathbf{x},\mathbf{x}') - l_i^{-2}\,(x_i - x_i')\,\frac{\partial k(\mathbf{x},\mathbf{x}')}{\partial x_j} \tag{11.35}$$

$$\frac{\partial^3 k(\mathbf{x},\mathbf{x}')}{\partial x_k \partial x_j \partial x_i} = -\delta_{ij} l_i^{-2}\frac{\partial k(\mathbf{x},\mathbf{x}')}{\partial x_k} - \delta_{ik} l_i^{-2}\frac{\partial k(\mathbf{x},\mathbf{x}')}{\partial x_j} - l_i^{-2}\,(x_i - x_i')\,\frac{\partial^2 k(\mathbf{x},\mathbf{x}')}{\partial x_k \partial x_j} \tag{11.36}$$

$$\begin{aligned}\frac{\partial^4 k(\mathbf{x},\mathbf{x}')}{\partial x_l \partial x_k \partial x_j \partial x_i} =& -\delta_{ij} l_i^{-2}\frac{\partial^2 k(\mathbf{x},\mathbf{x}')}{\partial x_l \partial x_k} - \delta_{ik} l_i^{-2}\frac{\partial^2 k(\mathbf{x},\mathbf{x}')}{\partial x_l \partial x_j} - \delta_{il} l_i^{-2}\frac{\partial^2 k(\mathbf{x},\mathbf{x}')}{\partial x_k \partial x_j} \\ & - l_i^{-2}\,(x_i - x_i')\,\frac{\partial^3 k(\mathbf{x},\mathbf{x}')}{\partial x_l \partial x_k \partial x_j}\end{aligned} \tag{11.37}$$

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

$$\frac{\partial k(\mathbf{x},\mathbf{x}')}{\partial x_i'} = -\frac{\partial k(\mathbf{x},\mathbf{x}')}{\partial x_i} \quad . \tag{11.38}$$

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

$$(-1)^l \quad , \tag{11.39}$$

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:

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

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

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

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

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

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

## 11.3 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 $3N_{\text{atom}} - 6$ 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 $N_{\text{atom}} \cdot (3N_{\text{atom}} + 1)/2$ 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.

- `SYMG2`

  The Cartesian coordinates are transformed to a set of atom-centered symmetry functions $G^2$:

$$G_i^2 = \sum_{j \neq i} \exp\left(-\exp(\eta)^2 R_{ij}^2 / R_c^2\right) f_c(R_{ij}) \tag{11.40}$$

  For more details see the entry concerning `SYMG4`.

- `SYMG4`

  The Cartesian coordinates are transformed to a set of atom-centered symmetry functions $G^4$:

$$\begin{aligned} G_i^4 =& 2^{1-\xi} \sum_{j,k \neq i (j \neq k)} (1 + \lambda \cos\theta_{ijk})^{\xi} \\ & \times \exp\left(-\exp(\eta)^2 / R_c^2 \left(R_{ij}^2 + R_{ik}^2 + R_{jk}^2\right)\right), \\ & \times f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}) \end{aligned} \tag{11.41}$$

  where $f_c$ is a cutoff function defined as

$$f_c(R_{ij}) = \begin{cases} 0.5\left(1 + \cos\frac{\pi R_{ij}}{R_c}\right), \text{if } R_{ij} \leq R_c \\ 0, \text{if } R_{ij} > R_c \end{cases} \tag{11.42}$$

  and $\theta_{ijk}$ is the angle between the triplet of atoms $i, j$, and $k$. The parameter $\lambda$ is either 1 or $-1$ and one should include both in a descriptor. The parameter $R_c$ is the cutoff radius while $\eta$ and $\xi$ control the sensitivity with respect to radial and angular information around atom $i$. For $G_i^2$ and $G_i^4$ the analytical derivatives are implemented, which allows to learn gradient and Hessian information using these descriptors.

# Part III

# MidasCpp Input/Output Reference Manual

# Chapter 12

# Midas Input Basics

In the following chapters, the keywords used by MidasCpp will be described in detail. Before that, general descriptions of the basic input structure and syntax, as well as operators and mathematical functions which MidasCpp is able to handle are provided. Additional information about the MidasCpp input format can be found in section 1.3 and appendix A.

## 12.1 Input structure and keyword syntax

A keyword is prepended by a hashtag, #, and a number indicating which level the keyword belongs to. The input of the keywords is not case-sensitive, but additional string input read by a keyword might be. Furthermore, any spaces present in the keyword will be ignored. Correspondingly, the inputs "#1General", "#1GENERAL", and "#1Ge Ne Ral" written on a line in the MidasCpp input will be interpreted in the same way. Some keywords are used to indicate the start of an input block, where each keyword belonging to that specific block will have a level of one higher than the level of the keyword that initializes the block. Some keywords can occur on several levels and will in the following chapters be prepended with <N>, where the number N has to be specified to correspond to the correct level of input in an actual calculation.

The keywords are read from top to bottom of the file they appear in. Generally, a set of begin and end keywords with level 0 will be used to indicate the beginning and end of an input file. The MidasCpp input file (.minp), for instance, is read from the #0 MidasInput keyword until the #0 MidasInputEnd keyword is reached.

The keywords presented in the following chapters will be classified as either general or advanced. The general keywords are those most commonly used for a standard MidasCpp calculation. A keyword might be related to a certain type of calculation, in which case a box to the right of the keyword name with a reference to one of the previous chapters discussing the calculation type will be given. Furthermore, if the keyword accepts or expects arguments the type of the arguments will be given under the keyword name. If default values of the arguments will be used in case the keyword is *not* specified on input, their values will also be specified on this line.

## 12.2 Rules for operator inputs (Midas Operator)

In this Section, valid operator input in MidasCpp is described.

### 12.2.1 Basic syntax for operator input

Operators in MidasCpp are required to have a sum-of-products form, meaning that any operator must consist of a sum of products of one-mode operators. The eligible forms of the one-mode

operators are described in the following sections. A line of operator input in the MidasCpp input (as for example specified through the #<N> OperatorTerms keyword), consists of a *real*, and a string that represents a product of one-mode operators:

```
#1 OperatorTerms
   0.25 Q^1(Q1) Q^1(Q3)
```

Each one-mode operator in a given product is expanded by a postfix (<mode_label>) that indicates the mode of which that operator is a function. The one-mode operators in an operator product are separated by a whitespace. The terms in a one-mode operator have to be provided without whitespace, i. e. the following one-mode operator is not valid:

```
#1 OperatorTerms
   0.25 Q^1 *  (DDQ)^1(Q1)
```

**Importantly, each mode can contribute at most a single one-mode operator to an operator product, i. e. the following operator term definition is not valid:**

```
#1 OperatorTerms
   0.25 Q^1(Q1) Q^1(Q1)
```

### 12.2.2 Mode labels and order

MidasCpp has several keywords which apply a certain condition to the modes in an ordered way, e.g. #3 LimitModalBasis. The mode order used under these keywords can be provided in three different ways. When running a PES calculation, the mode order is the same as the order of the modes specified in the .mmol file (or any of the other types of coordinate input MidasCpp accepts). The modes can be named in this file, but if they are not, the modes will be assigned the labels Q0, Q1, ..., QN-1, where N is the number of modes in the given input file. (We note however that, internally, MidasCpp does not use the mode labels to keep track of the modes, but an integer that is assigned to each mode.)

For a wave function calculation, there are two options. The recommended method is to use the keyword #<N> ModeNames in the operator input. The string of mode names under this keyword will specify the order of modes in MidasCpp and the rest of the input file.

If the #<N> ModeNames keyword is omitted from the operator file, the following ordering of the mode names will be adopted, using the operator terms under #<N> OperatorTerms. First, the operator terms are ordered by the number of one-mode operators they contain, i. e. all one-mode operator terms come before the two-mode operator terms etc. Then, the mode names are split into three parts. The first part is the string up till the first numeral found in the mode name. The second part is the first full number, i. e. string or numerals, found in the mode name. The third part is a string containing the rest of the mode name. The terms are read one by one and the ordering of the modes are then decided by

- the number of terms in the operator term the mode name first appears in,
- the first string part of the mode name,
- the first numeral of the mode name, and
- the last string part of the mode name.

An example is given by the following operator terms, which will result in the mode order Q2, Q1 and Q3.

```
#1 OperatorTerms
   0.25 Q^1(Q1) Q^1(Q3)
   0.50 Q^1(Q2)
```

Older operator input formats use the mode numbers from 1 to N directly to denote the mode names and by extension the order in which the modes are given. Correspondingly, for these inputs the mode number simply corresponds to the actual number used in the operator file.

### 12.2.3 Non-multi-state one-mode operators

Excluding multi-state operators (see Section 12.2.4), one-mode operators in `MidasCpp` have to abide by certain general forms. To be precise, the operators have to be either pure multiplicative functions (here: $F(Q)$ (`F(Q)`)), $n^{\text{th}}$ order derivatives, $\frac{\partial^n}{\partial Q^n}$ (in `MidasCpp` string format: `(DDQ)^n`), or combinations of these according to one of the following expressions:

- $F(Q) \cdot \frac{\partial^n}{\partial Q^n} \quad \longrightarrow \quad$ `F(Q)*(DDQ)^n`
- $\frac{\partial}{\partial Q} \cdot F(Q) \quad \longrightarrow \quad$ `(DDQ)^1*F(Q)`
- $\frac{\partial}{\partial Q} \cdot F(Q) \cdot \frac{\partial}{\partial Q} \quad \longrightarrow \quad$ `(DDQ)^1*F(Q)*(DDQ)^1`.

#### General information on the derivative operators

Firstly, regarding the derivative operators, the string `(DDQ)^n` is reserved to denote an $n^{\text{th}}$ order derivative in `MidasCpp` and cannot be overwritten. Derivative operators typically occur in kinetic energy and flux operators, and they are inserted into a `MidasCpp` calculation using either #3 KineticEnergy or some of the more advanced keywords in Chapter 17. Correspondingly, one should proceed with caution when defining operators containing derivatives manually, and check carefully if additional terms were added through the keywords in Chapter 17. Derivative operators are only allowed in `MidasCpp` operator input, and not for any of the other input channels that accept general functions, for example the definition of a custom fitting basis.

#### Operators combining scalar functions and derivatives

With respect to the one-mode operator forms that combine multiplicative and derivative operators, there are several important notions. First, derivatives on the left of a multiplicative function can only be first-order derivatives. The technical reason for this limitation lies in the way the one-mode integrals of the operator terms are calculated in `MidasCpp` and it is therefore non-trivial to extend the code to allow for further combinations of derivative and multiplicative operators. Moreover, when calculating matrix elements over operator terms involving derivatives, the derivatives are taken of the one-mode wave functions alone (applying partial integration), whereas the function `F(Q)` is not differentiated. Correspondingly, operator terms of the form $G(Q) \cdot \frac{\partial^i}{\partial Q^i} \cdot F(Q)$ with $G(Q)$ and $F(Q)$ as scalar functions are not valid operator input. As another result, the availability of automatic differentiation for a general function `F(Q)` (see Section 12.3) is not relevant in the context of evaluating matrix elements of operators with derivatives described in this subsection.

#### Scalar and distribution functions in the one-mode operators

The options and rules for inputting and using scalar functions $F(Q)$ as part of one-mode operators in `MidasCpp` are provided below in Sec. 12.3. The full general function framework can be used to generate the functions $F(Q)$ (`F(Q)`) for the one-mode operators. In addition to that, the function $F(Q)$ may contain certain distributions, namely Dirac delta functions and Heaviside step

functions. The operator string that can be used to input a Dirac delta function is `DELTA(X)`, where `X` is a number that specifies the position of the discontinuity of the delta function. For instance, `DELTA(1.5)` corresponds to the mathematical expression $\delta(x-1.5)$. Moreover, there are two options to provide a Heaviside step function. The operator input string `HEAVI(+Q-X)`, where `X` is a number that specifies the position of the step, specifies a step from 0 to 1 at `X`. For example, the string `HEAVI(+Q-2.0)` denotes the mathematical expression $\Theta(x-2.0)$. Additionally, the operator string `HEAVI(-Q+X)` can be used to provide a step function that jumps from 1 to 0 at `X`. In case that a one-mode operator contains a distribution, the special integration properties of the respective distribution function are exploited when calculating the matrix elements of the operators.

Several restrictions apply to the usage of the distribution functions as of now:

- **There can only ever be one distribution in a one-mode operator term, not more.**
- **The distributions can only occur as factors in $F(Q)$, not as summands, etc.!**
- **The distributions cannot be used in fitting procedures of molecular properties.** Somewhat related, distribution functions can also not be used in contexts where automatic differentiation of general functions is required (see appendix A.2).
- **If frequency-scaled coordinates are used, the position of the discontinuity in the distribution has to be provided in frequency-scaled coordinates as well!**

Examples of the application of distribution functions in `MidasCpp` are given in the regression tests `delta_function_bsplines` as well as `heaviside_functions_particle_in_box` and `heaviside_functions_bsplines` in the `test` directory.

### 12.2.4 Multi-state operators

This subsection deals with the specifics for inputting a multi-state operator to use with the multi-state wave function calculation options in `MidasCpp`. There is currently no way of creating this type of operator automatically from the PES module of the `MidasCpp` program. One therefore has to create this type of input by hand, and for this reason it is considered an expert option within the `MidasCpp` framework. It should furthermore be noted that this option is only available using the standard `MidasCpp` operator input described in Chapter 40 and not for any other operator formats that `MidasCpp` is able to read. In the appendix A.1, an example of the multi-state operator input based on the `mctdh2v_nonadiabatic` test case in the `test` directory will be shown.

A requirement for specifying a multi-state operator is the presence of the #<N> ElectronicDof keyword. This keyword is used to specify a dummy mode name, which is used to identify the state-specific or state-coupling operator terms. In the example in the appendix, this mode name is given as `S`. The introduction of the dummy mode also requires an additional scaling factor under the #<N> ScaleFactors, which has to be specified as 1.0 to avoid any specific scaling of the dummy mode terms. In the example below, #<N> Frequencies is used, but the convention remains the same.

The state-coupling operators are given as `|N><M|`, where `N` and `M` are integers. `N` and `M` can have the same value to signify an operator term which is specific to a state `N`. If `N` and `M` have different values, the operator term signifies a state-coupling operator between state `N` and `M`. The numbering of the states given by the `N` and `M` indices are not required to have any physical meaning beyond labeling the electronic states of the molecule. The state numbering used is however required to start counting from state `0` up to state `P`, where `P+1` is the total number of electronic states considered in the calculation. For example, if the state `1` was renamed to `3` in the example in appendix A.1,

that is the term `|0><1|(S)` becomes `|0><3|(S)` etc., the calculation would fail as there are only two states in the calculation.

## 12.3 Rules for function inputs (Midas Operator)

The `MidasCpp` framework can accept functions as input in a number of contexts. Currently the main uses are outlined in chapter 40 and 26 of this manual. It is important to note that all function-related input is not case-sensitive, as all input is initially converted to upper-case before being interpreted.

A detailed explanation of the technical details of the implementation of general functions in `MidasCpp` are given in Ref. [19]. Briefly, the implementation reads an expression from input where the symbols `+`, `-`, `*`, `/`, `^`, `(` and `)` are used to denote addition, subtraction, multiplication, division, raising, left and right parenthesis, respectively. The expression is recast into reverse Polish notation and the Shunting-Yard algorithm is used to generate an evaluation tree for internal use in `MidasCpp`. It should be noted that the minus operator, `-`, can both be used to denote a negative number, e. g. `-4.5`, and subtraction, e. g. `1.3-0.3`. Furthermore, internally in `MidasCpp` a distinction is made between expressions being raised to an integer power and those which are not - that is, the expression `Q^3` and `Q^3.0` are not exactly the same. For the computation of the function value, this should not produce different results, but especially for the evaluation of derivatives, the distinction is important. It is therefore recommended that one raises to integer powers where it is possible, and only uses floating point numbers or expressions when absolutely necessary.

Aside from the two exceptions given above, the definitions of an expression works as one would expect. For instance, the evaluation of `a*b+c` and `a*(b+c)` leads to different results. One should note that multiplication symbols cannot be omitted near variable names, function names or parentheses. That is, the inputs `4(2.5+3.2)`, `32.3Q^5` and `1.4F(Q)` are *invalid* and lead to a program crash. The valid forms of the expressions are, in order, `4*(2.5+3.2)`, `32.3*Q^5` and `1.4*F(Q)`.

Generally, each function given to `MidasCpp` is the function of a number of variables. These are usually given in parentheses after the expression, for example, `x*y+z(x,y,z)` corresponds to the expression `x*y+z` as a function of the variables `x`, `y` and `z`. The exception to this convention are operator terms given under the #<N> OperatorTerms keyword, where `Q` is used as a universal variable indicator related to the motion along a given mode. (The mode name given in the parentheses after the expression for the given one-mode term is the mode which the given operator term is related to. To give an example, `Q^5(Q10)` is the one-mode operator term `Q^5`, which is applied for mode `Q10`.) For any expression involving division, the divisor is explicitly checked to be different from zero for each evaluation. If a zero divisor is encountered, the program will crash with an error message.

Some input channels, such as the `MidasCpp` operator reader described in Chapter 40, allow for the definition of functions which can be used to simplify the input of subsequent functions. Functions are defined by a string and a function definition. The string could in principle be anything except names overlapping with the mathematical functions mentioned later in this subsection. The function definition contains a mathematical expression, including previously defined functions, and ends with a pair of parentheses containing a comma-separated list of the variables used. For instance, one could define the function `SQR` which squares its input by the following, `x^2(x)`. Functions are called by their given function name and a pair of parentheses containing the parameters passed to the function, e.g. to call the function `SQR` on the value `4.5`, one would write `SQR(4.5)`. It is also possible to nest function calls, so one could for instance call `SQR(SQR(4.5))` to get the value `(4.5^2)^2`.

Additionally, there is the possibility to pass variables, such as `Q`, or even expressions involving variables to a function. For example, the function input `SQR((Q*3.5)/(Q+1.0))` is interpreted as `((Q*3.5)/(Q+1.0))^2`. Furthermore, user-defined functions can accept several variables, as is illustrated by the function `fyx` defined as `x^2+y(x,y)`, which is called as `fyx(Q-3,Q+5)` to produce the function `(Q-3)^2+(Q+5)`.

Currently, the following mathematical functions are defined directly in the `MidasCpp` source code and are excluded from redefinition: `cos`, `sin`, `exp`, `tan` and `sqrt` (square root). All of these functions follow the standard function typesetting and only accept one argument, i. e. the cosine of `Q` is obtained from `cos(Q)`. For the trigonometric functions, we assume that their arguments are given in radians.

## 12.4 General keywords

### #0 MidasInput

This keyword is used to signify the beginning of a `MidasCpp` input file. All lines before this keyword will be ignored. All keywords presented in the following chapters are allowed in the `MidasCpp` input file, unless otherwise noted.

### #0 MidasInputEnd

This keyword is used to signify the end of a midas input file. All lines after this keyword will be ignored. This keyword is paired with the #0 MidasInput keyword.

# Chapter 13

# General input options

## 13.1 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

- 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.

**#2 Maindir**

< *string* >

Indicates the path to the main directory of a calculation. Directories for each multilevel surface calculation can be found in the main directory.

## 13.2 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 14

# Singlepoint input options

## 14.1 General keywords

### #1 SinglePoint

Read SinglePoint input until next level #1 input key.

### #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 InputCreatorScript (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 DaltonBasis

< *integer* *string* >
[< *integer* *string* >]

Assign the basis set to be used for the QM model when run by the DALTON program. The basis sets are given as a list of pairs consisting of nuclear charge and a basis set label.
(Only has an effect for `SP_DALTON` SP calculations)

**Example for DaltonBasis**

```
#2 DALTONBASIS
1  cc-pVDZ   // assign cc-pVDZ to hydrogen
8  cc-pVDZ   // assign cc-pVDZ to oxygen
```

### #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 DeterminantThr**

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

Threshold for comparing the determinant of the rotation matrix to +1 (or −1, if improper rotations are allowed, see #2 AllowImproperRotation).
(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 DoubleWellDefs**

< *string* >

Used in combination with #2 DoubleWellPot to generate a double-well potential. The string includes a set of assignments seperated by commas. The values needs to be assigned are: freq ($\omega$), b_val ($b$), rho ($\rho$), red_mass ($\mu$) and name. $\omega$ is the harmonic frequency in $\mathrm{cm}^{-1}$. $b$ is the barrier height of the double well potential in $\mathrm{cm}^{-1}$. $\mu$ is the reduced mass of the system in AMU. $\rho$ is a parameter that determines the shape of the potential. The name is the name of the property calculation according to the values read in from #2 PropertyInfo . The form of the potential is $V(r) = \frac{1}{2}\omega^2 Q^2 + A\exp(-a^2Q^2) - V_{min}$ where $V_{min}$ is the potential minimum energy, $A = \frac{b\rho}{\exp(\rho)-\rho-1}$, and $a = \frac{\exp(\omega^2\rho)}{2A}$
(No effect unless the #2 DoubleWellPot is defined. The order of the assignments does not matter.)

**Example for DoubleWellDefs**

```
#2 DoubleWellDefs
freq=974.2,red_mass=2.48658405611024369,b_val=2032.0,name=GSE,rho=0.6
```

**#2 DoubleWellPot**

Creates a double-well potential using the parameters given by #2 DoubleWellDefs. If #2 DoubleWellDefs is not given, the double-well potential for the $NH_3$ inversion parameterized by J. B. Coon et al. (*J.* Mol. Spectrosc., **20**(1966), 107) will be used. Look in the `test_suite` directory for input/output examples, fx pes_Adga_DoubleWell.
(Only has an effect for `SP_MODEL` SP calculations.)

