Contents

List of Tables VI
List of Figures VII
List of Examples VIII
Copyright IX

1 Introduction 1

2 An MCTDH tutorial 3
   2.1 Determining the absorption spectrum for the photodissociation of NOCl . . . 3
   2.2 Determining state populations for the photo-excitation of pyrazine . . . . 7
   2.3 Determining reaction probabilities for the exchange reaction of H+H₂ . . . 9
   2.4 Determining the vibrational spectrum of LiCN . . . . . . . . . . . . . . . . . 10
   2.5 Determining the vibrational spectrum of CO₂ by filter-diagonalisation . . . 12
   2.6 Determining eigenstates by improved relaxation . . . . . . . . . . . . . . . . . 14
   2.7 Determining eigenstates by block improved relaxation . . . . . . . . . . . . 15
   2.8 Using potfit and chnpot to fit a surface to ab initio data points . . . . . . . 18
      2.8.1 Transforming the ab initio data to product form . . . . . . . . . . . . . . 18
      2.8.2 Interpolating the natural potential to a new primitive grid . . . . . . . 18
   2.9 Optimizing an external field with Optimal Control Theory (OCT) . . . . . . . 19
   2.10 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3 Defining the type of calculation to be made 21
   3.1 Specifying the task for MCTDH . . . . . . . . . . . . . . . . . . . . . . . . . . 21
   3.2 Specifying the desired output . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
   3.3 Propagating a wavepacket . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
   3.4 Relaxing a wavepacket to produce the lowest eigenstate . . . . . . . . . . . . 23
   3.5 Improved relaxation. Generation of excited eigenstates * . . . . . . . . . . . 24
   3.6 Performing a numerically exact calculation . . . . . . . . . . . . . . . . . . . 27
   3.7 Diagonalising the Hamiltonian using the Lanczos algorithm . . . . . . . . . . 28
   3.8 Starting a calculation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
   3.9 Continuing or stopping a calculation * . . . . . . . . . . . . . . . . . . . . . . 29
   3.10 Using parallel shared memory hardware * . . . . . . . . . . . . . . . . . . . . 29
3.11 Using parallel distributed memory hardware *

4 Selecting a DVR/FBR-representation for the primitive basis 36
4.1 Available DVR/FBR-representations 36
4.2 Hermite and radial Hermite DVR 36
4.3 Legendre DVR 38
4.4 Sine DVR 39
4.5 Exponential DVR and fast Fourier transform 39
4.6 Spherical harmonics FBR * 41
4.7 Restricted Legendre DVR * 41
4.8 Extended Legendre DVR and Two-Dimensional Legendre DVR * 42
4.9 Three-Dimensional rotational DVR * 43

5 Defining the single-particle basis 45
5.1 Specifying the number of single-particle functions 45
5.2 Selecting degrees of freedom from a large system * 46
5.3 Combining modes to produce multi-dimensional single-particle functions 46

6 Setting up the Hamiltonian 48
6.1 The operator file 48
6.2 Defining numerical constants 49
6.3 Using symbolic expressions to define the Hamiltonian 51
6.4 Defining labels 53
6.5 Implementing user-defined 1D-operators 54
6.6 Defining new symbolic expressions * 54
6.7 Implementing separable potentials * 56
6.8 Implementing non-separable potentials (potential surfaces)* 58
6.9 Incorporating natural potentials 61
6.10 Using complex absorbing potentials (CAPs) 62
6.11 Altering a Hamiltonian from input file or command line * 64
6.12 Setting up auxiliary operators * 66
6.13 DOF, mode, and mulf potentials 67
6.14 Golden rules for writing operator files 69

7 Generating the initial wavepacket 71
7.1 Building Gaussian functions as initial functions 71
7.2 Setting up Legendre functions as initial functions 72
7.3 Setting up extended Legendre functions as initial functions 73
7.4 Generating spherical harmonics as initial functions 73
7.5 Generating Wigner functions as initial functions 74
7.6 Generating eigenfunctions of a one-dimensional Hamiltonian 75
7.7 Reading the initial wavepacket from file 76
7.8 Diagonalising a multi-dimensional operator to create multi-dimensional SPFs* 77
7.9 Generating an initial wavepacket using an operator* 77
7.10 Creating a set of initial wavepackets * 78
7.11 Setting up (a)diabatically corrected initial wavepackets * .......................... 78

8 Choosing an integration scheme ......................................................... 80
8.1 Using the VMF integration scheme in an MCTDH calculation .................. 80
8.2 Using the CMF integration scheme in an MCTDH calculation .................. 81
8.3 Description of the available integrators ........................................... 82
8.4 Fine-tuning the integration .............................................................. 84
8.4.1 Propagating in natural or interaction picture orbitals * ....................... 84
8.4.2 Suitable integrator settings for improved relaxation ......................... 84
8.4.3 Evaluating potentials using the TDDVR or CDVR method * .................. 85

9 Treating non-adiabatic systems ......................................................... 86
9.1 Setting up the Hamiltonian for a non-adiabatic system ......................... 86
9.2 Defining the primitive basis for a non-adiabatic system ....................... 87
9.3 Defining the single-particle basis for a non-adiabatic system ................ 87
9.4 Building the initial wavepacket for a non-adiabatic system .................... 90

10 Treating bosonic systems ............................................................... 91
10.1 Setting up the Hamiltonian ............................................................ 91
10.2 Modifying the input ........................................................................ 92

11 Analysing the results employing the Analyse programs ....................... 96
11.1 The Analysis Interface ................................................................... 96
11.2 Interpreting the MCTDH output ....................................................... 97
11.3 Checking the accuracy of a calculation ............................................. 99
11.3.1 Checking the primitive basis size ................................................. 99
11.3.2 Checking the single-particle function basis size ............................. 101
11.4 Checking the efficiency of a calculation .......................................... 102
11.5 Watching the system’s evolution ..................................................... 102
11.6 Determining photo-dissociation and photo-absorption spectra ............... 104
11.7 Computing excitation and reaction probabilities .................................. 105
11.8 Monitoring state populations of non-adiabatic systems ...................... 107
11.8.1 Diabatic populations .................................................................. 107
11.8.2 Adiabatic populations computed with adpop .................................. 107
11.8.3 Adiabatic populations computed with adproj ................................ 108
11.9 Plotting 2D cuts through the system density ..................................... 111
11.10 Plotting cuts through the potential energy surfaces .......................... 112

12 Using the Potfit program ................................................................. 114
12.1 Transforming a potential to product form ........................................ 114
12.2 Using ab initio data ....................................................................... 118
12.2.1 Using ab initio data directly with the mctdh program .................. 118
12.2.2 Using the potfit program ......................................................... 119
12.3 Extra flexibility, combining potfit and chnpot .................................. 120
12.3.1 Dealing with an arbitrary primitive grid .................................... 120
### Contents

12.3.2 Transforming between two natural potentials with \texttt{chnpot}  
12.4 Manipulating potentials with the \texttt{projection} program *  
12.4.1 Input and output files *  
12.4.2 Generating a Fourier-transformed potential *  
12.4.3 Using a Fourier-transformed potential in MCTDH *  
12.5 Downsizing previous \texttt{potfits}: the \texttt{cutnpot} and \texttt{rdnpot} functions  

13 Using the Monte-Carlo Potfit program  
13.1 Monte-Carlo Potfit  
13.2 Compiling  
13.3 Selecting the sampling method  
13.3.1 Sampling methods  
13.3.2 Remark on the Metropolis algorithm  
13.3.3 Choosing a sampling method  
13.3.4 Re-using and pre-sampling trajectories  
13.3.5 Re-using densities or SPP  
13.4 Solving for the coefficients  
13.4.1 Using conjugate gradients to solve for the coefficients  
13.5 Reducing memory consumption  
13.6 Restoring molecular symmetries  
13.6.1 Simple symmetries  
13.6.2 Coordinate-based expressions  
13.6.3 Periodic boundary conditions  
13.6.4 The \texttt{usersym} keyword  
13.6.5 Using intermediate extended grids  
13.6.6 Symmetry checking  
13.7 Implementing a surface outside the MCTDH operator library  
13.8 Checking convergence and fit quality  
13.8.1 Convergence of the SPP  
13.8.2 Convergence of the Coefficients  
13.8.3 Testing the fit with other trajectories  
13.9 Output files  

A The concept of the input file  

B The Structure of the Programs  

C The built-in symbolic expressions  

D Structure of the WF array  

E Installing the MCTDH package  

F The \texttt{svn}-repository of the Heidelberg MCTDH package  
  F.1 Useful \texttt{svn} commands
List of Tables

2.1 Vibrational energies of CO$_2$ computed with MCTDH/FD. .......................... 14
4.1 Available DVR/FBR-representations for the primitive basis. ......................... 37
6.1 Selection of built-in symbolic expressions. ............................................ 52
8.1 Available integrators in dependence of the calculation type ...................... 81
8.2 Optimal orders for the ABM and BS integrators .................................. 83
9.1 Built-in symbolic expressions for non-adiabatic systems .......................... 87
11.1 FWHM values of the window functions $\tilde{g}_k$ times the length of the autocorrelation function. Remember that the length of the autocorrelation function is twice the propagation time, if the t/2-trick is used. .......................... 104
13.1 Sampling methods and parameters ................................................... 133
13.2 Solvers for the coefficients. ........................................................... 135
13.3 Preconditioners for conjugate Gradients. ........................................... 136
13.4 Output files of mcpotfit .................................................................. 146
A.1 Input sections required for different calculation types .............................. 147
A.2 Description of the calculation types. .................................................. 148
C.1 Simple one-dimensional operators ...................................................... 152
C.2 Operator symbols which require no arguments ...................................... 153
C.3 One-dimensional operators which require arguments ............................. 159
C.4 One-dimensional potential energy curves ............................................. 160
C.5 Two-dimensional operators (surface scattering) ..................................... 161
C.6 Two-dimensional operators ($C_+, C_-$) ............................................... 161
C.7 Some general multi-dimensional operators .......................................... 161
C.8 One-dimensional operators (Rf, Rfm, hKEh, hFRh, hdqh, hdqR, dqR, dq2R) 162
C.9 Matrix operator symbols, used for an electronic degree of freedom ........ 164
List of Figures

2.1 The NOCl $S_1$ absorption spectrum ........................................... 4
2.2 Overlay Plot ................................................................................. 6
2.3 Diabatic state populations of pyrazine ........................................... 7
2.4 The pyrazine $S_2$ absorption spectrum ......................................... 8
2.5 H+H$_2$ reaction probability .......................................................... 10
2.6 The vibrational spectrum of LiCN .................................................... 11
2.7 The vibrational spectrum of CO$_2$ .................................................. 13
11.1 The natural orbital populations as a function of time for NOCl .......... 102
11.2 The density as a function of time for NOCl ...................................... 103
11.3 Total and projected flux of dissociating NOCl ................................. 106
12.1 Main concepts involved in the usage of \textit{ab initio} data with the MCTDH package 121
12.2 Jacobi coordinates for a 4-atomic system ...................................... 125
B.1 The structure of the MCTDH programs ......................................... 150
D.1 Structure of wave function ............................................................ 165
## List of Examples

4.1 An input file for a wavepacket propagation of NOCl .................. 38
6.1 An operator file for the NOCl S$_1$ state .......................... 49
6.2 An operator file for a propagation using the modified Henon-Heiles Hamiltonian .................................................. 52
6.3 A parameter file for the Henon-Heiles Hamiltonian ................. 64
6.4 A surface file for the NOCl S$_1$ potential .......................... 65
9.1 An operator file for the pyrazine 4-mode 2-state model system ... 88
9.2 An input file for the pyrazine 4-mode 2-state model system ..... 89
10.1 An operator file for $N = 3$ one-dimensional bosons in a harmonic trap .................. 94
10.2 An input file for $N = 3$ one-dimensional bosons in a harmonic trap .......................... 95
11.1 The analysis startup menu ............................................. 97
11.2 An output file from a wavepacket propagation of NOCl .......... 98
11.3 An input file for a potfit calculation ............................... 109
11.4 The input file for the mctdh-genoper-run ............................ 110
11.5 An operator file for a projection operator .......................... 110
11.6 The showsys menu ....................................................... 111
12.1 A potfit input file for the NOCl S1 surface .......................... 115
12.2 A potfit output file for the NOCl S1 surface ....................... 117
12.3 A projection input file for the BMKP surface for (H$_2)_2$ ......... 123
12.4 An operator file showing the use a Fourier-transformed potential .... 127
13.1 SAMPLING-SECTION in MC-Potfit ............................... 131
13.2 SAMPLING-SECTION in MC-Potfit ............................... 132
13.3 Symmetry operations within the Zundel cation ($D_{2d}$) ........... 137
13.4 Symmetry operations with periodic boundary conditions .......... 139
13.5 Excerpt from an output file for the Zundel cation ($D_{2d}$) ........ 143
13.6 An input file for npotminmax ......................................... 144
A.1 An input file for a propagation using the Henon-Heiles Hamiltonian .......... 149
Copyright


Permission is granted to use and copy this software and its documentation. Further distribution requires the agreement of the authors. Permission to modify the software is granted. The authors would welcome if additions and bug fixes are made available to them for inclusion in future releases of the package.

This software is provided “as-is” and without warranty of any kind.

Acknowledgements

The very first MCTDH program, later called version 1, was written by Uwe Manthe as part of his PhD work in Heidelberg. The Heidelberg MCTDH package was created by G. A. Worth, M. H. Beck, A. Jäckle, and H.-D. Meyer. Over the years several graduate students, post-docs and visitors have made contributions to the MCTDH package. We list them in chronological order: M. Ehara, M.-C. Heitz, A. Raab, S. Wefing, S. Sukiasyan, C. Cattarius, F. Gatti, F. Otto, M. Nest, A. Markmann, M. R. Brill, O. Vendrell, M. Schröder, D. Pelaez-Ruiz, Phillip S. Thomas, and David Mendive-Tapia. We are very, very grateful to all of them!

Citations

When citing the MCTDH program package in the literature, the following citation should be used:


Current versions: 8.4.17 and 8.5.10 (2019).

(Rather than “Current versions” you should give “Used version”)

A comprehensive description of the methods incorporated in the programs is in:


The original paper is:

The Multi-Layer (ML) extension is described in:


These three papers should be cited as well. You may further wish to include the references


A list of publications on the MCTDH method itself and on applications of MCTDH is given at the end of this Guide. The latest version of this list can be found on the MCTDH homepage:

http://mctdh.uni-hd.de

From this URL a review on the MCTDH scheme, Ref. [1], and the MCTDH feature article, Ref. [5], [7], as well as other articles can be downloaded. There you will also find a small bibtex file (mctdh.bib) which contains references to several MCTDH articles. This is for your convenience.
Chapter 1

Introduction

The MCTDH method is an efficient algorithm for the solution of the time-dependent Schrödinger equation. For a full description of the theory see the review [1]. You may also wish to read the MCTDH book [6]. The MCTDH program has been developed to perform quantum mechanical wavepacket propagations employing this method. All the options and variants of the MCTDH method presented in the review are implemented. Furthermore, the MCTDH program can be used to propagate wavefunctions numerically exactly and to diagonalise a Hamiltonian by the Lanczos algorithm. A variety of programs included in the MCTDH package serve to analyse the results of a calculation and compute observable quantities, which can directly be plotted with the help of GNUPLOT scripts.