**Example for DoubleWellPot**

```
#1 Singlepoint
#2 Name
DW_singlepoint
#2 Type
SP_MODEL
#2 DoubleWellPotential
#2 DoubleWellDefs
freq=974.2,b_val=2032.0,rho=0.6,red_mass=2.48658405611024369,name=GSE
```

**#2 DynlibFile**

< *string* >

Specifies the filename/path of the MidasDynlib. Used in combination with #2 DynlibPot.
(Only has an effect for `SP_MODEL` SP calculations.)

### #2 DynlibPot

Specifies a potential based on a MidasDynlib. When this keyword is specified the keyword #2 DynlibFile must also be present in the input block.
(Only has an effect for `SP_MODEL` SP calculations.)

**Example for DynlibPot**

```
#1 singlepoint
#2 Name
ammonia_rimp2_pot
#2 Type
SP_MODEL
#2 DynlibPot
#2 DynlibFile
ammonia_rimp2_augccpvtz_dynlib.so.1.0
```

### #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 FilePotential

Specifies a potential created by using the MidasCpp operator files. When this keyword is specified #2 OperatorFile and #2 MoleculeFile also needs to be present in the input block. Look in the `test_suite` directory for input/output examples, fx. pes_Adga_Filepot and pes_extrap1
(Only has an effect for `SP_MODEL` SP calculations.)

**Example for FilePotential**

```
#1 singlepoint
#2 Name
file_two_state_pot
#2 Type
SP_MODEL
#2 FILEPOTENTIAL
#2 MOLECULEFILE
Molecule.mmol
#2OperatorFile
prop_no_1.mop GROUND_STATE_ENERGY
prop_no_2.mop EXCITED_STATE_ENERGY
```

**#2 IoLevel**

< *integer* = 2 >

Set IO level for singlepoint calculations (only has an effect for the internal singlepoint framework in `MidasCpp` ).

**#2 InputCreatorScript**

< *string* = InputCreatorScript.sh >

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 MoleculeFile**

< *string* >

Used in combination with #2 FilePotential. Sets the file name of the molecule file, see section 41, used to transform the xyz coordinates provided by MidasCpp to the normal coordinate space of the operators given in #2 OperatorFile. This molecule file should be the same as the one being used to generate all the operators used for the FilePotential.
(Only has an effect for `SP_MODEL` SP calculations, using the #2 FilePotential option)

**#2 MorseDefs**

< *string* >

Used in combination with #2 MorsePotential to generate a Morse potential. The string includes a set of assignments seperated by commas. The values needs to be assigned are: de ($D_e$), acoeff ($a$), req ($R_e$) and name. $D_e$ is the electronic dissociation energy of the potential in au. $a$ is the exponent parameter in au. $R_e$ is the equilibrium distancense in au. The name is the name of the property calculation according to the values read in from #2 PropertyInfo. The form of the potential is as follows $V(r) = D_e(1 - \exp(-a(r - R_e)))^2$. Multiple lines can be used to define multiple morsepotentials with different property names.
(Only has an effect for `SP_MODEL` SP calculations, using the #2 MorsePotential option. The order of the assignments does not matter.)

**Example for MorseDefs**

```
#2MorseDefs
name=GROUND_STATE_ENERGY,de=0.1026,acoeff=0.732,req=1.9972
```

**#2 MorsePotential**

Creates a Morse potential using the parameters given by #2 MorseDefs. If #2 MorseDefs is not given, the Morse potential for $H_2^+$ is used with $D_e = 0.1026$ au, $R_e = 1.9972$ au and $a = 0.732$ au. This type of potential allows for the evaluation of gradients and hessians. Look in the `test_suite` directory for input/output examples, Fx. pes_Adga_Morse.
(Only has an effect for `SP_MODEL` SP calculations.)

**Example for MorsePotential**

```
#1 Singlepoint
#2 Name
morse_singlepoint
#2 Type
SP_MODEL
#2 MorsePotential
#2MorseDefs
de=0.1026,req=1.9972,acoeff=0.732,name=GROUND_STATE_ENERGY
```

### #2 Name

< *string* >

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

### #2 OperatorFile

< *string* >

Used in combination with #2 FilePotential. Sets the filenames and property names of the operator files used for generating the properties in the File Potential instance. Each line until the next keyword input is read and threated in the following way. Two space seperated strings are read, the first is the filename of the operator file the second is the property name. (Only has an effect for `SP_MODEL` SP calculations, using the #2 FilePotential option)

**Example for OperatorFile**

```
#2OperatorFile
prop_no_1.mop GROUND_STATE_ENERGY
prop_no_2.mop EXCITED_STATE_ENERGY
```

### #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`-directory of the MidasCpp source 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 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 higher order properties like the gradient it is important to rotate the structure 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.)

**#2 RotationScheme**

< *string* = Kabsch > [`Legacy`] [`Kabsch`]

Choose the algorithm used to find the rotation matrix to back-rotate the structure obtained from an external ES program. For higher order properties like the gradient it is important to rotate the structure back to the `MidasCpp` reference frame.
(If `MidasCpp` is unable to determine the rotation it might help to change the algorithm used here.)

**#2 RunScript**

< *string* = RunScript.sh >

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 SaveSpScratchDir**

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 SetupDir**

< *string* >

Specifies a directory/path to search for the files given by #2 InputCreatorScript, #2 RunScript, and #2 ValidationScript.
(Only has an effect for `SP_GENERIC` SP calculations. The recommendation is to NOT set this keyword for PES calculations.)

**#2 SinglePointScratchPrefix**

< *string* >

This keyword allows the user to specify a location for the scratch directories of the singlepoint calculations that may be run during a frequency analysis or the construction of a PES/property surface.

#### #2 TruncateZeroValues

Specifies that properties that have a near-zero value is truncated to be exactly zero. A value will be set to zero if it is smaller than $1.0 \cdot 10^2$ times the machine epsilon.

#### #2 Type

< *string* = N/A >

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)

## 14.2 Advanced keywords

#### #2 AllowImproperRotation

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

Allow improper rotations when back-rotating molecules to the `MidasCpp` frame of reference. (Consider using this keyword if `MidasCpp` keeps finding back-rotations with determinant close to $-1$.)

# Chapter 15

# Vib input options

## 15.1 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 16 for a list of the basis-set keywords.

### #2 McTdH

!MCTDH

Read input for the MCTDH module until next #2 key. See chapter 22 for a list of the MCTDH keywords.

### #2 Operator

!Operator

Read operator input until next #2 input key. See chapter 17 for a list of the operator keywords.

### #2 TdH

!TDH

Read input for the TDH module until next #2 key. See chapter 21 for a list of the TDH keywords.

### #2 TdVcc

!TDVCC

Read input for the TDVCC module until next #2 key. See chapter 23 for a list of the TDVCC keywords.

**#2 Vcc** !VCC !VCI !VMP

Read input for the correlated vibrational wave function until next #2 key. See chapter 19 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 18 for a list of the VSCF keywords.

## 15.2 Advanced keywords

**#2 IoLevel**

$< integer = 0 >$

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

# Chapter 16

# Basis input options

## 16.1 General keywords

**#3 BSplineBasis** !VibBasis

< *integer* = 10 **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). Reading bounds from a file is recommended although the default behaviour is to use the classical harmonic oscillator turning points for bounds, see #3 TurningPoint for default.

**#3 GaussianBasis** !VibBasis

< *real* = 1.0 **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 $\alpha$ 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).

**#3 GradScalBounds** !VibBasis !ADGA

< 2 *reals* = 1.5 30.0 >

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.

**#3 HOBasis** !VibBasis

< *integer* = 20 **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.

**#3 IoLevel** !VibBasis

< *integer* = 0 >

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

**#3 MaxNoPrimBasisFunctions** !VibBasis

< *integer* = 500 >

Reads in a single integer. The value indicates the maximal number of distributed Gaussian or B-spline functions that can be generated and inserted into the primitive basis for each mode. The keyword can be useful with DIF/DIFACT computations, where a too large number of primitive basis functions can often be generated.
(Recommended: 150–500)

**#3 NPrimBasisFunctions** !VibBasis

< *integer* = 200 **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.

**#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* = 0.8 **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.

#### #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>/one_mode_grids.mbounds, 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 *reals* $=$ 1.5 20.0 $>$

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. See also #3 GradScalBounds.
(In general the recommendation is to use #3 GradScalBounds. If using #3 ScalBounds, then the first number (scaling factor) should be in the range $[0.9, 2.0]$, and 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).)

## 16.2 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 SplitBasis

!VibBasis

$<$ *integer real* **OR** *real vector* $>$ [`<mode number> <split value(s)>`]
.
.
[$<$ *integer real* $>$]

Read in a number of lines equal to the number of modes to use the split basis option for. Each line specifies a mode to use the split basis option for as well as one or more the split values to split the basis at. Lines are read until the next keyword is found. Split basis is only available when using a #3 BSplineBasis.

**#3 TurningPoint** !VibBasis

< *integer* = 10 >

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. It is generally recommened to read bounds from a file and not use turning points.
(Recommended: 10)

# Chapter 17

# Operator input options

## 17.1 General keywords

**#3 CoriolisFile** !Operator

< *string* >

The string shall give the file name for the file containing the Coriolis matrices needed for the construction of the approximate or complete Coriolis operator. A file named coriolis_matrices.mpesinfo containing the Coriolis matrices can be generated and found in the `FinalSurfaces/savedir` if a calculation with the #2 CalcEffInertiaInvTens keyword is done beforehand.

**#3 Inertia** !Operator

< 3 *reals* >

Read in a line with 3 numbers, which defines the moments of inertia of the molecular system reference structure in a.u. If only this keyword is given in additon to the #3 KineticEnergy keyword using the ConstWatson option, then an approximate form of the Watson operator is used, where only the potential correctional term is applied. The moments of inertia can be taken from any source but also generated and found in the output if a calculation with the #2 CalcEffInertiaInvTens keyword is done beforehand. Note that the reference structure is assumed to be oriented along the $z$-axis for linear molecules and the $z$ component of the moment of inertia must be zero.

**#3 InertiaTensorFile** !Operator

< *string* >

Reads in a line with a string, which indicates the path and file name for the file containing the moment of inertia tensor in a.u. for the molecular reference geometry.

**#3 InvEffInertiaFiles** !Operator

< *integer* >

< *string string* >

.

.
< *string string* >

Read an integer giving the number of operator files for the inverse effective inertia tensor components, i.e. the $\boldsymbol{\mu}$ tensor components. A number of lines (corresponding to the value of the integer) are then read. These lines should contain two strings, where the first string indicates the file name and the second string gives the pertinent tensor component. For linear molecules, the tensor contain only one uinque component, which must be denoted LIN. For non-linear molecules, the tensor contains six unique components, which must be denoted XX, XY, XZ, YY, YZ, ZZ (corresponding to the upper triangular part). For non-linear molecules, the trace of the tensor can optionally be supplied as a separate file, denoted TRACE. The trace is used for the full pseudopotential term. If the trace is not supplied separately, it will be computed simply as the sum of the XX, YY and ZZ components (this procedure is of course mathematically equivalent, but may differ numerically due to differences in fitting etc.).

**Example for InvEffInertiaFiles**

```
#3 InvEffInertiaFiles // Example for water (non-linear).
6
h2o_mu_xx.mop  XX
h2o_mu_xy.mop  XY
h2o_mu_xz.mop  XZ
h2o_mu_yy.mop  YY
h2o_mu_yz.mop  YZ
h2o_mu_zz.mop  ZZ

#3 InvEffInertiaFiles // Example for water (non-linear with separate trace).
7
h2o_mu_xx.mop  XX
h2o_mu_xy.mop  XY
h2o_mu_xz.mop  XZ
h2o_mu_yy.mop  YY
h2o_mu_yz.mop  YZ
h2o_mu_zz.mop  ZZ
h2o_mu_tr.mop  TRACE

#3 InvEffInertiaFiles // Example for CO2 (linear)
1
mu.mop  LIN
```

### #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* > [None] [UserDefined] [Simple] [General] [Polyspherical] [Watson] [LinWatson] [FromFile]

Indicates the kind of nuclear kinetic energy operator to be used. For energy-type operators, this option defaults to Simple. For other type of operators, the default is None. Current options cover:

**None** No kinetic energy terms are assumed to be present in the operator supplied by the user (using #3 OperFile or #3 OperInput). No kinetic energy terms will be supplied by `MidasCpp`. This option should be used with non-energy type operators.

**UserDefined** The user manually specifies the kinetic energy terms in the operator (using #3 OperFile or #3 OperInput). No further kinetic energy terms will be supplied by `MidasCpp`.

**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** A general curvilinear coordinate KEO defined in terms of the $G$ matrix and (optionally) the extrapotential $V_{\text{ep}}$ is provided. $G_{ij}$ and $V_{\text{ep}}$ are given in terms of either Taylor expansions or $n$-mode expansions. The type of expansion is set through #4 KeoExpansionType and the G matrix is provided through #4 GMatrixExpansion. The addition of the extrapotential is activated through #4 AddExtrapotential and #4 ExtrapotentialExpansion. Coefficient screening is managed through #4 KeoScreening and #4 KeoScreeningThreshold. Additional truncated Taylor expansion-based schemes can be set via #4 TaylorKeoGmatDiag and #4 TaylorModeTruncApprox.

**Polyspherical** The kinetic energy operator in polyspherical coordinates will be used and should be supplied by the user as a standard operator file. The path of the file holding the kinetic energy operator must be supplied using #4 KeoFile.

**Watson** The Watson operator for a non-linear molecule (see Section 5.1.1). The simple kinetic energy operator, the Coriolis coupling term and the pseudopotential term are automatically generated. The default behaviour is to use an `n`-mode expansion for the inverse effective inertia tensor in both terms, which requires the additional specification of the #3 Inertia or #3 InertiaTensorFile keywords as well as the #3 CoriolisFile and #3 InvEffInertiaFiles keywords. The equilibrium approximation of the inverse effective inertia tensor can be enabled for one or both terms by using #4 Coriolis and/or #4 PseudoPot.

**LinWatson** The Watson operator for a linear molecule (see Section 5.1.2). The simple kinetic energy operator and the Coriolis coupling term are automatically generated. The default behaviour is to use an `n`-mode expansion for the inverse effective inertia tensor in the Coriolis term, which requires the additional specification of the #3 Inertia or #3 InertiaTensorFile keywords as well as the #3 CoriolisFile and #3 InvEffInertiaFiles keywords. The equilibrium approximation of the inverse effective inertia tensor can be enabled by using #4 Coriolis.

**FromFile** An arbitrary sum-of-product kinetic energy operator supplied by the user as a standard operator file. The path of the file holding the kinetic energy operator must be supplied using #4 KeoFile.

### #4 AddExtrapotential

!Operator

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

To be used with #3 KineticEnergy option `General` and #4 KeoExpansionType. Activates the addition of the scalar extrapotential term to a general-coordinate KEO. If this is active, #4 ExtrapotentialExpansion has to be set as well.

### #4 Coriolis

!Operator

< *string* = EXPANDED > [`NONE`] [`EQUILIBRIUM`] [`EXPANDED`]

To be used with the #3 KineticEnergy options `Watson` and `LinWatson`. Indicates how the Watson Coriolis term should be treated. The option `NONE` discards the term altogether. The option `EQUILIBRIUM` approximates the term by using the equilibrium value of the inverse effective inertia tensor (this requires the additional specification of the #3 Inertia or #3 InertiaTensorFile keywords and the #3 CoriolisFile keyword). The option `EXPANDED` represents the term by using an $n$-mode expansion of the inverse effective inertia tensor (this requires the additional specification of the #3 Inertia or #3 InertiaTensorFile keywords and the #3 InvEffInertiaFiles and #3 InvEffInertiaFiles keywords).

### #4 ExtrapotentialExpansion

!Operator

< *string* = N/A >

To be used with #3 KineticEnergy option `General` and #4 ExtrapotentialExpansion. Provides the path to and name of file containing the expansion of the extrapotential. If #4 KeoExpansionType is provided with `TAYLOR`, the coefficient file is generated by the external software Tana/Tnum. In this case, the same expansion order is used as for the G matrix. If #4 KeoExpansionType is provided with `NMODE`, the user has to provide a `.mop` file containing the expansion terms.

### #4 GMatrixExpansion

!Operator

< *integer* = N/A >
< *string* = N/A >

To be used with #3 KineticEnergy option `General` and #4 KeoExpansionType. Provides the required information on the G matrix for general-coordinate KEOs. The first argument is the order of the Taylor expansion of the G matrix (options: 0, 1, 2; same order used for extrapotential) if the input under #4 KeoExpansionType is `TAYLOR`, whereas it is the number of active modes in the `NMODE` case. The second argument is the path to and name of the file containing the expansion coefficients of the G matrix. For a Taylor expanded KEO, the coefficient file is generated separately through the external software Tana/Tnum. For an n-mode expanded KEO, the user has to provide the first $G_{ij}$ .mop file to be read. (The KEO is constructed only from the upper triangle of the $G$ matrix, so that for a system of $M$ modes $M \cdot (M+1)/2$ .mop files need to be read. Their names have to composed of a common base name and a step-wisely increasing integer index **connected by an underscore** (e. g. G_1.mop, G_2.mop, G_3.mop, ...). Importantly, the operator files have to contain the reference value of the respective G matrix element! (The reference value is provided through the #<N> ReferenceValue keyword in the operator file. If no reference value is provided, the default value is zero.)

#### #4 KeoExpansionType

!Operator

< *string* = N/A > [TAYLOR] [NMODE]

To be used with #3 KineticEnergy option `General`. Specifies the type of expansion that is used to describe the G matrix and extrapotential for general-coordinate KEOs. This keyword requires the user to set a G matrix for the KEO construction (#4 GMatrixExpansion).

#### #4 KeoFile

!Operator

< *string* = N/A >

To be used with the #3 KineticEnergy options `Polyspherical` and `FromFile`. Indicates the path to he file holding the kinetic energy operator. The file should be written in standard `MidasCpp` operator format.

#### #4 KeoScreening

!Operator

< *boolean* = FALSE > [TRUE] [FALSE]

To be used with #3 KineticEnergy option `General` and #4 KeoExpansionType. Activates screening of terms in general-coordinate KEOs. For Taylor expansion-based schemes, this refers to screening of 3M- and 4M-terms in the KEO terms stemming from the G matrix. For n-mode expansion-based schemes, this refers to screening of the coefficients of all terms in the expansions of G matrix and extrapotential (if used). If screening if activated, #4 KeoScreeningThreshold has to be set as well.

#### #4 KeoScreeningThreshold

!Operator

< *real* = 0.0 >

To be used with #3 KineticEnergy option `General` and #4 KeoScreening. Provides a threshold for screening of general-coordinate KEO terms.

#### #4 PseudoPot

!Operator

< *string* = EXPANDED > [NONE] [EQUILIBRIUM] [EXPANDED]

To be used with the #3 KineticEnergy option `Watson`. Indicates how the Watson pseudopotential term should be treated. The option `NONE` discards the term altogether. The option `EQUILIBRIUM` approximates the term by its equilibrium value (this requires the additional specification of the #3 Inertia or #3 InertiaTensorFile keywords). The option `EXPANDED` represents the term by an $n$-mode expansion (this requires the additional specification of the #3 Inertia or #3 InertiaTensorFile keywords and the #3 InvEffInertiaFiles keyword). Note that this option applies only for non-linear molecules (the Watson pseudopotential term is not defined for linear molecules).

#### #4 TaylorKeoGmatDiag

!Operator

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

To be used with the #3 KineticEnergy option `General` and a full second-order Taylor expansion of $G_{ij}$ (#4 KeoExpansionType `TAYLOR`). Specifies that the KEO is constructed from only the diagonal elements of the G matrix and that these contributions are restricted to the same coordinate as in the derivative part of the KEO. This leads to a one-mode coupled KEO with terms of the form $\partial/\partial Q_i \, G_{ii}(Q_i) \, \partial/\partial Q_i$.

#### #4 TaylorModeTruncApprox

!Operator

< *string* = N/A > [`AFTER2M`] [`AFTER3M`]

To be used with the #3 KineticEnergy option `General` and a KEO based on the full second-order expansion of $G_{ij}$. Truncates the KEO based on a second-order Taylor expansion of $G_{ij}$ such that only terms with up to 2-mode (`AFTER2M`) and 3-mode (`AFTER3M`) coupling remain.

#### #3 Name

!Operator

< *string* >

The name that will be used to label the operator.

#### #3 OperFile

!Operator

< *string* >

Provide the path to a file containing information on the operator in question. See section 40.
(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.)

## 17.2 Advanced keywords

#### #3 OperInput

!Operator

Allows defining an operator using the MidasCpp operator keywords. See 40.

## #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
```

# Chapter 18

# VSCF input options

## 18.1 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>^<exci-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 PrepareInitialTdAdgaDensity** !VSCF

< *integer vector* >

Prepare a TD-ADGA calculation by calculating an initial density for the wave function #2 AnalyzeStates. The initial density is calculated similar to the VSCF densities calculated in ADGA calculations, where the density is averaged over the n lowest modals for each mode.

**#3 Restart** !VSCF

Restart the calculation.

**#3 Threshold** !VSCF

< *real* = $100\epsilon_{\text{machine}}$ >

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

## 18.2 Advanced keywords

**#3 CheckOperModifiedInWfCalc** !VSCF

< *boolean* = TRUE > [`TRUE`] [`FALSE`]

Check whether the operator was modified in a previous wave-function calculation, and abort if so.

**#3 ForceOccPrint** !VSCF

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

MidasCpp will only print Occupation vectors in the VSCF summary, if the state has a number of modes less than 25. This keyword forces printing of the Occupation vectors even for cases where the number of modes is equal to or larger than 25.

**#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).
```

### #3 SaveModals

!VSCF

< *boolean* = TRUE > [TRUE] [FALSE]

Save modals to disk.

### #3 WriteEmbVibIntegrals

!VSCF

Write out integrals needed for later subspace approaches embedded in a mean-field treatment. (This keyword is only used in connection with the EmbVib python library.)

# Chapter 19

# 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.

## 19.1 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 EmbSys

!VCC

< *string vector* >

.

.

[< *string vector* >]

Defines the subsystems for an embedding VCC calculation. Each subsystem should be defined on a new line and formatted as 'NAME : <mode index vector>'. Names are only relevant if #3 FreezeEmbSys is set. Every mode in the system should belong to exactly one subsystem. Currently only supported for the VCC[2H2] method, and only excitation energies and transition strengths are implemented.

**Example for EmbSys**

```
#3 EmbSys
water1 : 0 1 2
water2 : 3 4 5
water3 : 6 7 8
water4 : 9 10 11
```

### #3 EnergyDiffMax

!VCI

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

Set maximum energy difference for ENERGY and OVERLAPANDENERGY targeting methods

### #3 FirstOvertones

!VCI

< *integer vector* >

Optimize selected first overtones with VCI.

### #3 FreezeEmbSys

!VCC

< *string vector* >

Reads a space separated list of subsystem names as defined in #3 EmbSys, which will be frozen. Frozen subsystems have their modal basis limited to 1 for each mode. This means that they will be exlcuded from the VCC calculation in the sence that no amplitudes or multipliers are associated with these, and no excitations involving these modes are calculated at any point. These modes will only perturb the VCC equations for the remaining systems as a VSCF mean-field.

**Example for FreezeEmbSys**

```
#3 FreezeEmbSys
water1 water2
```

**#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-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[`$n$`]`] [`VMP`$n$`[`$m$`]`]

[< *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* = 0.1 >

Set minimum squared overlap between target and solution vector for OVERLAP targeting method

#### #3 OverlapSumMin

!VCI

< *real* = 0.8 >

Set minimum overlap sum for OVERLAPSUM targeting method

#### #3 QpePreprocessor

Runs the quantum phase estimation preprocessor. Low-rank factorization of the Hamiltonian is enabled by default.

#### #4 LowRankThreshold

< *real* = 1e-08 >

Sets the low-rank factorization threshold. See #3 QpePreprocessor.

#### #4 OutputPrecision

< *integer* = 16 >

Sets output precision of QPE-Preprocessor output file. See #3 QpePreprocessor.

#### #4 TuckerThreshold

< *real* = 1e-10 >

Sets the threshold for the intermidiate Tucker decomposition of three-way and higher order tensors. See #3 QpePreprocessor.

#### #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 20).

#### #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] [BESTOVERLAP] [OVERLAP] [ENERGY] [OVERLAPANDENERGY] [OVERLAPSUM]

Set algorithm for targeting specific VCI states.

**BESTHITS** Choose the single state with the largest overlap with each target vector. Done in order of given target vectors.

**BESTOVERLAP** Choose the single state with the largest overlap with each target vector. Done in order of largest overlap.

**OVERLAP** Choose all states with at least a given overlap with each target vector.

**ENERGY** Choose all states which does not differ to much in energy from the state with the largest overlap.

**OVERLAPANDENERGY** Choose all states which satisfies the conditions of both OVERLAP and ENERGY.

**OVERLAPSUM** Choose best hits until a given total sum of overlaps is reached.

BESTHITS and BESTOVERLAP can differ in which solutions are chosen. BESTHITS will depend on the given order of the target vectors, whereas BESTOVERLAP will not. See also #3 OverlapMin, #3 OverlapSumMin, and #3 EnergyDiffMax.
(BESTHITS or OVERLAPSUM are often good methods.)

### #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. Note that it is **not** recommended to run state-specific VCC for excited VSCF states, e.g. by combining #3 UseAllVscf with #3 OccAllFund or #3 OccAllFirstOver. Instead the recommendation is to use VCC Response through #3 Rsp for calculation VCC excitation energies.

### #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.

### #3 WriteOper

Writes operator terms and matrix elements in human and machine readable formats to file. This includes the second quantized format that Midas Quantum accepts as input.
(Use to generate a midas operator and export it to a file which can be read by Midas Quantum.)

### #4 WriteOperScreen

< *real* = 0 >

Screens terms from the second quantized operator output.

## 19.2 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 $T^{(n)}_{\mathrm{CP},t} = C_t||\Delta\mathbf{t}^{(n)}||$.
(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 $T^{(n)}_{\mathrm{CP},e} = C_e T^{(n)}_{\mathrm{CP},t}$ (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 CheckOperModifiedInWfCalc

!VSCF

< *boolean* = TRUE > [`TRUE`] [`FALSE`]

Check whether the operator was modified in a previous wave-function calculation, and abort if so.

#### #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 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 ExtMcrMaxMcl

!VCC

< *integer* >

This keyword will limit the maximally allowed mode combination level one can obtain

when using the #3 ExtMcrModes keyword. This keyword takes an integer as argument and any mode combination containing more modes than this will not be included in the mode combination range.

### #3 ExtMcrModes

!VCC

< *string* **OR** *string* *integer* > [`<mode label>` **OR** `<mode label> <weigth>`]

.

.

[< *string* **OR** *string* *integer* >] [`<mode label>` **OR** `<mode label> <weigth>`]

This keyword will extend the maximum mode combination level (MCL) of the VCC[n] mode combination range (MCR) indicated by the #3 MaxExci keyword to include higher order mode combinations (MCs). For this purpose, a weight is assigned to each mode indicated under this keyword. MCs are added to the MCR if they satisfy $\sum_m w_m \leq n$, where $w_m$ is the weight assigned to each mode and $n$ is the "base" MCL provided under the #3 MaxExci keyword.
Only integer weights are accepted as input.
If only a mode label is provided per line the mode will be assigned a weight of 0.
Modes not listed under this keyword will be assigned the default weight of 1.

**Example for ExtMcrModes**

```
// Example input for H2O molecule creating a VCC[n] with a MCR
// containing all 1M MCs and the (Q0,Q1) and (Q1,Q2) MCs
#1 Vib
   #2 VCC
      #3 Method
         VCC[1] // "Base" MCL = 1
      #3 ExtMcrModes
               // Q0 is not given. Default weight is 1.
         Q1    // When no weigth is given the weight default to 0
         Q2 1 // Optional: Weight of 1 is default
// The resulting MCR calculation becomes:
// MC          | weight | included
// (Q0)        | 1      | yes
// (Q1)        | 0      | yes
// (Q2)        | 1      | yes
// (Q0,Q1)     | 1      | yes
// (Q0,Q2)     | 2      | no
// (Q1,Q2)     | 1      | yes
// (Q0,Q1,Q2)  | 2      | no
```

### #3 FranckCondon

!VCI

< *string vector* > [`OtherCalc=<name>`] [`Operator=<name>`]

Calculate Franck-Condon factors, $\langle \Psi_i^I | X | \Psi_j^J \rangle$, where $X$ is an operator. The wave functions must be obtained using VCI in two separate calculations, where one sets this keyword with `OtherCalc=<other_vci>` and possibly an operator (if no operator is given, simple overlaps are calculated).

(We recommend not to use an electronic transition dipole moment as operator unless one has taken utmost care this is calculated with consistent signs.)

**#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 (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-2 >

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

**#3 LambdaScan**
< *integerrealreal* >

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

!VCC !TENSOR

Pass control to the Laplace input driver for Laplace-decomposed energy denominators. See chapter 38.

### #3 Level2Solver

!VCI !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 MatRepFvciAnalysis

!VCC

Converts bra and ket of the ground state solution to FVCI parameterization and compare those FVCI vectors across other calculations with this keyword enabled. The FVCI-space vectors of the first #2 Vcc block containing this keyword will act as reference for the FVCI-space vectors. Only works for VCI methods and those of **??**. Note that none of the #2 Vcc calculations need be FVCI. If comparison with the actual FVCI vectors is desired, set the first #2 Vcc calculation to be VCI[M] by setting the #3 Method appropriately.
(Only use for small systems, since the FVCI-conversion quickly becomes computationally heavy. The limit is around 6 modes and 6 modals per mode.)

### #3 MaxBacksteps

!VCC

< *integer* = 10 >

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

### #3 MaxBacktracks

!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** !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>^<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.)

#### #4 LowRankFactorization

< *boolean* = True >

Use low-rank factorization. See #3 QpePreprocessor.

#### #4 SvdTruncationPolicy

< *string* = ErrorBound > [`ErrorBound`] [`SingularValue`]

Sets the criterion used to screen singular values. ErrorBound ensures that the maximum allowed error is set by #4 LowRankThreshold. SingularValue neglects all singular values smaller than #4 LowRankThreshold. See #3 QpePreprocessor.

#### #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 SaveModals

!VSCF

< *boolean* = TRUE > [`TRUE`] [`FALSE`]

Save modals to disk.

#### #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/TargetVectors or analysis/RspTargetVectors file. The format is the following. First one must provide the number of targets. For each target follows now a block defining the target. Each block must contain a line with the target index (starting from 0) and a number indicating the number of contributions to that target vector. Then follows a line for each contribution, each containing a floating-point coefficient and the occupation numbers for the contributing state. See short example below, outlining a target space of two target vectors for a three-mode, where the first target vector (index 0) has two contributions, and second target vector (index 1) has a single contribution. See also keywords #3 AllFundamentals, #3 AllFirstOvertones, #3 Fundamentals, and #3 FirstOvertones.
(Use this for targeting advanced states, not fundamentals and first overtones.)

**Example for TargetSpace**

```
2
0 2
0.5 0 0 1
0.5 0 1 0
1 1
1.0 0 1 2
```

### #3 TensorDecompInfo

!VCC !TENSOR

Read tensor-decomposition settings for recompressing the CP-VCC amplitudes. Passes control to tensor-decomposition input reader (see chapter 37).
(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 19.1 on page 136.
(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 37).
(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 MaxExci 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!)

## 19.3 Tables

Table 19.1: 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 MaxExci) 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 20

# Response input options

## 20.1 General keywords

**#4 AllFirstOvertones** !RSP

Target all first overtones.

**#4 AllFundamentals** !RSP

Target all fundamental excitations.

**#4 AnalysisEigenVectorElems** !RSP

< *integer* = 5 >

Set the number of elements to show in the eigenvector analysis. Elements are ordered by descending contribution.

**#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 DensityMatrixAnalysis

!VCC !VCI

< *boolean* = FALSE > [TRUE] [FALSE]

Construct Density matrix and diagonalize it to obtain natural modal occupation numbers and natural modals. In case of non-variational modals like VCC, the density matrix is symmetrized before diagonalization as is strongly recommended (see #4 SymDensNaMo). See also #4 ItNaMo.

### #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 EmbReducedTransform

!VCC !RSP

< *string* = OFF > [OFF] [ONLYA] [INCLUDEF]

Set reduced space transformation for embedding VCC eigenvalue equations. The embedding VCC eigenvalue equation will be solved for non-interacting subsystems first. The eigenvectors obtained in this way will then be used to transform the fully coupled problem into a smaller, approximate problem, from which approximate solutions to the fully interacting embedding VCC eigenvalues and eigenvectors may be obtained. This should reduce overall cost at the detriment of accuracy.

**ONLYA** Include only the subsystem local Jacobian contributions in the zeroth order picture.

**INCLUDEF** Include the subsystem local Jacobian- and F-matrix contributions in the zeroth order picture. Generally leads to a sizable increase in computational cost with insignificant increase in accuracy.

**OFF** Toggles the reduced transformation off

This keyword is only used in conjunction with #3 EmbSys.
(When using a reduced transformation, ONLYA is generally recommended.)

### #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** !RSP

< *real* = 1.e-6 >

Set residual threshold for solving VCC and VCI response equations.

**#4 ItNaMo** !VCC !VCI

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

Determine natural modals iteratively. That is, perform a wave function computation followed by determination of the natural modals. Restart the wave function computation for new wave function and new natural modals. See also #4 ItNaMoMaxIter and #4 ItNaMoThr.

**#4 ItNaMoMaxIter** !VCC !VCI

< *integer* = 1 >

Set maximum number of iterations for the generation of iterative natural modals. See also #4 ItNaMo.

### #4 ItNaMoThr

!VCC !VCI

< *real* = 1e-6 >

Set threshold for the generation of iterative natural modals. If convergence is reached, then for all modals $\phi^m_{p^m}$ indexed by $p^m$ for all modes it is satisfied that

$$\eta^m_{p^m} \left\| \phi^m_{p^m} - \tilde{\phi}^m_{p^m} \right\| < T_{\text{ItNaMo}} \tag{20.1}$$

where $\eta^m_{p^m}$ is the occupation number. See also #4 ItNaMo.

### #4 LanczosRsp

!RSP

< *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

!RSP

< *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.

**Example for RspFunc**

```
#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 SymDensNaMo

!VCC !VCI

< *boolean* = TRUE > [TRUE] [FALSE]

Do symmetrization of the density matrix before diagonalization. See #4 DensityMatrixAnalysis.

### #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.

**BESTHITS** Choose the single state with the largest overlap with each target vector. Done in order of given target vectors.

**BESTOVERLAP** Choose the single state with the largest overlap with each target vector. Done in order of largest overlap.

**OVERLAP** Choose all states with at least a given overlap with each target vector.

**ENERGY** Choose all states which does not differ to much in energy from the state with the largest overlap.

**OVERLAPANDENERGY** Choose all states which satisfies the conditions of both OVERLAP and ENERGY.

**OVERLAPSUM** Choose best hits until a given total sum of overlaps is reached.

BESTHITS and BESTOVERLAP can differ in which solutions are chosen. BESTHITS will depend on the given order of the target vectors, whereas BESTOVERLAP will not. See also #4 OverlapMin, #4 OverlapSumMin, and #4 EnergyDiffMax.
(BESTHITS or OVERLAPSUM are often good methods.)

## 20.2 Advanced keywords

**#4 AdiagVccRefEnergy** !RSP !VCC

< *boolean* = FALSE > [TRUE] [FALSE]

Subtract the VCC energy instead of the VSCF energy from the 0th order Jacobian Diagonal.

**#4 EnergyDiffMax** !RSP

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

Set maximum energy difference for ENERGY and OVERLAPANDENERGY targeting methods

**#4 OverlapMin** !RSP

< *real* = 0.1 >

Set minimum squared overlap between target and solution vector for OVERLAP targeting method

**#4 OverlapSumMin** !RSP

< *real* = 0.8 >

Set minimum overlap sum for OVERLAPSUM targeting method

**#4 VccIntegralsForEmb** !VCC !VCI

Write out integrals needed for later subspace approaches embedded in a correlated treatment of the environment.
(This keyword is only used in connection with the EmbVib python library.)

# Chapter 21

# TDH input options

## 21.1 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 CalcOneModeDensities** !TDH

Calculate the one-mode densities during the calculation. The the one-mode densities will be calculated in all interpolated points.

**#3 ExpectationValues** !TDH

Read in names of operators to calculate expectation values for. Use either of #4 Mop or #4 Internal.

**#4 Internal** !TDH

< *string vector* >
.
.
< *string vector* >

Read in names of operators to calculate expectation values for. Operators here are required to have the form of <Oper>(<ModeName>), fx Q^3(Q5). There can be multiple operators per line and properties are read until the next keyword.

**#4 Mop** !TDH

< *string vector* >
.
.
< *string vector* >

Read in names of #2 Operator 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 35). 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 OneModeDensityGridScal** !TDH
< *real* = 1.05 >

Scaling factor to extend the grid the one-mode densities are evaluated on. The scaling should match at least the fraction provided under the #3 SpaceSmoothening

**#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 disc. 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 SaveExptValContribsFromIndividualModes** !TDH

Print the contribution from each individual mode to expectation values (in addition to printing the expectation value itself). This is not enabled by default.

**#3 SpaceSmoothening** !TDH

< *real* = 0.05 >

Set the number of grid point to carry out a space-smoothening of the one-mode densities over by indicating a fraction of the basis. If 0.05 is used as input, the space-smoothening parameter $N_q$ will be set to 5% of the total number of grid points in the basis grid. If the fraction is set to 0 then no space-smoothening is carried out. The total number of grid points to smoothen over is $2 \times N_q + 1$.

**#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 TimeSmoothening** !TDH

< *integer* = 20 >

Set the number of historic time-steps to carry out a time-smoothening of the one-mode densities over. If the integer is set to 0 then no time-smoothening is carried out.

**#3 TwoModeDdensities** !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.)

**#3 WriteOneModeDensities** !TDH

Write the calculated one-mode densities to disc during the calculation.

## 21.2 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 FFTPadLevel** !TDH

< *integer* > [`Must be a power of 2`]

Round the number of points to the next power of two and multiply by this factor to get the number of points in the zero-padded signal.

**#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* = 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 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 22

# MCTDH input options

## 22.1 General keywords

**#3 ActiveSpace**
< *integer vector* >

Set the number of time-dependent modals for each mode.
(Using 6-10 time-dependent modals per mode usually gives converged results.)

**#3 Basis** !MCTDH
< *string* >

Name of the primitive one-mode basis.
(This must be the same basis as the one used in VSCF/VCI/MCTDH for generating the initial wave packet.)

**#3 CalcOneModeDensities** !MCTDH

Calculate the one-mode densities during the calculation. The one-mode densities will be calculated in all interpolated points.

**#3 ExpectationValues** !MCTDH

Read in names of operators to calculate expectation values for. Use either of #4 Mop or #4 Internal.

**#4 Internal** !MCTDH
< *string vector* >
.
.
< *string vector* >

Read in names of operators to calculate expectation values for. Operators here are required to have the form of <Oper>(<ModeName>), fx Q^3(Q5). There can be multiple operators per line and properties are read until the next keyword.

**#4 Mop** !MCTDH

< *string vector* >

.

.

< *string vector* >

Read in names of #2 Operator to calculate expectation values for. There can be multiple operators per line and properties are read until the next keyword.

**#3 ImagTime** !MCTDH

Do imaginary-time propagation in order to obtain the ground-state energy and wave function.

**#3 InitialWF** !MCTDH

< *string vector* > [`VSCF <vscf-name>`] [`VCI <vci-name>`] [`MCTDH <mctdh-name>`]
[< *string vector* >] [`APPLYOPERATORS <operators>`]

Policy for initializing the MCTDH wave function. This can be done from VSCF, VCI, or MCTDH calculations defined in the same input file. The APPLYOPERATORS options can be set if an operator (e.g. a dipole) should be applied to the initial wave packet before propagation. If multiple operators are given, individual propagations are performed for each operator.

**#3 Integrator** !MCTDH

Define integration scheme. Note that this keyword is mandatory!

**#4 OdeInfo** !MCTDH

Pass control to the ODE input reader (see section 35).

**#4 Scheme** !MCTDH

< *string* = VMF > [`VMF`]

Choose integration scheme. For now, only variable mean field (VMF) is implemented.

**#3 IoLevel** !MCTDH

< *integer* = 1 >

Set IO level for MCTDH calculation.

**#3 LimitModalBasis** !MCTDH

< *integer vector* >

Limit the number of primitive basis functions for each mode.

**#3 Method**

< *string* > [`MCTDH[n]`] [`MCTDH`]

Set the MCTDH method to use. This can be either truncated MCTDH[n] or full MCTDH. Input like MCTDH[n,D] (with n being an integer) for natural-modal propagation is also supported (the default constraint is fully variational MCTDH[n,V]).

**#3 Name** !MCTDH

< *string* = mctdh >

Assign a name to the MCTDH calculation.
(Assign a unique name to each calculation if there are multiple MCTDH input blocks in the input file.)

**#3 OneModeDensityGridScal** !MCTDH

< *real* = 1.05 >

Scaling factor to extend the grid the one-mode densities are evaluated on. The scaling should match at least the fraction provided under the #3 SpaceSmoothening

**#3 Oper** !MCTDH

< *string* >

Name of the operator the MCTDH wave-packet propagation is performed on.

**#3 Properties** !MCTDH

< *string vector* > [`norm`] [`energy`] [`autocorr`] [`halftimeautocorr`] [`refcoef`] [`densanalysis`] [`statepopulations`]
.
.
< *string vector* > [`norm`] [`energy`] [`autocorr`] [`halftimeautocorr`] [`refcoef`] [`densanalysis`] [`statepopulations`]

Read in the properties that should be calculated and saved to disc. There can be multiple properties per line and properties are read until the next keyword. The properties `autocorr` and `halftimeautocorr` are computed using the approximation described in Sec. 9.2.

**#3 Regularization**

< *string vector* = STANDARD 1.e-8 > [`NONE`] [`STANDARD <epsilon>`]

Set the type of regularization and the value of the regularization parameter.
(Use STANDARD regularization. The default epsilon is also good for most purposes.)

**#3 SaveExptValContribsFromIndividualModes** !MCTDH

Print the contribution from each individual mode to expectation values (in addition to printing the expectation value itself). This is not enabled by default.

**#3 SpaceSmoothening** !MCTDH

< *real* = 0.05 >

Set the number of grid point to carry out a space-smoothening of the one-mode densities over by indicating a fraction of the basis. If 0.05 is used as input, the space-smoothening parameter $N_q$ will be set to 5% of the total number of grid points in the basis grid. If the fraction is set to 0 then no space-smoothening is carried out. The total number of grid points to smoothen over is $2 \times N_q + 1$.

**#3 Spectra** !MCTDH

< *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 TimeSmoothening** !MCTDH

< *integer* = 20 >

Set the number of historic time-steps to carry out a time-smoothening of the one-mode densities over. If the integer is set to 0 then no time-smoothening is carried out.

**#3 WriteOneModeDensities** !MCTDH

Write the calculated one-mode densities to disc during the calculation.

## 22.2 Advanced keywords

**#3 AnalysisDir**

< *string* = <maindir/analysis> >

Set a separate analysis directory for MCTDH where all properties are saved. The default is to use the global MidasCpp analysis directory.

**#3 ConstraintOperator**

< *string vector* = VARIATIONAL ZERO > [`{NONE}`] [`DENSITYMATRIX`] [`VARIATIONAL}`] [`{ZERO}`] [`ONEMODEH}`]

For MCTDH[n], set the type of constraint operator. The first key is the type of non-redundant constraints and the second is the type of redundant constraints.
(Use DENSITYMATRIX or VARIATIONAL for the non-redundant constraints and ZERO for the redundant. Note that the non-redundant constraint can also be set via the #3 Method keyword.)

**#3 Exact**

If using the full modal basis for all modes (i.e. $n^m = N^m$), some steps in the derivative calculation can be optimized. This is done by setting this keyword. Note that if only works if the active space is set to match the full number of primitive basis functions.
(Set this keyword if using the exact propagation method.)

**#3 ExtMcrMaxMcl** !MCTDH

< *integer* >

This keyword will limit the maximally allowed mode combination level one can obtain when using the #3 ExtMcrModes keyword. This keyword takes an integer as argument and any mode combination containing more modes than this will not be included in the mode combination range.

**#3 ExtMcrModes** !MCTDH

< *string* **OR** *string* *integer* > [`<mode label>` **OR** `<mode label> <weigth>`]
.
.
[< *string* **OR** *string* *integer* >] [`<mode label>` **OR** `<mode label> <weigth>`]

Extend the maximum mode combination level (MCL) to run MR-MCTDH[$n$] with the input mode labels as MR modes. MCs are added to the MCR if they satisfy $\sum_m w_m \leq n$, where $w_m$ is the weight assigned to each mode and $n$ is the "base" MCL provided for MCTDH[n] under the #3 Method keyword.
For non-adiabatic calculations, the electronic DOF is added automatically with a weight of 0.
Only integer weights are accepted as input.
If only a mode label is provided per line the mode will be assigned a weight of 0.
Modes not listed under this keyword will be assigned the default weight of 1.

**Example for ExtMcrModes**