The installation of the MCTDH package is described in Appendix E.

This documentation is intended to help the user by explaining, with many examples, how to set up and run a calculation and analyse the results. For a calculation the Hamiltonian operator and the input parameters must be defined. This is done in two ASCII files, named operator and input file, which must have .op and .inp, respectively, as extension. The required data is put in as keywords. In both files, the keywords are grouped together into sections, each with a specific set of information. The sections start with a line containing the keyword XXX-SECTION, and end with END-XXX-SECTION, where XXX is the name of the section. Everything following a # is treated as comment.

How to set up the operator and input file will be detailed in the following chapters. Note, however, that this Guide does not claim to be complete. Although the majority of options of the MCTDH package — and in particular those being most important for your daily work — is described, there are probably still options useful for you that are not documented here. For the full list of options, see therefore the HTML manual. The HTML manual also describes the installation process.

Some parts of the User’s guide are labelled as advanced topics, indicated by a “*” in the table of contents. These parts contain information on features of the MCTDH package that make the programs more convenient to use but do not extend their functionality. The advanced topics also deal with options of the MCTDH package which are needed in special cases only. You may skip these parts until you got more experienced with the MCTDH package.

Please keep in mind the following typographical conventions which are designed to help you reading the User’s Guide:
**Typewriter** The typewriter font is used for literal characters, such as keywords and labels given in the input files, the names of routines and variables, and extracts of the source code.

**Italics** The italics font indicates arguments which are supposed to be substituted by the user.

**Bold face** Bold face emphasises the names of programs and scripts in the MCTDH Package, and their options.

**Sans serif** The sans serif font is employed for files, directories, and paths.

**UPPERCASE** The different sections that arrange the input and operator files are given in uppercase.

**SMALL CAPS** Small capital letters are used for the names of persons as well as programs that are not part of the MCTDH package.
Chapter 2

An MCTDH tutorial

When you have successfully installed the MCTDH package (see Appendix E), you have various programs in the field of multi-dimensional quantum dynamics at your disposal. Before we go into the details of how to use these programs, we would like to invite you to a short tour of the MCTDH package, by performing some exemplary calculations. On this trip you will get an overview of the opportunities the MCTDH package offers. The tour shall also demonstrate the ease of employing the program and give you an impression of the efficiency of the code.

First set up and move to a suitable directory in which to run the tutorial calculations (e. g. $MCTDH\_DIR/tutorial or $HOME/tutorial), then follow the instructions below. The tutorial uses standard problems. Once a calculation has been made, try to understand the input files, they can be used as templates for other calculations. The expression $MCTDH\_DIR occurring in the following examples stands for the path of the MCTDH-directory.

2.1 Determining the absorption spectrum for the photodissociation of NOCl

The photodissociation of NOCl is a simple photo-chemical reaction. After excitation from the ground to the first excited state, $S_0 \rightarrow S_1$, the chlorine atom dissociates on a femto-second time-scale. This results in a broad band for the absorption spectrum. This system was used for the first application of MCTDH to a realistic system [4].

The calculation consists of two stages. First, the ground state wavefunction is generated by energy relaxation of an initial guess wavefunction on the ground state surface, $S_0$. The second stage then places this wavepacket on the excited state surface, $S_1$, leading to photodissociation.

1. Copy the files $MCTDH\_DIR/inputs/nocl0.inp and $MCTDH\_DIR/inputs/nocl1.inp to your tutorial directory, and create there the directories nocl0 and nocl1.

2. To perform the ground state relaxation calculation, type

```
mctdh85 nocl0
```

You will have to wait about 2 seconds. (The timings given in this manual are for a 3 GHz PC running under Linux).
3. To perform the photo-dissociation calculation, type

\`mctdh85 noc1\`

This will again take about 2 seconds.

NB There is now the option \`-mnd\` (make name directory) which allows you to skip the creation of the name directory. E.g.

\`mctdh85 -mnd noc1\`

will make the name directory before starting the calculation.

The calculation can now be analysed. Move to the directory \`noc1\` which contains all the data files from the propagation.

1. To watch the system dissociating, type

\`showd1d85 -a -M -y 5 -sm f1\`

In order to understand the options and parameters, type \`showd1d85 -h\` and see the HTML documentation. Try the other format options (-S, -T) and inspect the motion of the other degrees of freedom (f2, f3). The program showd1d also supports interactive plotting. Start the program with

\`showd1d85 -inter\`

and follow the menu options to select and alter the plot.

2. To plot the spectrum, type
2.1 Determining the absorption spectrum for the photodissociation of NOCl

autospec85 -g 1 0.6 2.0 ev 0.0 1
gnuplot -persist spectrum.pl

The first line produces a GNUPLOT file with data to plot the spectrum. This is done from 0.6 eV to 2.0 eV and using a simple cosine cutoff function to allow for the finite propagation time. The result is shown in Fig. 2.1. In order to understand the options and parameters, type autospec85 -h and see the HTML documentation. Note that the spectrum shown is the Fourier-transform of the autocorrelation function times the energy. Hence it is assumed that the ground state energy is at zero, such that energy equals excitation energy. If this is not the case, use option -e to shift the energy scale. The -FT option suppresses the multiplication with the energy, showing directly the Fourier-transform. (NB The option -FT is now default. Use option -EP to switch on the energy prefactor, or use -Mb <dipole-moment>, to plot the properly normalized absorption spectrum in mega barns.)

To make life easier, there exist a number of bash scripts (so called pl-scripts) which automatically call an analyse routine and plot the results. The above commands are equivalent to

plspec 0.6 2.0 ev

One may alternatively call plspec without arguments. The script will then prompt you for the missing input. Finally, the command plauto plots the autocorrelation function, the command plnat plots the natural populations, pldq plots the expectation values of the coordinates, and the commands plupdate, plupdate -e, and plspeed show information on the performance of the integration. plall prints a list of all pl-scripts, but for more information see the HTML documentation. Note that all pl-scripts support the -h option. We do recommend the use of the pl-scripts!

The program showsys85 is a powerful tool for plotting 1D and in particular 2D views on wavepackets and potentials. To plot the potential one first has to generate a so called pes file. To do so, move up to the tutorial directory, where nocl1.inp is located, and type

mctdh85 -pes nocl1

This will generate the files pes, log.pes, and op.log.pes in the nocl1 name-directory. The pes file is an operator file in which all terms containing derivative operators or CAPs are deleted. The WARNING message which appears can be ignored. It just tells you, that the mctdh program will not perform a propagation, although there is a keyword propagation in the input file. Now move back to the name-directory and type

showsys85

A menu appears (see Example 11.6), which allows various options to be set. Go to menu point 10 (type 10), and change the plot task to 2 = plot pes (type 2). Next input a 1 three times and a 2D cut through the surface (with theta fixed to 1.545 radians) will pop up. Now use menu point 20 = change coordinate section, i.e. chose another cut. If one gives x and two numbers, a 1D plot will appear. After you have played around enough, go back to menu point 20 and input

x y 2.1

Then use menu point 5. You will be asked for a file name. Chose any convenient name, e. g. xyz. The plot data is then written to the file xyz for later use.
Next we want to inspect the wavefunction. Go to menu point 10 and chose $5 = \text{plot reduced density}$. The density, i.e. $|\Psi|^2$ integrated over all coordinates, except those specified by $x$ and $y$ (that is integrated over all angles in the present case), will be shown. Input a 1 three times and you will see the initial density. Pressing RETURN will display the density propagated by one time step, and so on. After you have returned to the menu, chose point $400 = \text{Overlay plots}$ and then $410 = \text{File for overlay}$ and enter the file name (xyz). After inputting 1’s you will now see an overlay plot, i.e. the wavepacket on top of the contour lines of the potential.