```
// Example input for H2O molecule creating a MR-MCTDH[n] with a MCR
// containing all 1M MCs and the (Q0,Q1) and (Q1,Q2) MCs
#1 Vib
   #2 MCTDH
      #3 Method
         MCTDH[1] // "Base" MCL = 1
      #3 ExtMcrModes
                  // Q0 is not given. Default weight is 1.
         Q1 0     // When no weigth is given the weight default to 0
         Q2 1     // Optional: Weight of 1 is default
// The resulting MCR calculation becomes:
// MC          | weight | included
// (Q0)        | 1      | yes
// (Q1)        | 0      | yes
// (Q2)        | 1      | yes
// (Q0,Q1)     | 1      | yes
// (Q0,Q2)     | 2      | no
// (Q1,Q2)     | 1      | yes
// (Q0,Q1,Q2)  | 2      | no
```

(Use MR-MCTDH if one (or a few) degrees of freedom are special, i.e. expected to govern the quantum dynamics.)

### #3 FFTPadLevel

!MCTDH

< *integer* > [Must be a power of 2]

Round the number of points to the next power of two and multiply by this factor to get the number of points in the zero-padded signal.

### #3 GOptRegularization

< *string vector* = STANDARD 1.e-8 > [NONE] [STANDARD <epsilon>] [SVD <epsilon>] [XSVD <epsilon>] [TIKHONOV <epsilon>]

Set the type of regularization for solving linear equations in determining the non-redundant constraint-operator matrix elements.
(Use STANDARD or XSVD regularization. The default epsilon is also good for most purposes.)

### #3 LifeTime

!MCTDH

< *real* >

Set a lifetime broadening of the spectra. Input is in atomic units.

### #3 NoAutocorrConvolution

!MCTDH

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 Projector** !MCTDH

< *string* = MODIFIED > [`MODIFIED`] [`SIMPLE`]

Type of secondary-space projector. The simple projector is given by $\mathbf{Q} = \mathbf{1} - \mathbf{V}_\mathrm{A}\mathbf{V}_\mathrm{A}^\dagger$. The modified projector is given by $\mathbf{Q} = \mathbf{1} - \mathbf{V}_\mathrm{A}\mathbf{S}^{-1}\mathbf{V}_\mathrm{A}^\dagger$, where $\mathbf{S}$ is the modal overlap matrix. The modified projector remains a true projector even if the basis is not exactly orthogonal. (Use the `MODIFIED` projector.)

**#3 SaveAcceptedWfs**

Write the wave function to disc after all accepted integration steps.

**#3 SaveInterpolatedWfs**

Write the wave function to disc at all interpolated points.

**#3 SpectralBasis** !MCTDH

[< *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.
(Unless a special intermediate basis (e.g. the eigenbasis of the position operator) is requested, this keyword can be omitted.)

**#3 SpectrumEnergyShift** !MCTDH

< *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** !MCTDH

< *real vector* >

Read two numbers. Only output the spectrum between these two frequencies.

**#3 SpectrumOutputScreening** !MCTDH

< *real* >

Only write the parts of the spectrum with intensity larger than this threshold.

# Chapter 23

# TDVCC input options

## 23.1 General keywords

**#3 Basis** !TDVCC

< *string* >

Name of the primitive one-mode basis.
(This must be the same basis as the one used in VSCF/VCC for generating the initial wave packet.)

**#3 CalcOneModeDensities** !TDVCC !TDMVCC

Calculate the one-mode densities during the calculation. The the one-mode densities will be calculated in all interpolated points.

**#3 CorrMethod** !TDVCC

< *string* = VCC > [VCC] [VCI] [EXTVCC] [TDMVCC]

Set the wave-function parameterization
(Note that TDMVCC is only implemented with the MATREP and VCC2H2 keywords under #3 Transformer as of now.)

**#3 DataWriteInterval** !TDVCC

< *real* = 0.0 >

The maximum size of time-intervals for how often Midas will write data values (fx. Energy, different norms, expvals, etc.) to file during the tdvcc propagation. This is in atomic time units.

**#3 ExpectationValues** !TDVCC

Read in names of operators to calculate expectation values for. Use either of #4 Mop or #4 Internal.

**#4 Internal** !TDVCC

< *string vector* >

.
.
< *string vector* >

Read in names of operators to calculate expectation values for. Operators here are required to have the form of <Oper>(<ModeName>), fx Q^3(Q5). There can be multiple operators per line and properties are read until the next keyword.

**#4 Mop** !TDVCC

< *string vector* >
.
.
< *string vector* >

Read in names of #2 Operator to calculate expectation values for. There can be multiple operators per line and properties are read until the next keyword.

**#3 ImagTime** !TDVCC

Do imaginary-time propagation in order to obtain the ground-state energy and wave function.

**#3 ImagTimeHaultThr** !TDVCC

< *real* = 0 >

Stop imaginary-time propagation when the derivative norm becomes less than this value.

**#3 InitState** !TDVCC

< *string* = VSCF > [VSCF] [VCC]

Choose to start from either a VSCF (set using #3 ModalBasisFromVscf) or VCC (set using #3 VccGs) wave packet.

**#3 Integrator** !TDVCC

Pass control to the ODE input reader (see section 35).

**#3 IoLevel** !TDVCC

< *integer* = 1 >

Set IO level for TDVCC calculation.

**#3 KappaReset** !TDMVCC

< *string* = NEVER > [NEVER] [ALWAYS] [THRESHOLD]

[< *real* = -1 >]

How to reset $\kappa$ matrices in TDMVCC with single or double exponential modal parametrization (see #3 ModalParametrization).

**NEVER** Never reset the $\kappa$ matrices. May cause near-singularities in the modal equations of motions, which result in temporarily small integrator steps.

**ALWAYS** Reset the $\kappa$ matrices after every integrator step. This is almost never necessary.

**THRESHOLD** Occasionally reset the $\kappa$ matrices based on a user-specified threshold. Thresholds around 0.1 work well.

(Use the threshold based reset.)

**#3 LimitModalBasis** !TDVCC

< *integer vector* = 6 >

Limit the number of primitive basis functions for each mode.

**#3 ModalBasisFromVscf** !TDVCC

< *string* = vscf >

Select the VSCF calculation to obtain modals from.
(If only one VSCF calculation is defined in the input file, TDVCC will automatically use that one.)

**#3 ModalParametrization** !TDMVCC

< *string* = ACT_SPLIT_LINEAR > [ACT_SPLIT_LINEAR] [ACT_LINEAR] [ACT_POLAR] [ALL_EXP] [ALL_DOUBLE_EXP] [ACT_ORTHOG_LINEAR]

Choose how to parametrize the time-dependent modals. The options with prefix `ALL` explicitly represent and propagate all (active and secondary) modals, while the options with prefix `ACT` only represent and propagate the active modals. Current option include:

**ACT_SPLIT_LINEAR** A split unitary/non-unitary parametrization. Fully bivariational, numerically stable, and convergent to MCTDH.

**ACT_LINEAR** The original linear modal parametrization.

**ACT_POLAR** Restricted polar parametrization. May enhance numerical stability at the cost of some non-variationality.

**ALL_EXP** Single exponential parametrization (see also #3 KappaReset).

**ALL_DOUBLE_EXP** Double exponential parametrization (see also #3 KappaReset).

**CT_ORTHOG_LINEAR** Orthonormal linearly parametrized modals. This option breaks strict convergence to the MCTDH limit.

### #3 Name

!TDVCC

< *string* = tdvcc >

Assign a name to the TDVCC calculation.
(Assign a unique name to each calculation if there are multiple TDVCC input blocks in the input file.)

### #3 OneModeDensityGridScal

!TDVCC !TDMVCC

< *real* = 1.05 >

Scaling factor to extend the grid the one-mode densities are evaluated on. The scaling should match at least the fraction provided under the #3 SpaceSmoothening

### #3 Oper

!TDVCC

< *string* >

Name of the operator the TDVCC wave-packet propagation is performed on.

### #3 Properties

!TDVCC

< *string vector* > [`energy`] [`phase`] [`autocorr`]
.
.
< *string vector* > [`energy`] [`phase`] [`autocorr`]

Read in the properties that should be calculated and saved to disc. There can be multiple properties per line and properties are read until the next keyword.

### #3 Pulse

!TDVCC

< *string* >
< *real vector* >

Add a time-dependent Gaussian pulse, $c(t)\hat{H}_1$, where $\hat{H}_1$ is a time-independent operator, and $c(t) = A\exp(-(t-t_0)^2/2\sigma^2)\cos(\omega(t-t_0)-\phi)$ is the time-dependent Gaussian profile. The first line must be the name of the $\hat{H}_1$ operator. The second line must be the 5 real numbers defining the Gaussian: amplitude $A$, peak position $t_0$, width/standard deviation $\sigma$, frequency $\omega$, and phase $\phi$ (all in atomic units). Add additional Pulse keyword if more than one pulse is wanted.
(The effective Hamiltonian will be $\hat{H}_0+c(t)\hat{H}_1$, where $\hat{H}_0$ is the operator defined by #3 Oper.)

**Example for Pulse**

```
#3 Pulse
   h_pulse              // name of a '#2 Operator' defined elsewhere
   0.01 100 10 7.0e-3 0 // A t0 sigma omega phi
   // Results in:
   // h_pulse * 0.01 * exp(-(t-100)^2/(2*10^2)) * cos(7.0e-3*(t-100) - 0)
```

**#3 SaveDataAtIntegratorSteps** !TDVCC

Midas will save the data values (fx. Energy, different norms, expvals, etc.) asked for in the tdvcc computation for each integrator time step, i.e. not equidistant points.

**#3 SaveDataAtInterpolatedPoints** !TDVCC

Midas will save the data values (fx. Energy, different norms, expvals, etc.) asked for in the tdvcc computation for each interpolated point, i.e. equidistant points.
(This can help limit the amount of data. It also makes it easier to calculate differences between different calculations.)

**#3 SpaceSmoothening** !TDVCC !TDMVCC
< *real* = 0.05 >

Set the number of grid point to carry out a space-smoothening of the one-mode densities over by indicating a fraction of the basis. If 0.05 is used as input, the space-smoothening parameter $N_q$ will be set to 5% of the total number of grid points in the basis grid. If the fraction is set to 0 then no space-smoothening is carried out. The total number of grid points to smoothen over is $2 \times N_q + 1$.

**#3 TdModalBasis** !TDMVCC
< *integer vector* = 1 >

Limit the number of time-dependent basis functions for each mode for #3 CorrMethod TDMVCC keyword.

**#3 TimeSmoothening** !TDVCC !TDMVCC
< *integer* = 20 >

Set the number of historic time-steps to carry out a time-smoothening of the one-mode densities over. If the integer is set to 0 then no time-smoothening is carried out.

**#3 VccGs** !TDVCC
< *string* = vcc >

Assign a VCC calculation which can be used for analysis or to start the propagation from (see #3 InitState).

**#3 WriteOneModeDensities** !TDVCC !TDMVCC

Write the calculated one-mode densities to disc during the calculation.

## 23.2 Advanced keywords

### #3 CompareCMatrices

!TDMVCC

Compare different implementations of the C matrix appearing in the constraint equations for TDMVCC. A developer's option.

### #3 CompareFvciVecs

!TDVCC

Convert bra and ket of the method to FVCI parameterization and save those FVCI vectors for later comparison with similar FVCI vectors of other #2 TdVcc calculations. Comparisons consist of distance and angle in FVCI-space; tables will be written to files. The FVCI vectors will be saved at the (equidistant) points specified by #<N> OutputPoints (which must be the same for all the #2 TdVcc calculations). At least two #2 TdVcc calculations must be performed and contain this keyword for it to be effective. The FVCI-space vectors of the first #2 TdVcc block containing this keyword will act as reference for the FVCI-space vectors. Note that none of the #2 TdVcc calculations need be FVCI. If comparison with the actual FVCI vectors is desired, set the first #2 TdVcc calculation to be TDVCI[M] by setting the #3 CorrMethod, #3 MaxExci and #3 Transformer appropriately.
(Only use for small systems, since the FVCI-conversion quickly becomes computationally heavy. The limit is around 6 modes and 6 modals per mode. To avoid interpolation errors for the first output points, set #<N> InitialStepSize to be less than the time for the first output point.)

**Example for CompareFvciVecs**

```
// Basic setup for comparing TDVCC[2] ket/bra against
// TDFVCI ket/bra (for a 3-mode system) at every 1 a.u.
#2 Tdvcc
   #3 Name
      tdfvci
   #3 CorrMethod
      vci
   #3 MaxExci
      3
   #3 Transformer
      matrep
   #3 CompareFvciVecs
   #3 Integrator
      #4 TimeInterval
         0.0 100.0 // <-- must be identical across all calcs.
      #4 OutputPoints
         101       // <-- must be identical across all calcs.
      #4 InitialStepSize
         0.1
   // ...
#2 Tdvcc
   #3 CorrMethod
      tdvcc2
   #3 MaxExci
      2
   #3 Transformer
      vcc2h2
   #3 CompareFvciVecs
   #3 Integrator
      #4 TimeInterval
         0.0 100.0 // <-- must be identical across all calcs.
      #4 OutputPoints
         101       // <-- must be identical across all calcs.
      #4 InitialStepSize
         0.1
   // ...
```

### #3 ExtMcrMaxMcl

!TDVCC !TDMVCC

< *integer* >

This keyword will limit the maximally allowed mode combination level one can obtain when using the #3 ExtMcrModes keyword. This keyword takes an integer as argument and any mode combination containing more modes than this will not be included in the mode combination range.

### #3 ExtMcrModes

!TDVCC !TDMVCC

< *string* **OR** *string* *integer* > [`<mode label>` **OR** `<mode label> <weigth>`]

.

.

[< *string* **OR** *string* *integer* >] [`<mode label>` **OR** `<mode label> <weigth>`]

This keyword will extend the maximum mode combination level (MCL) of the TDVCC[n]

mode combination range (MCR) indicated by the #3 MaxExci keyword to include higher order mode combinations (MCs). For this purpose, a weight is assigned to each mode indicated under this keyword. MCs are added to the MCR if they satisfy $\sum_m w_m \leq n$, where $w_m$ is the weight assigned to each mode and $n$ is the "base" MCL provided for TDVCC[n] under the #3 MaxExci keyword.
For non-adiabatic calculations, the electronic DOF is added automatically with a weight of 0.
Only integer weights are accepted as input.
If only a mode label is provided per line the mode will be assigned a weight of 0.
Modes not listed under this keyword will be assigned the default weight of 1.

**Example for ExtMcrModes**

```
// Example input for H2O molecule creating a TDVCC[n] with a MCR
// containing all 1M MCs and the (Q0,Q1) and (Q1,Q2) MCs
#1 Vib
   #2 TDVCC
      #3 CorrMethod
         VCC
      #3 MaxExci
         1    // The "base" MCL
      #3 ExtMcrModes
               // Q0 is not given. Default weight is 1.
         Q1    // When no weigth is given the weight default to 0
         Q2 1 // Optional: Weight of 1 is default
// The resulting MCR calculation becomes:
// MC          | weight | included
// (Q0)        | 1      | yes
// (Q1)        | 0      | yes
// (Q2)        | 1      | yes
// (Q0,Q1)     | 1      | yes
// (Q0,Q2)     | 2      | no
// (Q1,Q2)     | 1      | yes
// (Q0,Q1,Q2) | 2      | no
```

### #3 FFTPadLevel

!TDVCC

< *integer* > [Must be a power of 2]

Round the number of points to the next power of two and multiply by this factor to get the number of points in the zero-padded signal.

### #3 FullActiveBasis

!TDMVCC

Enables a slightly faster calculation in the limit of no secondary space, i.e. requires #3 LimitModalBasis and #3 TdModalBasis to be equal.

### #3 MaxExci

!TDVCC

< *integer* = 2 >

Maximum excitation level for the correlated wave-function calculation.

**#3 ModalStatistics** !TDVCC

< *string vector* > [nonbiortho] [nonorthoket] [nonorthobra]

.

.

< *string vector* > [nonbiortho] [nonorthoket] [nonorthobra]

Read in the modal statistics (norms and difference norms) that should be calculated and saved to disc. There can be multiple properties per line and properties are read until the next keyword. Only relevant for TDMVCC.

**#3 NoAutocorrConvolution** !TDVCC

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 NormalizeSpectrum** !TDVCC

Normalizes the spectrum so that the highest peak has a magnitude of 1.

**#3 PrintParamsToFile** !TDVCC !TDMVCC

Tells midas to save the TDVCC/TDMVCC parameters to files during the computation. Creates a file for each #<N> OutputPoints.
(If large system with many output points, might generate a lot of files. But useful if you want to be able to restart a computation fx. in event of a crash.)

**#3 PrintoutInterval** !TDVCC

< *real* = 0.0 >

Print status of propagation after this time interval. If 0.0 is given, the status will be printed 10 (equidistant) times during the propagation.

**#3 Projector** !TDMVCC

< *string* = MODIFIED > [MODIFIED] [SIMPLE]

Type of secondary-space projector (only relevant for the TDMVCC option under #3 CorrMethod). The simple projector is given by $\mathbf{Q} = \mathbf{1} - \mathbf{U}_\mathrm{A}\mathbf{W}_\mathrm{A}$ or $\mathbf{Q} = \mathbf{1} - \mathbf{V}_\mathrm{A}\mathbf{V}_\mathrm{A}^\dagger$ depending on #3 ModalParametrization. The modified projector is given by $\mathbf{Q} = \mathbf{1} - \mathbf{U}_\mathrm{A}\mathbf{S}^{-1}\mathbf{W}_\mathrm{A}$ or $\mathbf{Q} = \mathbf{1} - \mathbf{V}_\mathrm{A}\mathbf{S}^{-1}\mathbf{V}_\mathrm{A}^\dagger$, where $\mathbf{S}$ is the modal overlap matrix. The modified projector remains a true projector even if the basis is not exactly (bi)orthogonal.
(Use the MODIFIED projector.)

**#3 RestartFromFile** !TDVCC !TDMVCC

Restart TDVCC or TDMVCC calculation from parameter file.

**#4 InitStateForAutocorrPath** !TDMVCC

< *string* >

Tell midas the path and file, to where the parameters you want for the initial state in the autocorrelation function are located.

**#4 RestartPath** !TDVCC !TDMVCC

< *string* >

Tell midas the path and file, to where the parameters you want to restart from are located.

**#3 SpectrumEnergyShift** !TDVCC

< *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** !TDVCC

< *real vector* >

Read two numbers. Only output the spectrum between these two frequencies.

**#3 SpectrumOutputScreening** !TDVCC

< *real* >

Only write the parts of the spectrum with intensity larger than this threshold.

**#3 Statistics** !TDVCC

< *string vector* > [`fvcinorm2`] [`norm2`] [`dnorm2init`] [`dnorm2vccgs`]

.

.

< *string vector* > [`fvcinorm2`] [`norm2`] [`dnorm2init`] [`dnorm2vccgs`]

Read in the statistics (norms and difference norms) that should be calculated and saved to disc. There can be multiple properties per line and properties are read until the next keyword.

**#3 Tdmvcc2Autocorr** !TDMVCC

Specify settings for the TDMVCC[2] autocorrelation function. Only has an effect if autocorr is enabled under #3 Properties and if VCC2H2 is chosen under #3 Transformer.
(The FULL option gets very computationally heavy around 12 modes. Consider the experimental screening option above this system size.)

**#4 Algorithm** !TDMVCC

< *string* = Full > [`Full`] [`Truncate`] [`Screen`]

(The FULL option gets very computationally heavy around 12 modes, consider truncating or screening around/above this system size.)

**#4 CalcTypeA** !TDMVCC

< *boolean* = true >

Whether to calculate type A autocorrelation function.

**#4 CalcTypeB** !TDMVCC

< *boolean* = false >

Whether to calculate type B autocorrelation function.

**#4 DeltaThres** !TDMVCC

< *real* = 10 >

Multiplicative distance between thresholds.

**#4 MaxLevel** !TDMVCC

< *integer* = −1 >

The maximum level included in recursive sum when using truncated algorithm. The default setting (−1) includes all levels and is equivalent to using the full algorithm under #4 Algorithm.

**#4 NumThres** !TDMVCC

< *integer* = 5 >

Number of thresholds.

**#4 ScreenThres** !TDMVCC

< *real* = $1.0 \cdot 10^{-10}$ >

Screening threshold for screening algorithm.

**#3 TimeIt** !TDVCC

Output detailed timings during the propagation.

**#3 Transformer** !TDVCC

< *string* = VCC2H2 > [`VCC2H2`] [`MATREP`] [`GENERAL`]

Select the algorithm for calculating the time derivative. VCC2H2 is the fast implementation of TDVCC/TDVCI which requires #3 MaxExci to be 2. MATREP is for full-space reference calculations. GENERAL is an efficient implementation for general excitation TDVCC/TDVCI.
(MATREP is computationally heavy for all but the smallest systems, due to working with full-space vectors; the limit is around 6 modes and 6 modals per mode.)

# Chapter 24

# 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.

## 24.1 General keywords

**#1 Pes**

Read Pes input until next level #1 input key.

**#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 #2 ItResEnThr keyword.

**#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 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.

**Example for AdgaInfo**