With menu points 240 and 245 one may switch off the legend (or keys) and the title. Menu point 285 allows to take larger time steps and with point 280 one may switch to different plot forms, e.g. to plot all time slices at once. Such a plot is shown in Fig. 2.2.

Inspect the ASCII files of the name directories, in particular output, log, and timing. The file input contains a copy of the input file, the options, and the operator file. Thus, it tells you exactly what you have been doing. Since an NOCl run is so fast, NOCl is ideally suited for testing. Just play around with it! You may e.g. change the numbers of single particle functions or alter the integrator accuracies. You also may try the options, e.g. to start a continuation run type:

\[
\text{mctdh85 -c -tfinal 50 nocl1}
\]

Type \texttt{mctdh85 -h} to obtain the list of options.
2.2 Determining state populations for the photo-excitation of pyrazine

The pyrazine molecule contains a classic example of vibronic coupling. Two states, which are close in energy, are coupled by motion along one vibrational mode, resulting in a broad spectrum for the upper state. This system can be described using the simple vibronic-coupling model Hamiltonian.

The vibronic-coupling model Hamiltonian is well suited to the MCTDH method, being already in the product form required for maximum efficiency. For further details of this system, see Refs. [8–11], and the references therein.

In this tutorial, we use a simple 4-mode 2-state model. This qualitatively reproduces the experimental spectrum after the addition of phenomenological broadening. The calculation takes the ground state wavefunction (here a simple product of gaussians as the ground state surface is harmonic), and places it on the S_2 excited surface. Propagation then takes place, and rapid population transfer to the S_1 state is observed. Finally, the spectrum of the model system is calculated.

1. Copy the file $MCTDH_DIR/inputs/pyr4.inp, and create the directory pyr4.

2. To perform the photo-excitation calculation, type

   mctdh85 pyr4

   This will take about 20 seconds.

   The calculation can now be analysed. Move to the directory pyr4 which contains all the data files from the propagation.

Figure 2.3: The diabatic state populations of the pyrazine molecule after excitation to the S_2 state, calculated using a 4-mode model.
1. To plot the diabatic state populations, type

```
rdcheck85 -g 1 0
gnuplot -persist chk.pl
```

or, more simply, type

```
plstate
```

The result is shown in Fig. 2.3. Note that very fast transfer occurs to the $S_1$ state. At around 80 fs the system returns to the conical intersection connecting the two states, and a second transfer of population occurs.

2. To plot the spectrum, type

```
autospec85 -g 1 -e -0.2258 eV -1.0 1.0 eV 30 1
gnuplot -persist spectrum.pl
```

The first line produces a GNUPL OT file with data to plot the spectrum from -1.0 eV to 1.0 eV. An energy shift of 0.2258 eV has been added due to the zero point energy of the system. A phenomenological broadening with a relaxation time of 30 fs has also been added. The result is shown in Fig. 2.4. Again, the same figure is generated more simply by typing

```
plspec -e -0.2258 eV -1.0 1.0 eV 30 1
```
2.3 Determining reaction probabilities for the exchange reaction of H+H₂

The H+H₂ system is the smallest reactive molecular system, but it is the prototype of all three atom reactions. As interaction potential we will use the LSTH potential energy surface. This is a full 3D surface and as such must be first transformed to MCTDH product form. The Potfit program can accomplish this fast and reliably (at least as long as the full primitive product grid is not too large). After the wavepacket is propagated the reaction probability is determined by flux analysis. See the MCTDH review [1] or the original publication [12] for more details. Here we will perform a scattering calculation for vanishing total angular momentum \((J = 0)\) only. Thus the result is a initial-state selected reaction probability and not a cross section.

1. Copy the files $MCTDH\_DIR/pinputs/lsth.inp and $MCTDH\_DIR/inputs/hh2.inp to your tutorial directory, and create the directories lsthfit and hh2.

2. To perform the potential fit calculation, type

   potfit85 lsth

   This will take about 5 seconds.

3. To perform the scattering calculation, type

   mctdh85 hh2

   This will take less than 5 minutes.

4. To perform the flux analysis, move to the directory hh2 and type

   flux85 -e lsth 0.4 2.0 ev rv

   This will take less than 10 seconds. The option \(-e \text{lsth}\) sets the zero point of the energy to the minimum of the H₂ potential curve. The other arguments set the energy interval to 0.4 – 2.0 eV and select the rv-CAP for analysis.

   The results of the calculation can now be inspected. Type

   plflux

   and you will see the reactive flux, i.e. the quantum flux going into the rv–CAP, and the energy distribution of the initial wavepacket. The reaction probability is just the quotient of these two data sets. It can be seen by typing

   plflux -r

   The results are shown in Fig. 2.5. One may compare them with those of reference [13].

   Inspect the ASCII files of both name directories, lsthfit and hh2. The Potfit program will be described in more detail later in this guide. The motion of the wavepacket can again be visualised with the aid of showd1d85. In particular the \(\theta\) degree of freedom is interesting. Type
Figure 2.5: This picture shows the reaction probability of the system H+H₂(ν=0, j=0) for total angular momentum J=0 (solid line), which is the quotient of the quantum flux going into the rv-CAP (dashed line) and the energy distribution of the initial wavepacket (dotted line).

showd1d85 -a -y 10 f3
and repeatedly press RETURN to step through the pictures. Initially the molecule is in the j = 0 rotational state and the density is evenly distributed over all angles. After about 20 fs the wavepacket reaches the saddle-point region and the system is in the transition state. The transition state is collinear and consequently the angular distribution is now strongly peaked at zero degrees. At later times a more evenly angular distribution is again assumed. You may also inspect the motion of the other two degrees of freedom.

2.4 Determining the vibrational spectrum of LiCN

The MCTDH program is not only capable of propagating wavepackets but also of diagonalising a Hermitian Hamiltonian operator, by employing the Lanczos algorithm. The time-independent Schrödinger equation is then solved rather than the time-dependent one. This feature, and similarly the possibility of performing a numerically exact propagation, has been implemented into the mctdh package because then the very convenient operator generation is available for these tasks. Lanczos diagonalisation and exact propagation are, of course, possible only for comparatively small problems.

As a small example of this feature let us determine the vibrational spectrum of a two-dimensional model of the LiCN electronic ground state, with the CN bond length frozen at its equilibrium value. The initial wavefunction is chosen arbitrarily; the intensities thus have no physical meaning. To keep the CPU time short only a small number of Lanczos iterations will be made. The number of iterations is sufficient to converge the lowest 0.5 eV of the spectrum.
2.4 Determining the vibrational spectrum of LiCN

1. The LiCN surface is not linked by default. It must be first linked to the program by re-compiling MCTDH.

   compile -i licn mctdh

   It might be that you need to copy licnsrf.f from the addsurf directory to source/surfaces. Alternatively you may set a link (run the script mklinks). Type mctdh85 -ver to inspect, which surfaces are included. See the HTML documentation Installation and Compilation / Compiling the Programs and Hamiltonian Documentation / Available Surfaces for more details.

2. Copy the file $MCTDH_DIR/inputs/licn.inp, and create the directory licn.

3. To diagonalise the Hamiltonian, type

   mctdh85 licn

   This will take less than 20 seconds.

   The calculation can now be analysed. Move to the directory licn which contains all the data files from the diagonalisation. The eigenvalues, intensities and error estimates for the energies are stored in the ASCII file eigval.

1. To see the results, type e.g.

   less eigval

   The first line describes the entries of the eigval file.
2. To plot the spectrum, type

pleigval -a -6.6 -x -6.1

This displays the spectrum in the converged energy range. The result is shown in Fig. 2.6. Note that energies with very small intensities are not visible. To display all lines, add the option -1 in order to use a logarithmic scale for the intensities.

2.5 Determining the vibrational spectrum of CO$_2$ by filter-diagonalisation

To Fourier transform the autocorrelation function is the straightforward procedure to extract eigen-energies from a time evolved wavepacket. This, however, requires a very long propagation time $T$ as the resolution improves only like $\hbar/T$. This limit, set by the uncertainty relation, can be overcome when employing the filter-diagonalisation (FD) method introduced by Neuhauser. Our particular version of the FD method is discussed in Refs. [14, 15].

The following example shall show how filter-diagonalisation and MCTDH-propagation can be combined. The example is similar to the problem studied in Ref. [15], however, here we sacrifice some accuracy in order to gain speed.

1. Copy the file $\text{$MCTDH$DIR/inputs/co2t.inp}$, and create the directory co2.
2. Copy the file $\text{$MCTDH$DIR/inputs/co2ft.inp}$ to the directory co2.
3. To perform the MCTDH propagation, type

mctdh85 co2t

This will take less than 2 minutes.

As done in the previous examples, you should study the log, output, timing, etc files and investigate natural– and grid–populations. In particular it is useful to investigate the spectrum. Thus type

plspect -e -2534.52981 cm$^{-1}$ -200 7000 cm$^{-1}$

The option -e -2534.52981 cm$^{-1}$ shifts the zero point of the energy scale by -2534.52981 cm$^{-1}$ which is the ground state energy. Thus, the ground state is now expected at zero. Try the options -g 0, -g 1, and -g 2 and you will understand, why -g 1 is the default. The plot depicts the spectral lines having a width of almost 100 cm$^{-1}$. This demonstrates that a precise determination of eigen-energies by Fourier transform of the autocorrelation function is difficult. (See Fig. 2.7)

To continue with the tutorial move to the co2 directory and type

filter85 co2ft

This runs the filter-diagonalisation and creates the files filter.eig, filter.inp, and filter.log. The file filter.inp repeats the input file, but additionally shows all default and computed parameter values. The file filter.log displays what filter85 has been doing. It also contains a list of all computed eigenvalues and intensities. The file filter.eig again contains the computed
2.5 Determining the vibrational spectrum of CO$_2$ by filter-diagonalisation

![Figure 2.7: The vibrational spectrum of CO$_2$ as obtained by Fourier transform of the autocorrelation function and by FD using the same autocorrelation function. For better visibility, the Fourier spectrum is shifted upwards by 50 units.](image)

...eigenvalues, but omits those which lie outside the energy window or which are detected as spurious according to an internal error measure.

Unfortunately, the file filter.eig may still contain spurious eigenvalues. These are detected by performing several filter-diagonalisations with slightly different parameters and keeping only those eigenvalues which are stable. To perform the additional filter-diagonalisation runs, type

```plaintext
filter85 "file_outputname=f1,window_energypoints=150" co2ft
filter85 "file_outputname=f2,vp_principle = 1/H,filter_function=box" co2ft
```

The double quoted arguments of `filter85` overwrite the values from the input file. We used a different number of energy points (150 rather than 125) and a different variational principle ($1/H$ rather than $H$) together with a different damping function (box rather than cos) as compared to the first filter-diagonalisation. (See the HTML documentation for details). The output files are now `f1.*` and `f2.*`, respectively. A list of the stable eigenvalues together with an error estimate based on the spread of the eigenvalues is produced by the command

```plaintext
fdmatch85 filter.eig f1.eig f2.eig | sort -n | fdcheck85 3 0 > results
```

You may inspect this data, cat results, and plot a stick spectrum

```plaintext
plfdspec -c -a -200 results
```

The content of the file `results` is compiled in Tab. 2.1 (columns 3–6) and compared with experimental data (column 2).

To obtain all eigenvalues in this energy range, one has to run the propagation with different initial states, either sequentially or (more efficient) in parallel by performing a multi-packet propagation. Also, increasing the accuracy (more SPFs, e.g. 16/16/14) and increasing the propagation time (e.g. 250 fs) will help to detect states of low intensity. See Ref. [15] for details.
Table 2.1: Vibrational energies \((J = 0)\) of CO\(_2\). The MCTDH/FD energies, \(E_{FD}\), are compared with experimental ones, \(E_{exp}\). \(\Delta E\) and \(\Delta I\) denote internal error estimates of the eigen-energies and intensities, respectively. Missing entries specify states that have not been detected. In this case the intensity is taken from a larger calculation and is shown in brackets. The missed states are all of very low intensity, except for state 26. Here the computed state represents the two neighbouring states, 25 and 26. A calculation with a longer propagation time or with several wavepackets will detect more states. (See Ref. [15]). All energies are given in cm\(^{-1}\) with respect to the ground state energy.