```
#2 AdgaInfo
   2        // Specifies up-to 2 mode cuts.
   myvscf   // Use vscf named 'myvscf'.
```

### #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 AnalyzeStates

!ADGA

< *integer* = 4 **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 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 CalcDensMCL

!ADGA

< *integer* = 1 > [MCL]

Do a VSCF calculation to obtain the vibrational density after each ADGA iteration up

to and including the specified mode combination level, 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 UpdateDensMCL 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 independent components, which will be added to the list of calculated properties. References values and the moment of inertia for the reference geometry will also be written to files. For non-linear molecules, there six independent components (XX, XY, XZ, YY, YZ, ZZ) as well as the trace are fitted. For linear molecules, the single independent component is fitted.

### #2 CalcExtraPotential

!ADGA !Static

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

Calculate the extrapotential for each displaced geometry generated in a grid-based potential energy surface calculation in curvilinear coordinates. Produce a fitted representation of the extrapotential as well.

### #2 CalcGMatrix

!ADGA !Static

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

Calculate the components of the G matrix for each displaced geometry generated in a grid-based potential energy surface calculation in curvilinear coordinates. Produce fitted representations of these components as well. Only the upper triangle elements of the symmetric matrix is generated.

### #2 CoordTrafoScript

< *string* >

Takes the name of a script, which is used to run the TANA program and convert the output into a readable format for MidasCpp. This script should be placed in the `InterfaceFiles` directory. Note that the keyword #3 KineticEnergy will need to be specified as well if a kinetic energy operator based on polyspherical coordinates is to be used.

### #2 CoordinateType

< *string* = Normal > [`Cartesian`] [`Normal`] [`LocalBondStretch`] [`Curvilinear`]

Set the coordinate type used in the PES construction. Note that setting the curvilinear option, requires that the #2 CoordTrafoScript keyword is also set.

### #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 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 DoAdgaExtensionMCL

!ADGA

< *integer* = 1 > [MCL]

Allows the ADGA to extend the potential grid boundaries at each iteration for mode combination levels up to and including the input level.

### #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 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 ExtendedGridSettings

!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 ExtendedGridSettings**

```
#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 25.

### #2 GenHessian

!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

!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 IoLevel

!ADGA !Static

< *integer* = 0 >

Indicates the IoLevel of the PES module.

### #2 ItDensThr

!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 ItLinVDensThr** !TDADGA

$< real = 1.0 \cdot 10^{-4} >$

Convergence threshold $\epsilon_{\text{lin}}$ for the TD-ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed error of the integral over wave function density times linearized potential. The intervals with linear error of the integrals that falls below this threshold are considered converged in the ADGA.

**#2 ItNormResEnThr** !ADGA

$< real = 1.0 \cdot 10^{-6} >$

Convergence threshold $\epsilon_{\text{abs}}$ for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed absolute error of integrals over norm-like quantities

$$\sqrt{\int_i \left(\sqrt{\rho^{\text{ave}}_{\text{curr}}(Q_m)} V^{\vec{m}_n}_{\text{curr}}(Q_m) - \sqrt{\rho^{\text{ave}}_{\text{prev}}(Q_m)} V^{\vec{m}_n}_{\text{prev}}(Q_m)\right)^2 dQ_m} < \epsilon_{\text{abs}}$$
$$\wedge \quad \sqrt{\int_i \left(\sqrt{\rho^{\text{ave}}_{\text{curr}}(Q_m)} V^{\vec{m}_n}_{\text{curr}}(Q_m)\right)^2 dQ_m} < \epsilon_{\text{abs}},$$

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

$< real = 1.0 \cdot 10^{-2} >$

Convergence threshold $\epsilon_{\text{rel}}$ for the ADGA, when constructing potential energy or molecular property surfaces. This threshold sets the allowed relative error of integrals over norm-like quantity

$$\frac{\sqrt{\int_i \left(\sqrt{\rho^{\text{ave}}_{\text{curr}}(Q_m)} V^{\vec{m}_n}_{\text{curr}}(Q_m) - \sqrt{\rho^{\text{ave}}_{\text{prev}}(Q_m)} V^{\vec{m}_n}_{\text{prev}}(Q_m)\right)^2 dQ_m}}{\sqrt{\int_i \left(\sqrt{\rho^{\text{ave}}_{\text{curr}}(Q_m)} V^{\vec{m}_n}_{\text{curr}}(Q_m)\right)^2 dQ_m}} < \epsilon_{\text{rel}},$$

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 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 ItResDensThr** !ADGA

$< real = 1.0 \cdot 10^{-3} >$

Convergence threshold $\epsilon_\rho$ 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 ItResEnThr** !ADGA

$< real = 1.0 \cdot 10^{-6} >$

Convergence threshold $\epsilon_{\text{abs}}$ 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 product that falls below this threshold are considered converged in the ADGA.

**#2 ItVDensThr** !ADGA

$< real = 1.0 \cdot 10^{-2} >$

Convergence threshold $\epsilon_{\text{rel}}$ 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 product 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 integer >*

.

.

[*< label 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.

**Example for MultiLevelPes**

```
#2 MULTILEVELPES
 generic_sp_ccsdt 1
 generic_sp_ccsd 2
 generic_sp_rimp2 4
```

### #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 NoAdgaPlotData

!ADGA

Indicates that no MCL-dimensional potential cuts should be plotted and stored on file.

### #2 NoMaxVDensThr

!TDADGA

Disable the max convergence check for the TD-ADGA, when constructing potential energy or molecular property surfaces.

### #2 NormalCoordinateThreshold

!ADGA !Static

< *real* = $10^{-7}$ >

Sets the threshold for the check on normal coordinate ortho-normality.

### #2 Nthreads

!ADGA !Static

< *integer* = `--numthreads` **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.mmol` file. If this keyword is inactive a common scaling factor of 1.0 is assigned.

### #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 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 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 PlotAdgaDataMCL

!ADGA

< *integer* = 1 > [MCL]

Indicates that MCL-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 PscMaxGridBounds

< *real vector* = 0.0 −0.02 −0.02 >

This keyword takes a vector of three numbers, which can be both negative and positive in order to modify the displacement scale factor for curvilinear valence coordinates set by the Curvilinear option under the #2 CoordinateType keyword. The numbers specify the respective scaling factor for bond length, angular and dihedral coordinates. Negative numbers will in this context restrict the maximally allowed displacements to be smaller than the natural range of internal coordinates, while positive numbers will ensure a larger intially spanned space but with the risk of displacing through a point of periodicity.

### #2 SaveDispGeom

!ADGA !Static

< *string* = FALSE > [TRUE] [FALSE]

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. See also #2 SaveDispGeomFormat and #2 SaveDispGeomPrecision.
(The recommendation is to NOT use this keyword, as it will introduce concurrency and synchronization between otherwise embarrassingly parallel SPCs. This will increase computation times, especially in the case where the calculation is comprised of many cheap SPCs. The keyword can be used for debugging purposes, or for creating/saving smaller SPC datasets.)

### #2 SaveDispGeomFormat

!ADGA !Static

< *string* = SCIENTIFIC > [SCIENTIFIC] [FIXED]

Print displaced structures in scientific or fixed-point notation. See also #2 SaveDispGeom

and #2 SaveDispGeomPrecision.

**#2 SaveDispGeomPrecision** !ADGA !Static

< *integer* = 16 >

Set number og digits printed for displaced geometries. See also #2 SaveDispGeom and #2 SaveDispGeomFormat.

**#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 ScalCoordInFit** !ADGA !Static

< *string* = FREQUENCY > [`FREQUENCY`] [`FILE`] [`OFF`]

Specify how to scale the grids when fitting property surfaces. Available options are:

FREQUENCY Use harmonic frequencies from the mmol file to calculate scaling as $\sqrt{\omega_m}Q_m$.

FILE Use scaling given in potential file on input.

OFF Do not use any scaling, i.e. $1 \cdot Q_m$

**#2 SeqAdgaKeoBuild** !ADGA

< *string vector* = N/A >

Specifies that the sequential ADGA workflow for constructing PESs and KEOs in general coordinates simultaneously is invoked. The argument of the keyword is a vector containing the names of the KEOs that are used in the ADGA runs of the workflow. Further details are given in Section 3.3.3.

**#2 SeqAdgaStepFullRestart** !ADGA

< *boolean* = FALSE > [`TRUE`] [`FALSE`]

Specifies that the PES construction in each ADGA run of the sequential ADGA workflow is started from scratch. This entails the repetition of potentially costly electronic structure calculations.
(By default, the current ADGA run in the sequential workflow uses the data from the previous ADGA run. This is the recommended behavior, thus use this keyword only if there is a specific reason.)

**#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 Symmetry

!ADGA !Static

Indicates that point group symmetry is to be used for surface construction, where some single points might be symmetry equivalent. The point group is automatically determined by MidasCpp, unless #3 PointGroup is used. 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 in general be oriented such that the symmetry elements (mirror planes, rotation axes) align with the Cartesian $x, y, z$ planes/axes (see #3 Frame for details on how to circumvent this restriction). 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.

### #3 CustomRotation

!ADGA !Static

< *real* *real* *real* >
< *real* *real* *real* >
< *real* *real* *real* >

Read in a 3-by-3 rotation matrix, used with the CUSTOM option of #3 Frame.

**Example for CustomRotation**

```
#2 Symmetry
   #3 Frame
      CUSTOM
   #3 CustomRotation
      -0.500000000000000 0.000000000000000 0.866025403784439
       0.000000000000000 1.000000000000000 -0.000000000000000
      -0.866025403784439 0.000000000000000 -0.500000000000000
```

### #3 Frame

!ADGA !Static

< *string* = LAB > [LAB] [INERTIA] [CUSTOM]

Define which frame to do symmetry operations in, by applying a rotation before the symmetry analysis. This will not affect the coordinates used for PES generation, but only what

frame is used for symmetry analysis. All symmetry operations are rotated back into the original frame after the analysis. LAB will not apply any rotations, INERTIA will rotate the molecule to the molecular Inertia frame, and CUSTOM will apply a custom rotation defined by #3 CustomRotation

**#3 PointGroup** !ADGA !Static

< *string* >

Force specific point group. The main rotation axis is given in parentheses.

**Example for PointGroup**

```
#2 Symmetry
   #3 PointGroup
      CS(X)
```

**#3 ThrGetCharacterFloatEqUlps** !ADGA !Static

< *integer* = 100000000 >

ULPS used for checking whether characters of input point group and actual molecule input agrees, compared relative to 1.

**#3 ThrNucleiIsNumEqualUlps** !ADGA !Static

< *integer* = 100 >

ULPS used for checking whether two nuclei have the same position, compared relative to 1.

**#3 ThrVibCoordsNumEqZeroUlps** !ADGA !Static

< *integer* = 10000 >

ULPS used when determining whether a vibrational coordinate is zero, compared relative to 1.

**#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 TdAdgaCalcDensitiesInterval

!TDADGA

< *real* = 10 >

Set the interval (in a.u.) for how often the wave function density should be evaluated in the TD-ADGA.

### #2 TdAdgaFullRestart

!TDADGA

Indicate that the TD-ADGA algorithm should employ the full restart scheme.

### #2 TdAdgaInfo

!TDADGA

< *integer* > [`Max MCL`]
< *string* > [`TD-WF name`]

Specifies the mode combination range for an TD-ADGA calculation in order to construct potential energy and/or molecular properties surfaces. The second argument is used to specify the name of the time-dependent wave function calculation, to calculate the TD-ADGA for.

**Example for TdAdgaInfo**

```
#2 TdAdgaInfo
   2        // Specifies up-to 2 mode cuts.
   mytdh    // Use TDH named 'mytdh'.
```

### #2 TdAdgaResetTime

!TDADGA

< *real* = 100 >

Set the reset time (in a.u.) to use in the TD-ADGA algorithm.

### #2 TdAdgaSaveAllDensities

!TDADGA

Save the non-smoothened, the space-smoothened and the space- and time-smoothened densities of the TD-ADGA in a separate file in the #2 AnalysisDir directory.

### #2 TdAdgaSmoothInitialDdensities

!TDADGA

Perform a space-smoothening of the initial density inputted to the TD-ADGA.
(Use this keyword when initializing the TD-ADGA from a single VSCF density no averaged over multiple eigenstates (See #3 PrepareInitialTdAdgaDensity).)

### #2 TdAdgaUpdateInterval

!TDADGA

< *real* = 10 >

Set the interval (in a.u.) for how often the TD-ADGA convergence should be checked.

#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 **UpdateDensMCL** !ADGA

< *integer* = 1 > [MCL]

Allows the ADGA to calculate and update the vibrational density at each iteration for mode combination levels up to and including the input level. Note that by specifying this keyword, the #2 CalcDensMCL keyword will automatically be set to the same MCL.

#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.

#2 **MLPes** !ADGA

This keyword activates an iterative PES construction scheme in which a GPR potential together with the grid is iteratively refined. In each iteration ADGA is used on the current GPR potential and uncertain points are identified and recalculated for a subsequent GPR training. The procedure is converged if no new points for re-training are added.

#3 **TolSelect** !ADGA

< *real* = $1 \cdot 10^{-8}$ >

Sets the threshold in GPR-ADGA calculation for adding new electronic structure single

points. The criterion used for comparison is the vibrational density times box size times uncertainty for this point.
(Should typically be in the range from $1 \cdot 10^{-8}$ to $1 \cdot 10^{-10}$. A tighter criterion can be set for small molecules.)

## 24.2 Advanced keywords

**#2 AdgaExtendIntegrationMethod** !ADGA

< *string* = INTEGRATE > [`INTEGRATE`] [`TRIANGLE`]

Set the method used by ADGA to determine the amount of density to the left and right of the potential boundary. When using INTEGRATE, the density to the left/right will be determined by a proper integration. When using TRIANGLE, the density to the left/right will be determined by a approximating the density as a triangle with height of the density value at the left/right potential bound and a base with the width of the smallest extension value. Note: This keyword will not affect whether or not the ADGA will extend the potential boundaries as this is determined solely by the integrated density inside the potential bounds.

**#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. With the MAX option dynamic extension of the grid through #2 DynamicAdgaExt is not available. Likewise, the keyword #2 AdgaExtendIntegrationMethod will have no effect.

**#2 BarFileStorageType** !ADGA !Static

< *string* = `ONFILE` > [`ONFILE`] [`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 DetermineInitialPointsFromDens** !ADGA

Indicate that the placement of the initial SPs in the zeroth ADGA iteration should be

guided by a density provided by the user on file. The initial single points will be placed such that one is placed at the refernce geometry of the .mmol file and two single points will be placed such that the inputted density is contained within the points within the #2 ItResDensThr threshold. If the maximum density is not near the reference geometry, an additional single point will also be placed at the maximum density value. The initial density must be provided in a number of files called MeanDens_Prop1_mode_label.mplot in the #2 AnalysisDir directory, where mode_label must be the mode label given to a mode in the .mmol file.

### #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 DynamicAdgaExt

!ADGA

< *boolean* = TRUE >

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 ExtMcrMaxMcl

!ADGA !Static

< *integer* >

This keyword will limit the maximally allowed mode combination level one can obtain when using the #2 ExtMcrModes keyword. This keyword takes an integer as argument and any mode combination containing more modes than this will not be included in the mode combination range.

### #2 ExtMcrModes

!ADGA !Static

< *string* **OR** *string* *integer* > [`<mode label>` **OR** `<mode label> <weigth>`]

.

.

[< *string* **OR** *string* *integer* >] [`<mode label>` **OR** `<mode label> <weigth>`]

This keyword will extend the maximum mode combination level (MCL) of the mode combination range (MCR) indicated by the #2 AdgaInfo, #2 TdAdgaInfo, or #2 StaticInfo keywords to include higher order mode combinations (MCs). For this purpose, a weight is assigned to each mode indicated under this keyword. MCs are added to the MCR if they satisfy $\sum_m w_m \leq n$, where $w_m$ is the weight assigned to each mode and $n$ is the "base" MCL provided under the #2 AdgaInfo, #2 TdAdgaInfo, or #2 StaticInfo keywords.
Only integer weights are accepted as input.

If only a mode label is provided per line the mode will be assigned a weight of 0.
Modes not listed under this keyword will be assigned the default weight of 1.
If this keyword is used in connection with static grids, the higher order grids accessed by this keyword must additionally be provided trough the #2 ExtendedGridSettings keyword.

**Example for ExtMcrModes**

```
// Example input for H2O molecule creating a MCR with all 1M MCs
// and the (Q0,Q1) and (Q1,Q2) MCs
#1 Pes
   #2 AdgaInfo
      1          // The "base" MCL
      vscf_calc // The VSCF calc to use in ADGA
   #2 ExtMcrModes // Manually specify mode weights
            // Q0 is not given. Default weight is 1.
      Q1    // When no weigth is given the weight default to 0
      Q2 1 // Optional: Weight of 1 is default
// The resulting MCR calculation becomes:
// MC          | weight | included
// (Q0)        | 1      | yes
// (Q1)        | 0      | yes
// (Q2)        | 1      | yes
// (Q0,Q1)     | 1      | yes
// (Q0,Q2)     | 2      | no
// (Q1,Q2)     | 1      | yes
// (Q0,Q1,Q2) | 2      | no
```

### #2 InitialTdAdgaDensity

!TDADGA

Indicate that the initial TD-ADGA density will be provided by the user on file. The initial density must be provided in a number of files called MeanDens_Prop1_mode_label.mplot in the #2 AnalysisDir directory, where mode_label must be the mode label given to a mode in the .mmol file.

### #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 NoLinVDensThr

!TDADGA

Disable the linear convergence check for the TD-ADGA, when constructing potential energy or molecular property surfaces.

(Only disable the linear threshold if you have a good reason for doing so as it will result in the TD-ADGA requesting a large number of singe points.)

**#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 NormalizeNormalCoordinates** !ADGA !Static

Explicitly normalize the molecular normal coordinates before running PES calculation.

**#2 OrthoNormalizeNormalCoordinates** !ADGA !Static

Explicitly ortho-normalize the molecular normal coordinates before running PES calculation.

**#2 PesBoundsFile** !Static !ADGA
< *string* = N/A >

This keyword can be used in combination with #2 UserDefinedPesBounds and #2 PesBoundsInput to specify maximum extents of the boundaries for a grid-based generation of a PES or molecular property surface. The keyword allows the user to provide the bounds through a separate file. Bounds provided through this keyword are overwritten by corresponding input through #2 PesBoundsInput. The argument to this keyword is the path to and file name of the file that contains the user-defined bounds. In that file, the bounds are provided exactly in the same way as via #2 PesBoundsInput. Further details are described under #2 UserDefinedPesBounds.

**#2 PesBoundsInput** !Static !ADGA
< *string* *string* = N/A **OR** *string* *real* **OR** *string* *real* *real* >
.
.
< *string* *string* = N/A **OR** *string* *real* **OR** *string* *real* *real* >

This keyword can be used in combination with #2 UserDefinedPesBounds and #2 PesBoundsFile to specify maximum extents of the boundaries for a grid-based generation of a PES or molecular property surface. The keyword allows the user to provide the bounds in the MidasCpp input file. Input through this keyword overwrites corresponding input from #2 PesBoundsFile. Further details are described under #2 UserDefinedPesBounds.

**#2 PolyInterpolationPoints** !ADGA
< *integer* = 4 >

Defines the number of points to be used for polynomial interpolation when the ADGA evaluates the integral over vibrational density.

**#2 SaveTdAdgaCheckTimes** !TDADGA

Enable the saving of TD-ADGA iteration times on file.

**#2 ScreenOperCoef** !ADGA !Static

< *string* = TRUE > [TRUE] [FALSE]

Turns operator screening on or off when writing operator terms to file by specifying either TRUE or FALSE. The screening threshold can be set using #2 ScreenOperCoefThresh. (The recommendation is to leave operator screening on.)

**#2 ScreenOperCoefThresh** !ADGA !Static

< *real* = $1.0 \cdot 10^{-14}$ >

Screening threshold used with operator screening turned on. See #2 ScreenOperCoef.