<table>
<thead>
<tr>
<th>No.</th>
<th>(E_{exp})</th>
<th>(E_{FD})</th>
<th>(\Delta E)</th>
<th>Intensity</th>
<th>(\Delta I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>-0.002</td>
<td>0.054</td>
<td>4.49d-2</td>
<td>3.32d-5</td>
</tr>
<tr>
<td>1</td>
<td>1285.414</td>
<td>1285.393</td>
<td>0.096</td>
<td>1.07E-1</td>
<td>3.05E-5</td>
</tr>
<tr>
<td>2</td>
<td>1388.188</td>
<td>1388.276</td>
<td>0.264</td>
<td>3.48E-2</td>
<td>7.77E-5</td>
</tr>
<tr>
<td>3</td>
<td>2349.148</td>
<td>2349.090</td>
<td>0.086</td>
<td>3.17E-2</td>
<td>2.26E-4</td>
</tr>
<tr>
<td>4</td>
<td>2548.374</td>
<td>2548.349</td>
<td>0.017</td>
<td>1.06E-1</td>
<td>3.65E-5</td>
</tr>
<tr>
<td>5</td>
<td>2671.113</td>
<td>2671.152</td>
<td>0.008</td>
<td>9.08E-2</td>
<td>3.00E-5</td>
</tr>
<tr>
<td>6</td>
<td>2797.154</td>
<td>2796.339</td>
<td>0.752</td>
<td>4.23E-3</td>
<td>2.10E-4</td>
</tr>
<tr>
<td>7</td>
<td>3612.845</td>
<td>3612.860</td>
<td>0.013</td>
<td>5.61E-2</td>
<td>7.65E-5</td>
</tr>
<tr>
<td>8</td>
<td>3714.789</td>
<td>3715.040</td>
<td>0.108</td>
<td>1.57E-2</td>
<td>9.32E-5</td>
</tr>
<tr>
<td>9</td>
<td>3792.679</td>
<td>3792.531</td>
<td>0.075</td>
<td>6.78E-2</td>
<td>2.16E-4</td>
</tr>
<tr>
<td>10</td>
<td>3942.480</td>
<td>3942.562</td>
<td>0.002</td>
<td>8.62E-2</td>
<td>3.35E-4</td>
</tr>
<tr>
<td>11</td>
<td>4064.101</td>
<td>4064.190</td>
<td>0.040</td>
<td>1.34E-2</td>
<td>2.31E-4</td>
</tr>
<tr>
<td>12</td>
<td>4225.043</td>
<td></td>
<td></td>
<td>1.05E-4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4673.332</td>
<td></td>
<td></td>
<td>6.17E-5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4853.622</td>
<td>4853.747</td>
<td>0.046</td>
<td>4.27E-2</td>
<td>1.62E-4</td>
</tr>
<tr>
<td>15</td>
<td>4977.828</td>
<td>4997.548</td>
<td>0.572</td>
<td>2.73E-2</td>
<td>3.41E-4</td>
</tr>
<tr>
<td>16</td>
<td>5022.273</td>
<td>5022.408</td>
<td>0.492</td>
<td>3.41E-2</td>
<td>1.61E-4</td>
</tr>
<tr>
<td>17</td>
<td>5099.668</td>
<td></td>
<td></td>
<td>9.38E-4</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>5197.251</td>
<td>5197.442</td>
<td>0.055</td>
<td>4.47E-2</td>
<td>8.24E-6</td>
</tr>
<tr>
<td>19</td>
<td>5329.746</td>
<td>5329.986</td>
<td>0.029</td>
<td>1.31E-2</td>
<td>6.07E-5</td>
</tr>
<tr>
<td>20</td>
<td>5475.283</td>
<td>5480.947</td>
<td>0.839</td>
<td>2.19E-4</td>
<td>1.84E-5</td>
</tr>
<tr>
<td>21</td>
<td>5667.488</td>
<td></td>
<td></td>
<td>1.80E-7</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>5915.216</td>
<td></td>
<td></td>
<td>1.41E-4</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>6016.687</td>
<td></td>
<td></td>
<td>4.45E-4</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>6075.984</td>
<td>6076.471</td>
<td>0.554</td>
<td>2.12E-2</td>
<td>1.50E-3</td>
</tr>
<tr>
<td>25</td>
<td>6227.915</td>
<td>6233.344</td>
<td>0.074</td>
<td>3.08E-2</td>
<td>8.89E-5</td>
</tr>
<tr>
<td>26</td>
<td>6239.852</td>
<td></td>
<td></td>
<td>1.31E-2</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>6347.956</td>
<td>6333.856</td>
<td>2.964</td>
<td>2.15E-3</td>
<td>3.33E-4</td>
</tr>
<tr>
<td>28</td>
<td>6435.398</td>
<td>6434.692</td>
<td>0.261</td>
<td>1.59E-2</td>
<td>6.00E-4</td>
</tr>
<tr>
<td>29</td>
<td>6503.081</td>
<td></td>
<td></td>
<td>3.09E-6</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>6588.730</td>
<td>6588.345</td>
<td>0.745</td>
<td>5.48E-3</td>
<td>2.46E-5</td>
</tr>
</tbody>
</table>

2.6 Determining eigenstates by improved relaxation

Improved relaxation is a MCSCF variant where the SPFs are optimised by relaxation (propagation in negative imaginary time), but the A-vector is determined by diagonalisation of the Hamiltonian matrix evaluated in the set of the present SPFs. In contrast to filter diagonalisation, improved relaxation yields not only the eigenenergies but also the eigenstates. Improved relaxation is more accurate than filter diagonalisation, but also more elaborate, because one has to perform a separate calculation for each state.

1. Copy the file \$MCTDH\_DIR/inputs/co2\_gs.inp to your tutorial directory and similarly the files co2\_sym.inp, co2\_asym.inp, and co2\_excite.inp.
2.7 Determining eigenstates by block improved relaxation

To perform the relaxation, execute the command:

```
mctdh85 -mnd co2_gs
```

and, after the job has finished, run the inputs co2_sym, co2_asym, and co2_excite. Move to the directory co2_gs and type `rdrlx -e` to read the rlx_info file. This produces the following output.

<table>
<thead>
<tr>
<th>time</th>
<th>order</th>
<th>q</th>
<th>beta*1000 Energy[cm⁻¹]</th>
<th>ovl*1000</th>
<th>Delta-E</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.0</td>
<td>0</td>
<td>0</td>
<td>0.000E+00</td>
<td>2922.190</td>
<td>202 862</td>
</tr>
<tr>
<td>0.000</td>
<td>15*</td>
<td>0</td>
<td>804.71296</td>
<td>2534.937</td>
<td>314 025</td>
</tr>
<tr>
<td>0.500</td>
<td>11</td>
<td>0</td>
<td>999.98713</td>
<td>2534.663</td>
<td>002 346</td>
</tr>
<tr>
<td>1.250</td>
<td>10</td>
<td>0</td>
<td>999.99715</td>
<td>2534.558</td>
<td>864 426</td>
</tr>
<tr>
<td>2.000</td>
<td>9</td>
<td>0</td>
<td>999.99971</td>
<td>2534.535</td>
<td>729 282</td>
</tr>
<tr>
<td>4.000</td>
<td>8</td>
<td>0</td>
<td>999.99990</td>
<td>2534.528</td>
<td>474 396</td>
</tr>
<tr>
<td>6.000</td>
<td>6</td>
<td>0</td>
<td>0.293E-08</td>
<td>2534.528</td>
<td>204 472</td>
</tr>
<tr>
<td>8.000</td>
<td>4</td>
<td>0</td>
<td>0.874E-10</td>
<td>2534.528</td>
<td>194 788</td>
</tr>
</tbody>
</table>

**WARNING:** Davidson did not converge for 1 diagonalisations.

The star * at 15 indicates that that this Davidson diagonalisation did not converge. 16 Davidson iterations are needed for convergence, but in the input file the maximum number of iterations was limited to 15. The non-convergence is of no relevance here, because all later iterations converged. However, if non-convergence of the Davidson happens frequently, one cannot trust the results, but has to repeat the calculation with a different integrator setting.

The first data line with the negative time gives the energy expectation value of the initial wavefunction. The second line, \( t = 0 \), gives the energy obtained by diagonalising the Hamiltonian represented in the orbitals of the initial wavefunction. Then the orbitals (SPFs) are relaxed and the Hamiltonian matrix, built from the new orbitals, is diagonalised again. This procedure is repeated till convergence is reached.

For the present example the convergence of the improved relaxation scheme is fast. \( \beta \) denotes the squared overlap of the current A-vector with the previous one. If \( \beta \) is very close to one, the difference from 1 is printed. \( \text{ovl} \) denotes the squared overlap of the current wavefunction with the initial wavefunction. This data is evaluated only for relaxation=lock runs. More information on the performance of the improved relaxation run is obtained when dropping the option -e from `rdrlx`. (Try `rdrlx -h`). A graphical visualisation of the convergence is provided by `plrlx`. Try `plrlx`, `plrlx -a 3`, and `plrlx -E -l`.

Inspect the outputs of the other relaxation runs in a similar way. Note that considerably more Davidson iterations are needed for converging higher excited states. Note also that the energy scale is shifted via the keyword `rlxunit=cm⁻¹,2534.528194` to display directly the excitation energies. Inspect the input files and try to understand every line.

The tiny 3D problem CO\(_2\) is, of course, too simple to show the strengths of improved relaxation. If you wish to solve some 6D problems, run the input files `hono.dav.inp` and `H2CS.*.inp`
packages are formally put on different (single-set) electronic states. True electronic states, either in multi-set or single-set formalism, can be added as well. Because the packets are treated in single set, the SPFs are propagated (or relaxed) on a state-averaged mean field. As there is only one set of SPFs, the SPFs cannot be optimal for one eigenstate, they are optimized for the full block of eigenstates to be computed. Hence the block form will in general require more SPFs to achieve the same accuracy as a (single) improved relaxation. But because the block form generates several eigenstates at once, it is often more convenient and sometimes even numerically more efficient. However, a block-relaxation requires considerably more memory than a single relaxation.

The following example takes more computation time than the previous ones. It may be skipped if one is not particularly interested in block improved relaxation.

1. Copy the file $MCTDH_DIR/inputs/blkHONO.inp to your tutorial directory.

2. To perform the block-relaxation, execute the command:
   
   mctdh85 -mnd blkHONO &

   Edit the input file such that the numbers and keywords, which appear after “##” or “###”, become valid. Then run the input again. The convergence of the eigenenergies is most conveniently visited by running the script rdlrx. The convergence can be inspected graphically by running plbrlx, which is very similar to plrlx, but requires as argument the number of the state to be plotted. The converged eigenenergies (in cm$^{-1}$) obtained from these runs read as follows:

<table>
<thead>
<tr>
<th>blk.1</th>
<th>sing.1</th>
<th>blk.2</th>
<th>blk.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.006</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>93.992</td>
<td>93.973</td>
<td>93.974</td>
</tr>
<tr>
<td>2</td>
<td>600.920</td>
<td>600.872</td>
<td>600.873</td>
</tr>
<tr>
<td>3</td>
<td>710.781</td>
<td>710.623</td>
<td>710.625</td>
</tr>
<tr>
<td>4</td>
<td>796.056</td>
<td>795.999</td>
<td>796.000</td>
</tr>
<tr>
<td>5</td>
<td>944.422</td>
<td>944.111</td>
<td>944.116</td>
</tr>
<tr>
<td>6</td>
<td>1055.391</td>
<td>1055.384</td>
<td>1055.385</td>
</tr>
<tr>
<td>7</td>
<td>1188.605</td>
<td>1188.073</td>
<td>1188.079</td>
</tr>
<tr>
<td>8</td>
<td>1267.671</td>
<td>1267.600</td>
<td>1267.609</td>
</tr>
<tr>
<td>9</td>
<td>1306.671</td>
<td>1306.601</td>
<td>1306.604</td>
</tr>
<tr>
<td>10</td>
<td>1313.852</td>
<td>1312.748</td>
<td>1312.761</td>
</tr>
<tr>
<td>11</td>
<td>1386.094</td>
<td>1385.262</td>
<td>1385.263</td>
</tr>
<tr>
<td>12</td>
<td>1405.637</td>
<td>1405.519</td>
<td>1405.545</td>
</tr>
<tr>
<td>13</td>
<td>1549.283</td>
<td>1547.471</td>
<td>1547.458</td>
</tr>
<tr>
<td>14</td>
<td>1575.849</td>
<td>1574.831</td>
<td>1574.851</td>
</tr>
<tr>
<td>15</td>
<td>1640.938</td>
<td>1640.887</td>
<td>1640.887</td>
</tr>
<tr>
<td>16</td>
<td>1690.148</td>
<td>1690.009</td>
<td>1690.034</td>
</tr>
<tr>
<td>17</td>
<td>1726.574</td>
<td>1726.015</td>
<td>1726.050</td>
</tr>
<tr>
<td>18</td>
<td>1770.187</td>
<td>1761.572</td>
<td>1761.638</td>
</tr>
<tr>
<td>19</td>
<td>1783.998</td>
<td>1779.401</td>
<td>1779.466</td>
</tr>
<tr>
<td>20</td>
<td>1829.101</td>
<td>1829.017</td>
<td>1829.017</td>
</tr>
<tr>
<td>21</td>
<td>1858.538</td>
<td>1858.223</td>
<td>1858.242</td>
</tr>
<tr>
<td>22</td>
<td>1901.955</td>
<td>1902.884</td>
<td>1897.807</td>
</tr>
<tr>
<td>23</td>
<td>1909.695</td>
<td>1897.012</td>
<td>1902.886</td>
</tr>
</tbody>
</table>
The first run, blk.1, used the SPF set 9/4/16/18 and took 46 min CPU time on a 3.2 GHz Pentium 4, and 165 MB RAM, the second run with the SPF set 10/5/30/20 took 3 h CPU time and 560 MB RAM, the third run with 12/5/42/28 SPFs took 11 h CPU time and 1340 MB RAM. The third calculation, blk.3, is fully converged and serves as reference. Deviations from these results are indicated by . > 0.2, : > 0.5, + > 1.0, * > 2.0, # > 5.0, $ > 10., u unconverged.

As one notices, the eigenenergies obtained with the first run are not too accurate. In particular the last four states are quite bad, showing deviations of more than 10 cm$^{-1}$. The second calculation, blk.2, however, shows rather good results. All deviations are below 1 cm$^{-1}$, and in most cases the deviations are even below 0.1 cm$^{-1}$.

To compare block- with single-relaxation, separate single-relaxations were performed using the eigenstates of the block-relaxation as start vectors. The small set, 9/4/16/18, of SPFs was used and the results are shown in the column sing.1. The results of the single relaxations are much improved as compared to the corresponding block relaxation, except for the last four states and for the excited states no 18 and 23. These states (as well as state 36) did not converge, i.e. the energy kept on oscillating. The energies displayed in this case are some mean value. In the block-relaxation the SPFs are optimized to represent all 40 states under consideration, whereas in the single-relaxation they are optimized for a particular eigenstate. Obviously more SPFs are needed in a block-relaxation to obtain results of similar quality. However, the single relaxations took between 30 s and 3 min each, depending on the state to be relaxed. In total they took 1 h CPU time, which is similar to the 3 h used by the second block-relaxation. Remembering that for the single-relaxations we took excellent starting vectors, namely the eigenstates obtained by the first block-relaxation, and that the second block-relaxation yield eigenenergies of better accuracy than the sing.1 calculations, one may conclude that single- and block-relaxation take similar amounts of CPU time for obtaining similar accuracy. But the memory consumption of the block-relaxation is considerably larger (20 MB compared to 560 MB). However, it requires much less human effort to run a block-relaxation as compared to run 40 single relaxations.
2.8 Using potfit and chnpot to fit a surface to ab initio data points

In this example it is shown how to use ab initio data points to generate a natural potential and how this natural potential can then be interpolated into a more suitable grid for a MCTDH simulation. To perform such tasks the programs potfit85 and chnpot85 will be used, respectively. A detailed description of how such tasks are accomplished is found in chapter 12.2 of this guide.

The data used in this example – taken from Ref. [16] – is a 2D cut corresponding to the PES of the CO$_2^-$ anion in C$_2v$ symmetry. The two coordinates are the length of the two CO bonds and the angle between them, i.e. symmetric stretch and bending.

2.8.1 Transforming the ab initio data to product form

1. Create a new directory, for example co2fit
2. From the directory $MCTDH$DIR/pinputs/ copy to the directory co2fit the files:
   - co2_potfit.inp
   - co2_pes
   - co2_r.grid
   - co2_theta_grid

   The file co2_pes contains the ab initio energies and the two following files the corresponding grid points.
3. Move to the co2fit directory and execute the command:

   potfit85 -mnd co2_potfit

   A new subdirectory co2_potfit has been created, change to it and execute

   showpot85 -vfit

   In the interactive menu type 3 times 1, return to see the plot of the natural potential fit. By definition it is identical to the original data on the grid points since the same number of natural potentials as grid points has been used (Inspect the co2_potfit.inp file).