**#2 UserDefinedPesBounds** !Static !ADGA

< *boolean* = FALSE > [TRUE] [FALSE]

This keyword can be used to activate user-defined maximum extents of the boundaries for a grid-based generation of a PES (or other molecular property surface). The bounds restrict the real-space domain of each mode (as opposed to the domain of the integer logic used for the iterative grid extensions). The bounds may be specified through the MidasCpp input file (#2 PesBoundsInput) or a separate file (#2 PesBoundsFile). There are several ways to provide the bounds, which are illustrated in the example below. If the bounds for a mode are not specified or if its label is provided as 'OFF', MidasCpp's default way of setting up the bounds is used for that mode. The bounds are given as absolute (positive) distances from the value of the respective mode at the reference position. The units for the maximum extensions are normal coordinate ($Q$) units divided by $\sqrt{u/m_e}$ ($m_e$: *atomic units of mass*, $u$: *atomic mass units*) if normal coordinates are used, and $r_\mathrm{B}$ (Bohr) and radians if general coordinates are used. Internally, the bounds provided through this keyword are always multiplied by the grid scaling factors (optionally specified through #2 PesGridScalFac). For rectilinear coordinates, the maximum bounds are further extended when the ADGA is used, by multiplying with the factor $2^{f_\mathrm{it}}$, where $f_\mathrm{it}$ is specified through #2 ItExpGridScalFact. If polyspherical or general coordinates are used, the provided bounds multiplied with the grid scaling factors are not further modified. In this case, the user has to pay attention that the provided maximum bounds do not surpass the natural domains of the internal coordinates. For example, for a dihedral angle the bounds should be within 0 and $2\pi$ ($-\pi$ and $\pi$ if the reference angle is 0).
(In a standard (ADGA) PES calculation, there is no need for this keyword as the default maximum extensions are well selected. In fact, setting bounds manually is somewhat at odds with the ADGA that determines bounds efficiently and autonomously.)

**Example for UserDefinedPesBounds**

```
// situation with five sequentially labeled modes 'Qx'
#2 UserDefinedPesBounds
   True
#2 PesBoundsInput
   Q2   OFF          // bounds for Q2 set in default way
   Q1   0.9   1.4    // bounds for Q1 set to 0.9 (left) and 1.4 (right)
   Q4   1.2          // both bounds for Q4 set to 1.2
                     // bounds for Q3 also set in default way
#2 PesBoundsFile
   <path to and name of file with Q5 bounds> // contains e.g. 'Q5 0.5'
```

### #2 GLQuadraturePoints

!ADGA !Static

< *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 25

# 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`.

## 25.1 General keywords

**#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.

**#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 FitBasisDef

!FitBasis

Directs the input to the general fit-basis handler, see section 26.

### #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 **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* = 12 **OR** *integer vector* >

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 (see #3 AddFitFuncsScheme).

### #3 IoLevel

!FitBasis

< *integer* = 0 >

Indicates the IoLevel of the FitBasis block.

### #3 Name

!FitBasis

< *string* >

Identify the fit-basis by a name.

## 25.2 Advanced keywords

**#3 AddFitFuncsScheme** !FitBasis

< *string* = CONSERVATIVE > [`CONSERVATIVE`] [`LEGACY`]

Sets the scheme for truncating the order of the fitting basis, based on the number of single points available (see also #3 FitFunctionsMaxOrder). This applies until the maximum order of the fit functions is reached. There are two modes of operation:

**CONSERVATIVE**

The number of fit functions is 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.

**LEGACY**

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 is what historically has been used in older versions of MidasCpp.

**#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* = 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 NthreadsFit** !FitBasis

< *integer* = `--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 26

# FitBasisDef and .mfitbas input options

This section details the input options available to the general fit-basis handler, which can be invoked either from the main `MidasCpp` input file or from an `.mfitbas` file. If the input if found in the `MidasCpp` input file, it goes under the #3 FitBasisDef input block, and the keywords listed here should be given with `N=4` and `N1=5`. The `.mfitbas` file are read between the #0 MidasFitInfo and #0 MidasFitInfoEnd keywords, and the other keywords listed in this section should in this case be given with `N=1` and `N1=2`.

## 26.1 General keywords

**#<N> SpecFitBasis** !FitBasis !FitBasisDef

Directs the input to the general fit-basis handler.

**#<N1> FitFunctions** !FitBasis !FitBasisDef
< *string* >

Defines the complete set of fit-basis functions, which are to be used in the linear fitting procedure.

**#<N1> Modes** !FitBasis !FitBasisDef
< *integer vector* >

Defines the vibrational modes that should be linearly fitted with the set of fit-basis functions specified under the #<N1> FitFunctions keyword. Note that the modes are counted from 0 to $N_{\text{modes}} - 1$, where $N_{\text{modes}} = 3N - 6(5)$, in accordance with appearance in the `Molecule.mmol` file.

**#<N1> OptFunc** !FitBasis !FitBasisDef
< *string* >

Give a function which is used to optimize the parameters of the functions given under

#<N1> 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)
```

**#<N1> OptFuncThr** !FitBasis !FitBasisDef

< *real* = $1.0 \cdot 10^{-8}$ >

This number determines the gradient convergence for the non-linear optimization of parameters in the function specified under #<N1> OptFunc.

**#<N1> Props** !FitBasis !FitBasisDef

< *string vector* >

Defines the properties that the set of fit-basis functions given under the #3 FitBasisDef 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.

**#0 MidasFitInfo**

This keyword is used to signify the beginnig of a `MidasCpp` fitbasis input file (.mfitbas). All lines before this keyword will be ignored. The keywords allowed in this type of input file are outlined in the chapter 26.

**#0 MidasFitInfoEnd**

This keyword is used to signify the end of a `MidasCpp` fitbasis input file (.mfitbas). All lines after this keyword will be ignored. This keyword is paired with the #0 MidasFitInfo keyword.

## 26.2 Advanced keywords

**#<N> Constants** !FitBasis !FitBasisDef

< *string 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.

### #<N> Functions

!FitBasis !FitBasisDef

Read a set of functions in the same format as #<N1> FitFunctions.

# Chapter 27

# System input options

This section details the input options available to both defining or modifying a molecular system. The #1 ModSys keyword block is used to specify in which manner the coordinates of a given subsystem should be modified.

## 27.1 General keywords

**#1 System**

Read System input until next level #1 input key.

**#2 AddLocalTransRot**

Switches on the addition of the local translations and rotations for each subsystem.

**#2 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.)

**#2 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 #2 MoleculeFile keyword.

**#2 MoleculeFile**
< *string* = MIDAS > [MIDAS] [METAMOL] [MOLDEN] [TURBOMOLE] [ORCAHESS] [SINDO]
< *string* = Molecule.mmol >

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 involves the reference geometry in Cartesian coordinates, vibrational frequencies and vibrational coordinates.
(The standard setting should be MIDAS for all purposes regarding surface generation for a single molecular system. For more information on the MIDAS molecule format, see 41. In case of double incremental calculations (DIF/DIFACT), METAMOL is used. For details on the METAMOL file format, see. 42.)

**Example for MoleculeFile**

```
#2 MoleculeFile
   MIDAS               // Read input in MidasCpp molecule format (.mmol).
   mymolecule.mmol     // Read file 'mymolecule.mmol' in main-directory.
```

### #2 Name

< *string* >

Indicates the name of the system or subsystem for later reference.

### #1 ModSys

Read definitions for modifying a molecular system.

### #2 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 $C_1$ point group.
(Note that this keyword will only have an effect if the ROTCOORD scheme is selected under the #2 ModVibCoord keyword.)

### #2 ModVibCoord

!ModVibCoord

< *string* > [FREQANA] [FALCON] [NORMALCOORDINATES] [LOCALBONDSTRETCHES] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

Indicates the scheme with which to modify the vibrational coordinates, see sections 29, 30 , 32 for more details.

### #2 RemoveGlobalTr

Remove the global translational and rotational degrees of freedom before entering the selected scheme under the #2 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.

**#2 RotateToPrincipalAxes**
< *boolean* = true >

Indicate whether the input structure should be rotated to the principal axes when constructing vibrational coordinates. Only has an effect when chosing the `NORMALCOORDINATES` or `LOCALBONDSTRETCHES` options under the #2 ModVibCoord keyword.

**#2 SysNames**
< *integer* >
< *string* >

Indicates the number of subsystem names, followed by lines giving the names.

## 27.2 Advanced keywords

**#2 RmZeroFreqComponentsUlps**
< *integer* = 64 >

Set number of ULPS for relative cut-off of Hessian Eigenvalues. Cut-off is relative to largest Eigenvalue.

# Chapter 28

# Falcon input options

This section details the input options for the iterative merge of subsystems, which is known as the FALCON algorithm.

## 28.1 General keywords

**#2 ModVibCoord** !ModVibCoord

< *string* > [FREQANA] [FALCON] [NORMALCOORDINATES] [LOCALBONDSTRETCHES] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

When called with FALCON as argument, the following keywords are available.

**#3 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.

**#3 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.

**#3 MaxSubSysForRelax** !FALCON

< *integer* = $\infty$ >

Reads in the maximum number of initial subsystems in a fusion groups to be relaxed.

**#3 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.

### #3 MinCouplingForRelax

!FALCON

< *real* = 0.0 >

Reads in a value specifying an upper bound to the coupling estimate for relaxation of the fusion group.

### #3 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, #3 HessType has to be set.

### #3 NoCapping

!FALCON

Turns off saturation of cut covalent bonds with hydrogens (capping). This option is useful for non-covalently bonded clusters, where capping should not be applied at all and, if used nevertheless, could go very wrong.

### #3 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.

### #3 WriteIncrInputFromFile

!FALCON

< *string* >

Sets up the input for a double incremental calculation. A filename is given as the argument. From this file, user-defined fragment combinations and coefficients are read in.

**Example for WriteIncrInputFromFile**

```
#3 WriteIncrInputFromFile
setup.txt

//Example for setup.txt:
1 1,2 1  // First column: FC count, starting from one
2 2,3 1  // Second column: numbers of the fragments part of a FC
3 2 -1  // Third column: coefficients
```

## 28.2 Advanced keywords

**#3 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).

**#3 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)

**#3 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.

**#3 PrepRun** !FALCON

Performs only a preparation run, where subsystems are formed but no operations with modes are carried out.

# Chapter 29

# 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 4.3. This is done using electronic single point calculations with input as described in section 14 in order to calculate gradients for the displaced nuclear configurations. The gradients are subsequently used to calculate the Hessian, which is afterwards diagonalized.

If #3 Targeting is enabled the program uses the modes in the MoleculeInfo file (i.e. those listed under the keywords `#1VIBCOORD -> #2COORDINATE` as described in 41) as targets when iteratively solving mass-weighted Hessian eigenvalue problem.

## 29.1 General keywords

**#2 ModVibCoord** !ModVibCoord

< *string* > [FREQANA] [FALCON] [NORMALCOORDINATES] [LOCALBONDSTRETCHES] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

When called with `FREQANA` as argument, the following keywords are available.

**#3 DispFactor** !ModVibCoord

< *real* = 0.01 >

The displacement factor used when creating the displaced configurations. ($\delta$ in section 4.3.)

**#3 DumpInterval** !ModVibCoord

< *integer* = 10 >

Interval for dumping property files to disc (as a safety in case of crash).

**#3 ItEqBreakDim** !ModVibCoord

< *integer* = 1000 >

Break dimension for the iterative solver. Only applicable if #3 Targeting is enabled.

**#3 ItEqEnerThr** !ModVibCoord

< *real* = $10^{-8}$ >

Convergence threshold for the relative change in energy for the iterative solver ($\epsilon_{\text{energy}}$ in section 4.3). Only applicable if #3 Targeting is enabled.

**#3 ItEqMaxIt** !ModVibCoord

< *integer* = 1000 >

Maximum number of iterations for the iterative solver. Only applicable if #3 Targeting is enabled.

**#3 ItEqResThr** !ModVibCoord

< *real* = $10^{-8}$ >

Convergence threshold for the norm of the residual for the iterative solver ($\epsilon_{\text{res}}$ in section 4.3). Only applicable if #3 Targeting is enabled.

**#3 MolInfoOutName** !ModVibCoord

< *string* = `MoleculeInfo_FreqAna.mmol` >

Name of the output file that the resulting MoleculeInfo (containing eigenfrequencies and -vectors) should be written to.

**#3 NonSymmetricHessian** !ModVibCoord

If this keyword is set the (reduced) Hessian is not symmetrized before diagonalization.

**#3 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 #3 OverlapSumMin ($n_{\text{overlap,max}}$ in section 4.3). Only applicable if #3 Targeting is enabled.

**#3 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 ($\epsilon_{\text{overlap}}$ in section 4.3). Eigenvectors are allowed to occur in the overlap sums for several targets. Only applicable if #3 Targeting is enabled.

**#3 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.)

**#3 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).)

**#3 SinglePointName** !ModVibCoord

< *string* = `SP_FreqAna` >

Name of singlepoint calculation. Must correspond to a name given under #2 Name in #1 SinglePoint (see chapter Singlepoint input options).

**#3 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 #3 ItEqEnerThr, #3 ItEqResThr, #3 ItEqBreakDim, #3 ItEqMaxIt, #3 OverlapSumMin, #3 OverlapNMax, only have effect if this keyword is set.

# Chapter 30

# Normal Coordinates input options

This section details the input options for generating the normal coordinates from a molecule input containing hessian information. It is generally advised to use this method for calculating the normal coordinates for use in a PES calculation, since the normal coordinates provided by external electronic structure programs does not necessarily use the same atomic masses as `MidasCpp` does.

## 30.1 General keywords

**#2 ModVibCoord** !ModVibCoord

< *string* > [FREQANA] [FALCON] [NORMALCOORDINATES] [LOCALBONDSTRETCHES] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

**#3 Disablefrequencycheck** !ModVibCoord

Disables the check of the frequencies after they have been calculated.
( It is strongly recommended not to disable this check.)

**#3 Ignoregradient** !ModVibCoord

Ignores gradient information, if it is present in the molecule input.

**#3 Outputfile** !ModVibCoord

Setup filename and precision of values for output file.

**#4 Filename** !ModVibCoord

[< *string* = NormCoord.mmol >]

The name of the output .mmol file, given in the format explained in 41.

**#4 Precision** !ModVibCoord

[< *integer* = 12 >]

Sets the number of digits to print in output files.

**#3 Projectouttransrot** !ModVibCoord

[< *integer* = 1 >]

If no integer is given as input, the default behaviour of projecting out translation and rotation from the hessian is enabled. If a number larger than one is given the projection is repeated that number of times.

# Chapter 31

# Local Bond Stretch Coordinates input options

This section details the input options for generating the local bond strecth coordinates from a molecule input containing hessian information.

## 31.1 General keywords

**#2 ModVibCoord** !ModVibCoord

< *string* > [FREQANA] [FALCON] [NORMALCOORDINATES] [LOCALBONDSTRETCHES] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

**#3 Lbstype** !ModVibCoord

< *string* > [LBS] [pLBS|ProjLBS] [opLBS|OrthoProjLBS]

Select the type of local bond stretch to use when constructing coordinates and kinetic energy operator. The LBS and pLBS coordinates are not orthonormal and when specifying the Hamiltonian, the FROMFILE option must be specified under the #3 KineticEnergy keyword. The opLBS coordinates are orthonormal and can use the same kinetic energy options as normal coordinates.

# Chapter 32

# 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.

## 32.1 General keywords

**#2 ModVibCoord** !ModVibCoord

< *string* > [FREQANA] [FALCON] [NORMALCOORDINATES] [LOCALBONDSTRETCHES] [ROTATEMOLECULE] [ROTCOORD] [TRANSFORMPOT]

**#3 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.

**#3 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.)

**#3 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.

**#4 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.

**#4 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.)

**#4 OptParam**

< *string* = 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 #3 Measure keyword is chosen.)

**#4 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.

**#4 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.)

**#3 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.

**#4 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.)

**#4 Name**

< *string* >

Specifies the name of the VSCF calculation.

**#3 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.

**#3 PrintRotMatToFile** !ModVibCoord

< *string* >

Specifies a file name which will be given to a file that contains the final rotation matrix.

**#3 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.

**#3 RotationSuffix** !ModVibCoord

< *string* = _rotated >

Specifies the suffix appended to all files which contains rotated values, e.g. .mop and .mmol files.

**#3 Sweeps** !ModVibCoord

< *integer* = 6 >

Defines the maximum number of sweeps that can be carried out in the Jacobi sweep algorithm.

# Chapter 33

# Machine Learning options

## 33.1 General keywords

**#1 MLTask**

Read General input until next level #1 input key.

**#2 CoVarAlgo** !MLEARN

< *string* = CHOL > [CHOL] [LU] [SVD] [BKD] [SPARSE]

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 decomposition 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.
- SPARSE It activates sparse GPR. Note that you then also have to set #2 InducingPoints.

**#2 CoordType** !MLEARN

< *string* > [MINT] [XYZ] [ZMAT] [XYZCENTER] [DIST] [EIGDIST] [SYMG2] [SYMG4]

Specifies the coordinate type used to interpret the structures you specified via the xyz files.
(It is recommended to use internal coordinates MINT.)

**#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 GprShift** !MLEARN

< *real* = 0.0 >

Sets a shift, which is applied to all learned inputs $y$. This might help to increase numerical stability.

**#2 Hopt** !MLEARN

Specifies the hyper parameters optimization settings.

**#2 IoLevel** !MLEARN

< *integer* = 0 > [< *integer* >]

Indicates the IoLevel (print level) for Machine Learning modules.

**#2 InducingPoints** !MLEARN

< *integer* = -1 >

Sets the number of inducing points if Sparse GPR is selected in #2 CoVarAlgo. Per default all points are inducing points, which makes SGPR wasteful.

**#2 Kernel** !MLEARN

< *string* >
[< *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. Note that more than one kernel can be specified by giving additional lines. This will activate Multilayer GPR, where each line indicates one layer. Usually more than 2 layers don't increase accuracy and can cause numerical problems.

**Options**

- 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

$$k_{\mathbf{xy}} = \sigma_n^2 \left(1 + \sqrt{5} \sum_i^d \frac{|x_i - y_i|}{l_i} + 5 \sum_i^d \frac{(x_i - y_i)^2}{3 l_i^2}\right) \times \exp\left(-\sqrt{5} \sum_i^d \frac{|x_i - y_i|}{l_i}\right)$$

- 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

$$k_{\mathbf{xy}} = \sigma_n^2 \left(1 + \frac{\sqrt{5}\,\|\mathbf{x} - \mathbf{y}\|}{l} + \frac{5\,\|\mathbf{x} - \mathbf{y}\|^2}{3 l^2}\right) \exp\left(-\sqrt{5} \frac{\|\mathbf{x} - \mathbf{y}\|}{l}\right)$$

- 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 MeanFunction** !MLEARN

< *string* = `ZERO` > [`ZERO`] [`HESSIAN`]

Specifies the mean function used in the Gaussian Process. Usually a `ZERO` mean function is applied. Also a Hessian can be selected as mean function, but then an additional line is required given the path to the Hessian in the ORCA format.
(It is recommended to use internal coordinates with `ZERO`.)

**#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 OnlyICoord** !MLEARN

Activates that only internal coordinates should be computed and nothing else. This option is mainly meant for analysis and debugging.

**#2 PredictMode** !MLEARN

< *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** !MLEARN

< *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 avoids recomputation of the co-variance matrix for each single point and is therefore the efficient choice. However only simple sanity checks are performed to verify that it is a valid co-variance matrix.

**#2 ReadIcoordDef** !MLEARN

< *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 ReadWeights** !MLEARN

< *string* >

Specifies the location of pre computed weight for GPR, which can be used in GPR to bypass the construction of the co-variance matrix and its inversion. This can be a major efficiency boost, but without co-variance matrix no uncertainty can be computed.

**#2 Sample** !MLEARN

< *integer* >

Activates Farthest point sampling that will be performed on the data set specified in #2 TrainSet. This keyword reads in one integer argument which specifies the number of samples it should generate. Note that it uses a kernel to define the distance measure:
$d_i = 1 - \frac{1}{n-k^{(n)}} \sum_j^{n-k^{(n)}} \tilde{k}(\mathbf{x}_i, \mathbf{x}_j)$

**#2 StoreCoVar** !MLEARN

< *string* >

Specifies the location to save a (decomposed) co-variance matrix, which can later be reused with the #2 ReadCoVar keyword.

**#2 StoreWeights** !MLEARN

< *string* >

Specifies the location where weights of a GPR training are saved.

**#2 TestSet** !MLEARN

< *string* >

A keyword used to provide training set data.

**#3 File** !MLEARN

< *string* >

Specifies the location of the file containing the coordinates for which a ML algorithm should be used for prediction or classification. For the format, see the subkey #3 File of #2 TrainSet.

**#3 Units** !MLEARN

< *string* > [BOHR] [ANGSTROM]

Specifies units used for the data in the file provided under #3 File.

**#2 TrainSet** !MLEARN

< *string* >

A keyword used to provide training set data.

**#3 File** !MLEARN

< *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 File**

```
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
```

### #3 Units

!MLEARN

< *string* > [BOHR] [ANGSTROM]

Specifies units used for the data in the file provided under #3 File.

### #2 GPR

!MLEARN

Activates that Gaussian Process Regression (GPR) should be performed for the coordinates given in #2 TestSet. The GPR is trained using the data specified in #2 TrainSet

# Chapter 34

# Additional Machine Learning options

The following keywords can be used under both #1 MLTask (in the stand-alone machine learning module) and #2 MLPes (in GPR-ADGA-type computations). Therefore, the input level can vary.

## 34.1 General keywords

**#<N> Noise** !MLEARN

$< real = 1 \cdot 10^{-8} >$

Specifies the noise imposed on the input data. The noise acts as regularization term and can generate a smoother predictor.
(Should typically be in the range from $1 \cdot 10^{-8}$ to $1 \cdot 10^{-10}$. A tighter criterion can be set for small molecules.)

# Chapter 35

# 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).

## 35.1 General keywords

**#<N> Type**

$<$ *string vector* $=$ MIDAS DOPR853 $>$ [`MIDAS DOPR853`] [`MIDAS DOPR5`] [`MIDAS TSIT5`] [`GSL RK2`] [`GSL RK4`] [`GSL RKF45`] [`GSL RKCK`] [`GSL RK8PD`] [`GSL MSADAMS`]

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 (no limit) $>$

Maximum number of steps in the integrator. -1 means no limit.

#<N> **Tolerance**
< *real vector* = 1.e-12 1.e-12 >

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.

## 35.2 Advanced keywords

#<N> **StepSizeControl**
< *real vector* = Individual defaults set for each stepper >

Set constants for step-size control. Reads in 5 numbers: $\alpha$, $\beta$, $safe$, $minscale$, $maxscale$. 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> AllowStepsizeOvershoot**

Allow the ODE driver to go past the end time in an integration call if the final step can be larger than the one required for reaching the end time. This keyword can be used in a TD-ADGA calculation to let the ODE integrator overstep a TD-ADGA check time before checking for convergence or calculating a new density. Even though this keyword is set, the integration will never go past the final end time.
(Only use if you have a good reason to not hit TD-ADGA time intervals exactly.)

**#<N> ErrorType**
< *string* = 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 36

# Tensor-decompostion options (stand-alone module)

This section describes how to run a stand-alone tensor decomposition, which will read in a tensor from disk and decompose it. It includes some keywords for defining the file paths, and types of files, and allows to control the actual tensor decomposition through the keywords listed in section 37 (see #2 TensorDecompInfo).

## 36.1 General keywords

**#1 TensorDecomp**

This keyword is used to signify the beginning of a Tensor Decomp block.

**#2 HOOIMaxIter**
< *integer* = 10 >

Set maximum number of iterations for HOOI.

**#2 HOOIMode**
< *string* = HOOI > [`HOOI`] [`HOSVD`]

Switch between HOOI and the simpler HOSVD.

**#2 HOOIThresh**
< *real* = 1e-12 >

Set SVD rank truncation threshold for HOOI and HOSVD.

**#2 IoLevel**
< *integer* = 1 >

Set the IoLevel for the Tensor Decomp calculation.

**#2 Name**
< *string* = tdecomp >

Set the name of the Tensor Decomposition calculation. The name is used for naming of, for example, tensor output files.

**#2 RTENInfo**

Start RTEN input.

**#3 RTENChronicle**
< *string* = N/A >

Create a fit chronicle file containing some metrics for each iteration. Takes one argument, which is the path of the data file to create.

**#3 RTENMaxIter**
< *integer* = 100 >

Set maximum number of iterations allowed in RTEN. The RTEN expansion will stop after the maximum number of iterations is reached regardless of whether a converged solution was found.

**#3 RTENThr**
< *real* = 1e-2 >

Set global RTEN threshold.

**#3 RTENUseActAccErr**
< *string* = FALSE > [`TRUE`] [`FALSE`]

Use Actual Accumulated Error (ActAccErr) instead of Bound Accumulated Error (BoundAccErr).

**#3 RTENWriteToDisk**
< *string* = FALSE > [`TRUE`] [`FALSE`]

Write the RTEN expanded tensor to a file called `rlten.data` after the algorithm has finished.

**#2 TensorDecompInfo**
< *string* >

Start a TensorDecompInfo input block. See section 37 for allowed options.

**#2 TensorDecompType**

< *string* = DECOMPOSER > [DECOMPOSER] [TUCKER] [RTEN] [HOOI]

Choose between the MidasCpp decomposer frameworks.

**DECOMPOSER** CP decomposition.

**TUCKER** Tucker decomposition.

**RTEN** RTEN tensor expansion.

**HOOI** HOOI/HOSVD.

**#2 TensorFilePath**

< *string* >

Path to the tensor file.

**#2 TensorFileType**

< *string* = SIMPLEASCII > [SIMPLEASCII] [NICETENSOR] [MATLAB] [TENSORDATACONT] [TURBOMOLE]

Type of tensor file.

**SIMPLEASCII** Simple ASCII format. First line defines the dimensional extents, and subsequent lines gives tensor elements, one element pr. line.

**NICETENSOR** NiceTensor binary format.

**MATLAB** Matlab **.mat** file. Must also provide tensor name with #2 TensorName.

**TENSORDATACONT** TensorDataCont (stacked tensors), originating from CP-VCC or similar.

**TURBOMOLE** Turbomole RI-integrals (**.aux**).

**#2 TensorName**

< *string* >

Name of the tensor. Used to identify the tensor when reading from MatLab .mat files.

**#2 TensorOutput**

< *string* = NICETENSOR > [NICETENSOR] [ASCII] [MATLAB] [NONE]

Type of tensor output file. Multiple types can be given (each on a separate line), and an output will be created for each type. Will read until the next keyword.

**NICETENSOR** NiceTensor binary format.

**ASCII** Simple ASCII format.

**MATLAB** Matlab **.mat** file.

**NONE** No output (disable default output).

#### #2 TensorReorder

< *integer vector* = N/A >

Provide a new order of dimensions to which the tensor should be re-ordered before decomposition.

#### #2 TuckerInfo

Start Tucker decomposition input.

#### #3 SvdAlgorithm

< *string* = GESDD > [GESDD] [GESVD] [SYEV] [SYEVD]

Use either GESDD or GESVD for determining SVDs. When using #3 TuckerAlgorithm NATURALBASIS one can additionally use either SYEV or SYEVD.

#### #3 SvdRankType

< *string* = ACCREL > [ACCREL] [ACCABS] [RANKREL] [RANKABS]

Define how to determine SVD rank using #3 SvdThr. Error meassures are defined for HOSVD as follows:

**ACCREL** Rank $k$ is based on relative accuracy, such that the truncated SVD satisfies

$$sqrt(||A - \tilde{U}\tilde{S}\tilde{V}^{\dagger}||^2/||A||^2) <= T$$

.

**ACCABS** Rank $k$ is based on absolute accuracy, such that the truncated SVD satisfies

$$sqrt(||A - \tilde{U}\tilde{S}\tilde{V}^{\dagger}||^2) <= T$$

.

**RANKREL** Rank $k$ is based on singular value cut-off relative to largest singular value, such that

$$s_{k+1}/s_0 <= T$$

.

**RANKABS** Rank $k$ is based on singular value cut-off, such that

$$s_{k+1} <= T$$

.

Corresponding measures are defined for `NATURALBASIS`, which ensure that using the same #3 SvdThr and #3 SvdRankType we obtain the same decomposition regardless of which #3 TuckerAlgorithm is used. This allows for direct comparison of the two algorithms.

**#3 SvdSignFlipType**

< *string* = NONE > [`NONE`] [`SIMPLE`]

Employ a strategy to prevent singular vector sign ambiguity.

**NONE** Nothing is done.

**SIMPLE** Define first element of each $u_i$ vector to be positive and determine sign based on that.

**#3 SvdThr**

< *real* = 1e-12 >

Set threshold for SVD. See also #3 SvdThrType and #3 SvdRankType.

**#3 SvdThrType**

< *string* = FULL > [`FULL`] [`MODE`]

Use #3 SvdThr as threshold for the full decomposition or each individual mode. If set to `FULL` each mode will use a threshold of $T^{1/N}$ for an N-order tensor. If set to `MODE` each mode will use a threshold of $T$ directly.

**#3 TuckerAlgorithm**

< *string* = HOSVD > [`HOSVD`] [`NATURALBASIS`]

Choose between the MidasCpp Tucker decomposition algorithms.

**HOSVD** Do standard HOSVD.

**NATURALBASIS** Do Natural Basis decomposition.

**#3 TuckerSequential**

< *string* = FALSE > [`TRUE`] [`FALSE`]

If set to `TRUE`, the algorithm will do projections on the target tensor with each orthogonal mode matrix before finding the projection for the next mode. The projection matrices are thus found sequentially so to speak. This means that the order in which the projections are done will have an effect on the projection matrices and thus the resulting core matrix. Some strategies have been implemented to make the behaviour not depend on the order of modes in the tensor (see #3 TuckerSequentialStrategy).

**#3 TuckerSequentialStrategy**

< *string* = DEFAULT > [DEFAULT] [BESTRANK] [BESTREDUCTION] [BESTERROR]

Strategy to employ for determining mode projection order in a sequential calculation.

**DEFAULT** Default strategy is to do it in order from last to first. Will not recalculate any SVDs of matricizations. Result will depend on tensor mode ordering.

**BESTRANK** Do the projections in order of which has the lowest rank. Will recalculate SVDs of matricizations after a projection is made before choosing which projection to make next.

**BESTREDUCTION** Do the projections in order of which has the largest rank reduction. Will recalculate SVDs of matricizations after a projection is made before choosing which projection to make next.

**BESTERROR** Do the projections in order of which introduces the smallest absolute truncation error. Will recalculate SVDs of matricizations after a projection is made before choosing which projection to make next.

For the BESTRANK and BESTREDUCTION strategies, ties between modes are settled using the BESTERROR option, *i.e.* using the SVD truncation error to select the mode for which the rank truncation introduces the least error in an absolute sense.

# Chapter 37

# 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).

## 37.1 General keywords

**#<N> Type**

$<$ *string vector* $=$ NOPREPROCESSOR <driver> <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-12 $>$

Set threshold for CP-ALS algorithm.

**#<N> CpAlsRelativeThreshold**
< *real* = 0 (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.e-2 >

Requested absolute accuracy of CP fit in FINDBESTCP driver.

**#<N> Guesser**
< *string* = RANDOM > [`RANDOM`] [`CONSTVAL`] [`SVD`] [`SCA`] [`UNIT`] [`DIAGNORM`]

Set the policy for generating starting guesses for CP decompositions.
(Use SVD or RANDOM initialization.)

**#<N> GuessIncrementer**

< *string* = 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* = 1.e-10 >

Set threshold for CP-NCG algorithm.

**#<N> CpNcgRelativeThreshold**

< *real* = 0 (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* = 1.e-10 >

Set accuracy of intermediate Tucker representation used in the C2T preprocessor.

**#<N> CpNcgType**

< *string* = 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* = WOLFE > [WOLFE] [EXACT] [HZ]

Set type of NCG line search.
(EXACT is often the best.)

**#<N> CpAsdThreshold**

< *real* = 1.e-10 >

Set threshold for CP-(P)ASD algorithm.

**#<N> CpAsdRelativeThreshold**
< *real* = 0 (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-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.

#<N> **Diis**
< *string* = NONE > [`NONE`] [`AGGRESSIVE`] [`CONSERVATIVE`]

Use DIIS to try to accelerate CP-ALS iterations. DIIS step is taken at the end of the ALS iterations, after convergence checks, just before the next ALS iteration. This means that when the algorithm converges, the returned tensor is a solution to the ALS problem. Also any errors or metrics reported are for the ALS solution. Options are:

**NONE** No DIIS acceleration.

**AGGRESSIVE** Aggressive DIIS (aDIIS), always use the DIIS step.

**CONSERVATIVE** Conservative DIIS (cDIIS), only use DIIS step if it improves on the ALS solution. Has the added cost of an extra norm evaluation compared to aggresive DIIS.

#<N> **DiisSubspace**
< *integer* = 3 >

The maximum number of subspace vectors to use with DIIS accelerated CP-ALS.

#<N> **DiisStartIter**
< *integer* = 3 >

On which iteration to start the DIIS acceleration of CP-ALS.

#<N> **DiisRcond**
< *real* = 1e-12 >

Rcond for DGELSS used to solve the DIIS linear system. Cuts off small singular values compared relatively to the largest singular value.

#<N> **FitChronicle**
< *string* = N/A >

Create a fit chronicle file which reports errors etc. for each CP-ALS iteration. Takes one argument, which is the path of the data file to create.

## 37.2 Advanced keywords

#<N> **CpAlsRCond**
< *real* = 1.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 (turned off) >

Read two parameters $\alpha$ and $q$. $\alpha$ is the Tikhonov regularization parameter. The regularization can be turned off during the iterations by multiplying $\alpha$ 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 $\Delta R_{in}$ 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 $\Delta R_{in}$ is set using #<N> NestedCpRankIncr.

#<N> **CpAlsNormRegularization**
< *real* = 0.0 (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 38

# Laplace-decomposition options

## 38.1 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'.

## 38.2 Advanced keywords

#<N> **HistogramDelta**
< *real* = 1.0e-1 >

Set the spacing for constructing an energy histogram used for optimizing the quadrature.

# Chapter 39

# 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 #3 MoleculeInput keyword.

The default behaviour is (1), so if one wish to utilize behaviour (2) either #3 MidasDynLib or #3 MidasPot should be present in the input.

## 39.1 General keywords

**#1 FileConversion**

Read input for fileconversion until next #1 input key.

**#2 MoleculeFileConversion**

Allows input of Molecule file conversion input, e.g. for creating MidasPot.

**#3 Description**

$< \textit{string} = \text{N/A} >$

Gives a description of the MidasPot which can be output by the generated program.

**#3 MidasDynLib**

If set, the potential C++ file generated will be for creating a dynamically linked library.

**#3 MidasPot**

If set, the potential C++ file generated will be for creating a stand alone program.

#### #3 MoleculeInput

< *string* > [MIDAS] [MOLDEN] [TURBOMOLE] [ORCAHESS] [SINDO] [XTB]
< *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 involves the reference geometry in Cartesian coordinates, vibrational frequencies and vibrational coordinates.
(For more information on the MIDAS molecule format see 41.)

**Example for MoleculeInput**

```
#2 MoleculeInput
   MIDAS               // Read input in MidasCpp molecule format (.mmol).
   mymolecule.mmol     // Read file 'mymolecule.mmol' in main-directory.
```

#### #3 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 41.)

**Example for MoleculeOutput**

```
#2 MoleculeOutput
   MIDAS               // Write in MidasCpp molecule format (.mmol).
   mymolecule.mmol     // To the file 'mymolecule.mmol' in main-directory.
```

#### #3 OperatorInput

< *string* = N/A >
< *string* = N/A >
< *string* = N/A > [< *string* >]
[< *string* = N/A >]

Takes an operator name, description, filename of the operator file, optionally a reference value index and optionally the file name of an one_mode_grids.mbounds file. These are then parsed into C++ code according to the MidasPot procedure.

**Example for OperatorInput**

```
#2 OperatorInput
   Energy                      // Name of the operator
   H2O GS energy HF/sto-3g     // Description of operator
   prop_no_1.mop 0             // Operator File (index 0 reference value)
   one_mode_grids.mbounds      // Grid bounds file (optional)
```

### #3 ReferenceValues

< *string* = N/A >

Give a filename from which the operator reference values should be read. 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 #3 OperatorInput.

### #3 UseIncludeStatements

If set will use include statements instead of copying contents of header files when making a MidasPot or a MidasDynlib. This will reduce source code file size, but will require headers to be available when compiled.
(For the most portable MidasPot and MidasDynlib source files it is recommended to leave this setting off.)

# Chapter 40

# Operator Reader Input Options

This section outlines the keywords used in the MidasCpp operator reader. This functionality can currently be invoked from either reading a .mop file, or by reading the input after the #3 OperInput. Thus 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). The input from a .mop file is expected to the encapsulated between the #0 MidasOperator and #0 MidasOperatorEnd keywords. One example of the input in a .mop file is given below, this is just to illustrate the input format and does not represent a physical potential of any kind.

**Example for MidasCpp operator file**

```
#0 Midas Operator
   #1 ModeNames
      Q1 Q2
   /* Define Constants which can used in Functions and Operator Terms */
   #1 Constants
      D_e_1 1.1233423
      A_1 1.5424324
      R_e_1 1.134232
      D_e_2 1.49358
      A_2 1.306784
      R_e_2 1.42342
   /* Define re-usable Functions */
   #1 Functions
      Morse(D_e,A,R_e,Q) D_e*(1.0-EXP(-A*(Q-R_e)))^2
   #1 Scaling Factors
      0.121212 0.13417832
   /* Define actual Operator Terms */
   #1 Operator Terms
      0.462352 Morse(D_e_1,A_1,R_e_1,Q)(Q1)
      0.834236 Morse(D_e_2,A_2,R_e_2,Q)(Q2)
#0 Midas Operator End
```

In the section 40.3 alternative operator formats accepted by MidasCpp are outlined. These are not produced by the current version of MidasCpp , but can however still be read and processed by the wave function part of the program.

## 40.1 General keywords

**#<N> Constants**

*< string real >*

.
.
< *string real* >

This keyword is used to define constants used in the #<N> Functions and #<N> OperatorTerms blocks. The input consists of a string, the constant name, and a real, the constant value. The two parts are seperated by a space. See the example at the beginning of the chapter.
(Use this keyword to make the #<N> OperatorTerms less cluttered, when writing your own operator input files.)

### #<N> Functions

< *string string* >
.
.
< *string string* >

This keyword is used to define functions for the use in the #<N> OperatorTerms block. The first string defines a name used to refer to the function, and the second string defines the function itself. The format for the input functions are given in section 12.3 of the appendix. Note that the only whitespace allowed is between the function declaration and the function definition. No whitespace is allowed anywhere else.
(Use this keyword to make the #<N> OperatorTerms less cluttered, when writing your own operator input files.)

### #0 MidasOperator

This keyword is used to signify the beginning of a `MidasCpp` operator input file (.mop). All lines before this keyword will be ignored. The keywords allowed in this type of input file are outlined in the chapter 40.

### #0 MidasOperatorEnd

This keyword is used to signify the end of a `MidasCpp` operator file (.mop). All lines after this keyword will be ignored. This keyword is paired with the #0 MidasOperator keyword

### #<N> ModeNames

< *string* >

This reads an ordered list of the mode names in the operator. All modes present under the #<N> OperatorTerms keyword should be present here. The list might contain modes not present under the #<N> OperatorTerms keyword, but were none the less present in the calculation of this property surface. The order of the mode names are required to be the same as the order of the scaling factors given under #<N> ScaleFactors.

#<N> **OperatorTerms**

< *real string vector* >

.

.

< *real string vector* >

After this keyword the operator is read in the sum over product (SOP) form. Each line is expected to contain a *real* and a vector of strings, seperated by spaces, containing the individual operator terms. The input format for the operator terms are described in sections 12.2 and 12.3. Each of the read terms are expected to contain a set of parenthesis with the name of the mode for which the given term applies to. The input is read in this way until the next keyword is encountered.

**Example for OperatorTerms**

```
#1 Operator Terms
0.5 Q^2(Q0) // The 1-mode term 0.5*Q0^2
...
0.01 Q^1(Q0) Q^1(Q1) // The 2-mode coupling term 0.01*Q0*Q1
```

## 40.2 Advanced keywords

#<N> **ElectronicDof**

< *string vector* >

List of dummy mode names used to define the state specific operator terms. See 12.2.4 for their use.
(Only use this keyword if you are doing multistate calculations.)

#<N> **Frequencies**

< *real vector* >

This reads an ordered set of harmonic vibrational frequencies which are used to scale the modes axies. This scaling corresponts to what is done for the #<N> ScaleFactors keyword, with the exception that the square root of the frequencies provided are used as scaling factors. This keyword exists because of historical reasons and should generally not be used. Instead it is recommended to use the #<N> ScaleFactors keyword.
(The recommendation is to NOT use this keyword, but instead use #<N> ScaleFactors.)

#<N> **IgnoreModes**

< *string vector* >

Reads a list of mode names for modes which should be ignored. If given, any term under #<N> OperatorTerms which includes one or more of the modes on the list will be removed. A file with the suffix "test_removed.mop" will be written to the calculation directory for the operator without the given mode(s).

#<N> **ReferenceValue**

< *real* = 0.0 >

This gives the value of the operator at the origin. This is generally only used for the MidasPot functionality and in the operator files for n-mode expansion-based kinetic energy operators in general coordinate systems (mandatory when using keywords **??** and **??**).

#<N> **ScaleFactors**

< *real vector* >

This gives a set of real factors which are used to scale the individual axis of the potential. The order of the axis are the same as the order given under the #<N> ModeNames It is generelly not advisable to change the values given in this keyword, as doing so will inevitably change the coefficients given in #<N> OperatorTerms. Furthermore any change made to these values will also affect the KEO of the potential, so trust the values given by the potential fit.

#<N> **SetInfo**

< *string* >

This keyword works similarly to the #3 SetInfo keyword.

## 40.3 Alternative operator file inputs

Two examples of alternative operator inputs are the historical Midas OPerator format and the spectroscopic format, which can be recognized by presence of "DALTON_FOR_MIDAS" or "GEN_VSCF_95" being present in the respective files.

### 40.3.1 Alternative Midas OPerator format (Historical)

This type of file either has "DALTON_FOR_MIDAS" or "SCALING FREQUENCIES N_FRQS=" as the first characters of the first line. If "SCALING FREQUENCIES N_FRQS=" is given as the first line, the last part of the line should be an integer corresponding to the number of modes in the operator. After this line should be a number of lines each with a real number corresponding to the harmonic frequency of the ith mode. The scaling used here corresponds to what is described under the #<N> Frequencies keyword. "SCALING FREQUENCIES N_FRQS=" can be omitted, resulting in no scaling being applied to the coefficients under the "DALTON_FOR_MIDAS" tag.

On the line where the "DALTON_FOR_MIDAS" tag is given, one are free to add any comment. Such as is done in the example below, where some notes about the electronic structure calculations are given. After the line containing "DALTON_FOR_MIDAS" 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). Furthermore there is no formal requirement for the order of the terms given for this operator, although one will often see the input divided into 1M, 2M, *ect.* terms.

**Example for Historical `MidasCpp` operator file**

```
SCALING FREQUENCIES N_FRQS=3
7.9619420722044820000000e-05
3.5815003568840300000000e-04
3.6553980092491320000000e-04
DALTON_FOR_MIDAS  MP2(Full)/aug-cc-pVTZ 6F 10F
 8.6835281001659460000000e-06    1
 1.3259972868006448000000e-01    1    1
 1.0889683201219089000000e-02    1    1    1
-2.8621711925145635000000e-02    1    1    1    1
-1.0740042834094813000000e-06    2
 5.0999513029812385000000e-02    2    2
...
 9.7251404904151910000000e-05    3    3    3    3
 4.3802119183971420000000e-02    1    2
-8.4709523449546450000000e-03    1    2    2
-9.0835938209989550000000e-04    1    2    2    2
 5.3293071747605960000000e-03    1    1    2
-4.2508048771674650000000e-03    1    1    2    2
-2.5465590749504868000000e-02    1    1    1    2
 3.7859061678128880000000e-02    1    3
 2.4935829337411217000000e-02    1    3    3
...
 9.4248156073021800000000e-03    2    2    2    3
```

### 40.3.2 Spectroscopic format

For the spectroscopic format 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 $1.92595 \times 10^{-9} q_2^1 q_3^1 q_6^2$.

# Chapter 41

# Midas Molecule Input Options

Standard input for molecular structure information is stored in a file named `Molecule.mmol`.

Input for rectilinear vibrational coordinates is

#0 MIDASMOLECULE

# 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 MIDASMOLECULEEND

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. The masses used by `MidasCpp` can be found in the `data/atomic_masses` file in the main directory of `MidasCpp` .

## 41.1 General keywords

### #0 MidasMolecule

This keyword is used to signify the beginning of a `MidasCpp` molecule file (.mmol). All lines before this keyword will be ignored. The file will be read until the #0 MidasMoleculeEnd keyword is encountered.

### #1 Freq

< *integer string* > [`CM-1`] [`AU`]
< *real string string* >
.
.
< *real string string* >

Reads in the frequency and optionally symmetry and mode label for each vibrational mode of the molecule. The first line contains an interger which gives the number of frequencies to be read and a string which denotes the units of the frequencies, either inverse centimeters (`CM-1`) or atomic units (`AU`). The number of frequencies should be the same as the number of coordinates given under the #1 VibCoord keyword. The following lines contain one frequency each, and optionally two strings. The first of the strings denote which symmetry group the mode belongs to, default being A. The second string gives a name to the mode, which by default will be `Q` plus its position in the list of frequecies counted from zero.

### #1 InternalCoord

< *integer string string* >
< *real string string* >
.
.
< *real string string* >

Defines a set of internal coordinates. The first line specifies the number of coordinates, as well as the units for distances (AU or AA) and angles (RAD or DEG). The lines after that specify the value, the type (distance: R, valence angle: V, dihedral angle: D) and the name of the respective valence coordinate in `MidasCpp` (Q0, Q1, Q2,...). An example input file is given in Section 3.3.1.

### #1 VibCoord

< *string* > [`AA`] [`AU`]

Defines the beginning of the block containing the linear displacement coordinates, where each

set of coordinates are given by the #2 Coordinate keyword. The first line after this keyword contains a string which states that the subsequent displacement coordinates are either given in atomic units (AU) or aangstrom (AA). There should be as many displacement coordinates given here as there are frequencies under the keyword #1 Freq.

**#2 Coordinate**

*< real real real >*

.

.

*< real real real >*

Used to denote each of the rectilinear set of coordinates of the molecule. The displacements are read line by line as displacements for the ith nuclei in the x, y and z coordinates respecively. The order of the blocks are expected to be the same as the order of the frequencies given under #1 Freq. Note that the displament vectors multiplied entry wise by $\sqrt{(m_i)}$, where $m_i$ is the mass of the ith nuclei, should form an orthonormal set.

**#1 Xyz**

*< integer string >* [AA] [AU]

*< string >*

*< string real real real (string string string string string) >*

.

.

*< string real real real (string string string string string) >*

Denotes the beginning of the coordinates of the molecule in the XYZ format. The first line contains the number of atoms in the molecule and the units used, either atomic units (AU) or aangstrom (AA). The following line contains a comment which has no effect on the execution of the program. From line 2 and until line 2 + number of atoms the atomic symbol and positions of the nuclei in the order x, y, z are read in. These lines can furthermore contain up to five optional arguments seperated by spaces.
The optional arguments are used to signify which fragment the atom belongs to and/or which mass to use for the given atom either from a isotopic mass or a user defined mass. The optional arguments are given as a string an equal sign and an integer/number (see example below), the string can either be ISO, MASS, MASSFACTOR, TREAT or SUB each of which are described in the following. ISO signifies the which isotope mass should be used for this nuclei, e.g., ISO=12 for carbon. If this is not specified the mass of the most abundant isotope will be used. Note that changing the mass of a given atom will generally require the generation of a new set of vibrational coordinates. MASS allows the user to specify a specific mass for the given atom. Note that specifying the MASS argument will override the mass from the ISO argument MASSFACTOR signifies that the mass of the atom will be scaled by the specified amount. This scaling will be applied regardless whether ISO or MASS has been specified SUB specifies the subsystem/group/fragment that atom is assigned to. Possible values are integers, e.g., SUB=16. TREAT specifies the treatment in the FALCON scheme, this keyword is coupled with one of four keywords. These keywords are ACTIVE, INACTIVE, FROZEN, CAP, e.g. TREAT=INACTIVE. If modes are set up automatically (see Section 27) prior to any system

modification, this is done only for the subsystems that contain at least one active atom. This is also important for the coupling estimates to `ACTIVE` atoms in the FALCON scheme Groups containing only `INACTIVE` atoms can contribute in the growing FALCON scheme, while groups containing `FROZEN` atoms will not be considered here. `CAP`ping atoms occur in the setup for the double incremental potential energy surface generation and will hardly be used in manual setup.

**Example for Xyz**

```
#1 XYZ
3 AU
D2O(18)
 O  0.0000000000E+00  0.0000000000E+00 -7.3944050027E-01 ISO=18 SUB=1
 H -1.4313404527E+00  0.0000000000E+00  3.6972025014E-01 ISO=2 SUB=1
 H  1.4313404527E+00  0.0000000000E+00  3.6972025014E-01 ISO=2 SUB=1
```

### #0 MidasMoleculeEnd

This keyword is used to signify the end of a `MidasCpp` molecule file (.mmol). All lines after this keyword will be ignored. This keyword is paired with the #0 MidasMolecule keyword.

# Chapter 42

# Midas Metamolecule Input Options

Meta files are used for DIF(ACT) calculations, when a number of different fragment combinations should be provided each with its own `.mmol` file.

## 42.1 General keywords

**#0 MidasMetaMol**

This keyword is used to signify the beginning of a METAMOL file (.mmetamol). All lines before this keyword will be ignored. The file will be read until the #0 MidasMetaMolEnd keyword is encountered.

**#1 Datadir**
< *string* >

Defines the path to the folder containing .mmol files for all fragment combinations.

**#1 Fragments**
< *string* >
.
.
< *string* >

Reads in lines with information about fragment combinations (FCs) used. Each line defines a single FC and contains the following arguments: i) FC name, ii) indices of subsystems, which contribute to the current FC, iii) the number of interconnecting modes, and iv) the number of auxiliary modes. For each specified FC, the corresponding <FC name>.mmol file should be present in the directory specified under #1 Datadir. All FC .mmol files as well as the content of .mmetamol can be generated using the #3 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 FC range is legitimate.

**Example for Fragments**

```
#1 Fragments
FC_0 20,     11    // running DIF(ACT) calculation for 3 FCs.
FC_1 21,     11    // Each FC is composed of a single subsystem
FC_2 22,     11    // (i.e., 20, 21, 22; often referred to as DIF-1F).
```

### #0 MidasMetaMolEnd

This keyword is used to signify the end of a METAMOL file (.mmetamol). All lines after this keyword will be ignored. This keyword is paired with the #0 MidasMetaMol keyword.

# Appendix A

# Additional input description

## A.1 Example for a multi-state operator

Below, the operator from the `mctdh2v_nonadiabatic` test case from the `test` directory is shown. Another example using more states can be found in the `1dho_state_transfer` test case.

```
#0 MidasOperator
   #1 ModeNames
      Q1 Q6 Q9 Q10
   #1 ElectronicDof
      S
   #1 Frequencies
      4.623062562861982548l8E-03 2.7157736358942806929 9E-03
      5.604269005059239749 93E-03 4.185745833942606207 45E-03 1.0
   #1 OperatorTerms
      2.3115312814309912740 9E-03   Q^2(Q1)
      1.3578868179471403464 9E-03   Q^2(Q6)
      2.8021345025296198749 7E-03   Q^2(Q9)
      2.0928729169713031037 2E-03   Q^2(Q10)
     -1.5544955994361038878 5E-02   |0><0|(S)
      1.5544955994361038878 5E-02   |1><1|(S)
      1.8495925181943051202 7E-03   Q^1(Q1)  |0><0|(S)
      3.6036368435154691840 2E-03   Q^1(Q6)  |0><0|(S)
      5.3363665719649323870 8E-03   Q^1(Q9)  |0><0|(S)
      6.2841311466565906093 7E-03   Q^1(Q1)  |1><1|(S)
     -4.9776933556411407169 6E-03   Q^1(Q6)  |1><1|(S)
      1.3766289634722565869 0E-03   Q^1(Q9)  |1><1|(S)
      7.6453254020493396914 1E-03   Q^1(Q10) |0><1|(S)
      7.6453254020493396914 1E-03   Q^1(Q10) |1><0|(S)
     -4.2592444438450222880 1E-04   Q^2(Q10)
      7.9378498694609558373 5E-05   Q^1(Q1)  Q^1(Q6)   |0><0|(S)
     -3.4838341093745308057 3E-04   Q^1(Q1)  Q^1(Q9)   |0><0|(S)
      1.4993716420092919141 6E-04   Q^1(Q6)  Q^1(Q9)   |0><0|(S)
     -2.1902585750920044484 2E-04   Q^1(Q1)  Q^1(Q6)   |1><1|(S)
     -1.1392284534874519863 9E-04   Q^1(Q1)  Q^1(Q9)   |1><1|(S)
      1.3891237271556673054 2E-04   Q^1(Q6)  Q^1(Q9)   |1><1|(S)
      4.0644731276036192733 6E-04   Q^1(Q1)  Q^1(Q10)  |0><1|(S)
```

```
      4.0644731276036192736E-04   Q^1(Q1)  Q^1(Q10)  |1><0|(S)
      7.3498609902416264809E-04   Q^1(Q6)  Q^1(Q10)  |0><1|(S)
      7.3498609902416264809E-04   Q^1(Q6)  Q^1(Q10)  |1><0|(S)
      9.2608248477044487027E-05   Q^1(Q9)  Q^1(Q10)  |0><1|(S)
      9.2608248477044487027E-05   Q^1(Q9)  Q^1(Q10)  |1><0|(S)
#0 MidasOperatorEnd
```

## A.2 Automatic differentiation of general functions

In order to obtain derivatives of function input as described in Section 12.3, the `LibTaylor` library[70] is employed. Derivatives are required in the case of functions which are used in non-linear fitting to the PES given under the #<N1> OptFunc keyword, or when a `MidasCpp` operator file (".mop") is used for generating derivatives through the #2 FilePotential option of the #1 SinglePoint keyword. The use of automatic differentiation through `LibTaylor` gives rise to a number of limitations for the input functions. Firstly, automatic differentiation with `LibTaylor` is not available for expressions comprising `tan` and Delta as well as Heaviside distributions. Secondly, singularities that can arise from dividing by zero will also give rise to errors.

## A.3 Vector Input Unfolding

When a keyword requires a vector as an input, it might be both cumbersome and error prone to provide the whole vector manually. To facilitate an easier input of such vectors, `MidasCpp` offers a number of vector unfolding methods, which can help shortening inputs and avoiding errors.

The most simple of these methods are integer repetition and range expressions. Both of these methods require that the expression itself is encapsulated in brackets (`[]`). Repetition expressions are provided as `[M*N]`, which results in the integer `N` being repeated `M` times. For example, `[6*2]` would be expanded to the integer vector `2 2 2 2 2 2`. In the case of the range expressions, the input is given as `[N..M;S]`, where `N`, `M` and `S` are integers. `N` denotes the starting point of the range, `M` the last point of the range , and `S` the step width. For instance, the expression `[0..4;2]` produces the vector `0 2 4`.

A more advanced method for generating a vector is using `EXPAND(...)` in a vector input. This command takes three arguments separated by commas:

1. the base expression which should be expanded,
2. a variable name, and
3. a vector containing the substitutions used for the variable given in the previous argument.

An example of this type of input is given in the `test pes_Adga_Multistate`. Here, the input to the keyword #<N1> FitFunctions is given as `EXPAND(EXP(-$i*alpha*(Q)),$i,[1..4;1])`, which upon reading by `MidasCpp` will be expanded to the following vector of strings:
`EXP(-1*alpha*(Q)) EXP(-2*alpha*(Q)) EXP(-3*alpha*(Q)) EXP(-4*alpha*(Q))`.

# 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.

- `<VccName>.moped_integrals` !VCC

  One-mode integrals in human readable format.

- `<VccName>.moped_opdef` !VCC

  Operator definition in human readable format.

- `<VccName>.moped_info` !VCC

  Extra operator info such as MCR.

- `<VccName>.moped_sqterms` !VCC

  All operator terms in second quantized format.

- `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_grids.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>.mpoints` !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>.mbar` !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>_bar.mpoints` !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 `prop_ref_values.mpesinfo` 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.mpesinfo` !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.mpesinfo` !ADGA !Static !Taylor

  File holding the inertia tensor for the reference structure.

- `prop_bar_files_navigation.mpesinfo` !ADGA !Static

  File holding information on which blocks of the `prop_no_<Propnr>.mbar` file belongs to which mode or mode combination.

- `prop_ref_values.mpesinfo` !ADGA !Static !Taylor

  File holding the property/properties value/values for the reference structure.

- `sp_restart_<PropNr>.mrestart` !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.

- `prop_files.mpesinfo` !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.

- `coriolis_matrices.mpesinfo` !Operator

  File containing the Coriolis matrices needed for the construction of the approximate or complete Coriolis operator.

# Bibliography


[1] O. Christiansen, D. G. Artiukhin, F. Bader, I. H. Godtliebsen, E. M. Gras, W. Győrffy, M. B. Hansen, M. B. Hansen, M. G. Højlund, N. M. Høyer, R. B. Jensen, A. B. Jensen, E. L. Klinting, J. Kongsted, C. König, S. A. Losilla, D. Madsen, N. K. Madsen, M. Majland, K. Monrad, A. S. Lykke-Møller, L. H. Olsen, K. Rüther, G. Schmitz, P. Seidler, K. Sneskov, M. Sparta, B. Thomsen, S. R. Thomsen, D. Toffoli, A. Zoccante, *MidasCpp*, version 2026.10.0.

[2] D. Lauvergnat, A. Nauts, "Exact numerical computation of a kinetic energy operator in curvilinear coordinates", *J. Chem. Phys.* **2002**, *116*, 8560–8570.

[3] M. Ndong, L. Joubert-Doriol, H.-D. Meyer, A. Nauts, F. Gatti, D. Lauvergnat, "Automatic computer procedure for generating exact and analytical kinetic energy operators based on the polyspherical approach", *J. Chem. Phys.* **2012**, *136*, 034107.

[4] M. Ndong, A. Nauts, L. Joubert-Doriol, H.-D. Meyer, F. Gatti, D. Lauvergnat, "Automatic computer procedure for generating exact and analytical kinetic energy operators based on the polyspherical approach: General formulation and removal of singularities", *J. Chem. Phys.* **2013**, *139*, 204107.

[5] B. Thomsen, K. Yagi, O. Christiansen, "Optimized coordinates in vibrational coupled cluster calculations", *Journal of Chemical Physics* **2014**, *140*, 154102–124101–15.

[6] 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.

[7] E. L. Klinting, C. König, O. Christiansen, "Hybrid Optimized and Localized Vibrational Coordinates", *The Journal of Physical Chemistry A* **2015**, *119*, 11007–11021.

[8] 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.

[9] J. K. G. Watson, "Simplification of the molecular vibration-rotation hamiltonian", *Mol. Phys.* **1968**, *15*, 479 –490.

[10] M. Neff, T. Hrenar, D. Oschetzki, G. Rauhut, "Convergence of Vibrational Angular Momentum Terms within the Watson Hamiltonian", *The Journal of Chemical Physics* **2011**, *134*, 064105.

[11] J. K. G. Watson, "VIBRATION-ROTATION HAMILTONIAN OF LINEAR MOLECULES", *Mol. Phys.* **1970**, *19*, 465.

[12] J. T. Hougen, "Rotational Energy Levels of a Linear Triatomic Molecule in a $^2$ Π Electronic State", *J. Chem. Phys.* **1962**, *36*, 519–534.

[13] 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.

[14] 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.

[15] 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.

[16] C. König, O. Christiansen, "Linear-scaling generation of potential energy surfaces using a double incremental expansion", *J. Chem. Phys.* **2016**, *145*, 064105.

[17] 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.

[18] 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.

[19] 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* **2018**, *148*, 064113.

[20] D. G. Artiukhin, I. H. Godtliebsen, G. Schmitz, O. Christiansen, "Gaussian process regression adaptive density-guided approach: Toward calculations of potential energy surfaces for larger molecules", *The Journal of Chemical Physics* **2023**, *159*, 024102.

[21] N. M. Høyer, O. Christiansen, "Quasi-direct Quantum Molecular Dynamics: The Time-Dependent Adaptive Density-Guided Approach for Potential Energy Surface Construction", *J. of Chem. Theory Comput.* **2024**, *20*, 558–579.

[22] O. Christiansen, "Møller-Plesset perturbation theory for vibrational wave functions", *J. Chem. Phys.* **2003**, *119*, 5773–5781.

[23] J. M. Bowman, "Self-consistent field energies and wavefunctions for coupled oscillators", *J. Chem. Phys.* **1978**, *68*, 608–610.

[24] R. B. Gerber, M. A. Ratner, "self-consistent field review", *Adv. Chem. Phys.* **1988**, *70*, 97.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

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

[34] 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.

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

[36] 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.

[37] O. Christiansen, "Response theory for vibrational wave functions", *J. Chem. Phys.* **2005**, *122*, 194105.

[38] P. Seidler, O. Christiansen, "Vibrational excitation energies from vibrational coupled cluster response theory", *J. Chem. Phys.* **2007**, *126*, 204101.

[39] P. Seidler, M. Sparta, O. Christiansen, "Vibrational coupled cluster response theory: A general implementation", *J. Chem. Phys.* **2011**, *134*, 054119–15.

[40] 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.

[41] 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", *Journal of Chemical Physics* **2012**, *136*, 124101–124101–17.

[42] 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", *Physical Chemistry Chemical Physics* **2013**, DOI 10.1039/C3CP50283J.

[43] 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", *The Journal of Chemical Physics* **2015**, *143*, 134108.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] I. H. Godtliebsen, B. Thomsen, O. Christiansen, "Tensor Decomposition and Vibrational Coupled Cluster Theory", *The Journal of Physical Chemistry A* **2013**, *117*, 7267–7279.

[49] 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.

[50] 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* **2018**, *148*, 024103.

[51] N. Makri, "Time Dependent Quantum Methods for Large Systems", *Annual Reviews in Physical Chemistry* **1999**, *50*, 167–191.

[52] M. Beck, A. Jackle, G. Worth, H. Meyer, "The multiconfiguration time-dependent Hartree (MCTDH) method: a highly efficient algorithm for propagating wavepackets", *Phys Rep* **2000**, *324*, 1–105.

[53] N. K. Madsen, M. B. Hansen, A. Zoccante, K. Monrad, M. B. Hansen, O. Christiansen, "Exponential parameterization of wave functions for quantum dynamics: Time-dependent Hartree in second quantization", *J. Chem. Phys.* **2018**, *149*, 134110.

[54] H.-D. Meyer, U. Manthe, L. S. Cederbaum, "The multi-configurational time-dependent Hartree approach", *Chemical Physics Letters* **1990**, *165*, 73 –78.

[55] N. K. Madsen, M. B. Hansen, G. A. Worth, O. Christiansen, "Systematic and variational truncation of the configuration space in the multiconfiguration time-dependent Hartree method: The MCTDH[n] hierarchy", *The Journal of Chemical Physics* **2020**, *152*, 084101.

[56] N. K. Madsen, M. B. Hansen, G. A. Worth, O. Christiansen, "MR–MCTDH[n]: Flexible Configuration Spaces and Nonadiabatic Dynamics within the MCTDH[n] Framework", *J. Chem. Theory Comput.* **2020**, *16*, 4087–4097.

[57] M. B. Hansen, N. K. Madsen, A. Zoccante, O. Christiansen, "Time-dependent vibrational coupled cluster theory: Theory and implementation at the two-mode coupling level", *The Journal of Chemical Physics* **2019**, *151*, 154116.

[58] N. K. Madsen, A. B. Jensen, M. B. Hansen, O. Christiansen, "A general implementation of time-dependent vibrational coupled-cluster theory", *The Journal of Chemical Physics* **2020**, *153*, 234109.

[59] N. K. Madsen, M. B. Hansen, O. Christiansen, A. Zoccante, "Time-dependent vibrational coupled cluster with variationally optimized time-dependent basis sets", *The Journal of Chemical Physics* **2020**, *153*, 174108.

[60] C. d. Boor, *A Practical Guide to Splines*, Springer Verlag, New York, **1978**.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] C. E. Rasmussen, C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*, The MIT Press, **2005**.

[66] M. Riedmiller, H. Braun, RPROP - A Fast Adaptive Learning Algorithm, tech. rep., Proc. of ISCIS VII), Universität Karlsruhe, **1992**.

[67] C. Igel, M. Hüsken in Proceedings of the Second International Symposium on Neural Computation, **2000**, pp. 115–121.

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

[69] 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.

[70] U. Elkström, "LibTaylor", https://github.com/uekstrom/libtaylor.