2.8.2 Interpolating the natural potential to a new primitive grid

1. Copy the file $MCTDH$DIR/pinputs/co2_chnpot.inp to the co2fit directory.
2. Execute the command:

   chnpot85 -mnd co2_chnpot

   A new subdirectory co2_chnpot has been created, which contains new dvr and natpot files. As before, the newly interpolated potential can be inspected using the utility showpot85 and following the interactive menu.

   The process outlined in the present and previous subsections can be repeated using the alternative set of files co2_r_dense_grid, co2_dense_pes, co2_theta_dense_grid,
co2_chnpot_dense.inp, co2_potfit_dense.inp. The initial grid is double as dense as the original one. After the potfit stage the data points are interpolated to the same grid as in the previous case. The rms of the difference of the two interpolations to the same grid is $9.55 \times 10^{-6}$ au, i.e. 0.26 meV. This is a very good value considering that the potential spans an energy interval of 8 eV. Indeed, using showpot85 to compare the outcome of both chnpot85 runs shows that the obtained potential energy surfaces are virtually equal, which constitutes a nice example of the usefulness of the chnpot85 utility when preparing an MCTDH calculation from ab initio data points.

We have used this method quite successfully to create multi-dimensional 3D-fits to ab initio data. The only restriction is that the data is to be supplied on a product grid. Equidistance of the grid points, however, is not required.

### 2.9 Optimizing an external field with Optimal Control Theory (OCT)

MCTDH can be used to perform coherent control calculations within the OCT scheme. The OCT algorithm was developed by Tannor and coworkers [17] and by Rabitz and coworkers [18]. For this purpose mctdh and the routine efield are called from the script optcntrl. OCT maximizes the expectation value $\langle \Psi(T) | O | \Psi(T) \rangle$ at the final time $T$, where $O$ denotes some positive semidefinite hermitian operator.

The control target $O$ can be defined in two different ways. If $O$ is the projector onto a target quantum state, i.e., $O = |\Psi_{tar}\rangle \langle \Psi_{tar}|$, then it is sufficient to specify the target state $|\Psi_{tar}\rangle$. If $O$ is a general operator, e.g., a projector onto an electronic state $O = |S\rangle \langle S|$, it has to be specified in the operator file. At present only one target operator can be specified in the operator file, it is, however, possible to use it with multiple initial states.

Multi-target optimizations are possible by using the multi-packet algorithm for target states. Multi-packet wave functions are treated within a multi-target optimization procedure. For target states the control functional $J$ can be chosen either as

$$J_1(E) = N_{tar}^{-1} \sum_{i} \left| \langle \Psi_{i}(T) | \Psi_{(tar,i)} \rangle \right|^2 - \alpha_0 \int_{0}^{T} dt \frac{E^2(t)}{S(t)},$$

or as

$$J_2(E) = N_{tar}^{-2} \sum_{i} \left| \langle \Psi_{i}(T) | \Psi_{(tar,i)} \rangle \right|^2 - \alpha_0 \int_{0}^{T} dt \frac{E^2(t)}{S(t)}.$$  

(2.1)

Here $\alpha_0$ is the so-called penalty factor that penalizes for strong fields and $S(t)$ serves as a pulse envelope that can be defined in the operator file. The functional (2.1) leads to optimizations of the target state populations only while within (2.2) the phases are also aligned. [19]

If $N_{tar} = 1$, both functionals are identical.

Example inputs are provided under $\$MCTDH\_DIR/inputs/optcntrl$. The Python script optcntrl parses the input file and invokes OCT related programs from the MCTDH package. Note: the script requires Python 2.4 or 2.5 and relies on the Python executable being found in /usr/bin/env python. This path can be changed in the first line of the script $\$MCTDH\_DIR/bin/python/optcntrl.py.

From the input file a number of temporary input files for the actual MCTDH calculations are created. The same applies for the operator file. The operator file must contain at least two
operators, the system Hamiltonian including the dipole operator multiplied with the electric field and a second operator containing the dipole operator alone. The system Hamiltonian is used to perform the propagations while the dipole operator is used to evaluate the electric field.

To execute an example, create a new empty directory, change to it, and copy the example input files, e.g. “pyrazine.inp” and “pyrazine.op” (see Ref. [20]). Inspect the input file and run the command:

```
optcntrl -mnd pyrazine.inp
```

The command

```
optcntrl -h
```

will provide help about options.

Recently, a number of new features have been implemented into OCT-MCTDH, such as optional filtering of the field and the use of different optimization schemes. [21–23] Please refer to the HTML documentation for details.

### 2.10 Concluding Remarks

This tutorial has shown you some typical applications of MCTDH. In order to ensure that within this tutorial all calculations can be done quickly – only the optimal control example takes somewhat longer, 60 minutes (on a 3 GHz PC) – we have chosen rather small example systems. This, however, should not mislead you, MCTDH is for treating large systems! The full power of MCTDH is uncovered when turning to problems which require such a large primitive product grid, that a standard (numerical exact) wavepacket propagation becomes impossible on a workstation. A good example for such a problem is the calculation of the absorption spectrum of pyrazine, as discussed in Ref. [9]. There the primitive product grid amounted to $6.6 \times 10^{20}$ points whereas the MCTDH calculation required only $3.76 \times 10^6$ configurations, 687 MByte RAM and 52 h CPU time. (This was done in 1998 on an IBM RS/6000 power2 workstation, which is much slower than any modern PC.)
Chapter 3

Defining the type of calculation to be made

In this chapter we present how to define and start the calculation to be made. Possible types are propagation, relaxation or diagonalisation. Propagation and relaxation calculations can be performed either using the MCTDH method, or numerically exactly, i.e. using the full primitive product grid. We also give a brief overview of the output to be produced.

3.1 Specifying the task for MCTDH

The MCTDH program package can perform different tasks specified in the RUN-SECTION. The following tasks are possible:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gendvr</td>
<td>1</td>
<td>A DVR file will be generated (see Sec. 4).</td>
</tr>
<tr>
<td>genoper</td>
<td>2</td>
<td>An operator file will be generated (see Sec. 6).</td>
</tr>
<tr>
<td>genpes</td>
<td>2</td>
<td>A special operator file, called pes, will be generated, that contains the potential energy surface (see below).</td>
</tr>
<tr>
<td>gengmat</td>
<td>2</td>
<td>A special operator file, called pes, will be generated, that contains an element of the G-matrix of the kinetic energy (see below).</td>
</tr>
<tr>
<td>geninwf</td>
<td>3</td>
<td>An initial wavefunction (restart file) will be generated (see Sec. 7).</td>
</tr>
<tr>
<td>test</td>
<td>4</td>
<td>All input files will be checked and all other files, necessary for a propagation, will be created, but no propagation step will be performed (see below).</td>
</tr>
<tr>
<td>propagation</td>
<td>4</td>
<td>Propagation in real time (see Sec. 3.3).</td>
</tr>
<tr>
<td>relaxation</td>
<td>4</td>
<td>A relaxation or an improved relaxation will be performed (see Sec. 3.4 or Sec. 3.5 respectively).</td>
</tr>
<tr>
<td>continuation</td>
<td>4</td>
<td>A continuation of the run will be performed (see Sec. 3.9).</td>
</tr>
<tr>
<td>diagonalisation</td>
<td>4</td>
<td>The Hamiltonian will be diagonalised (see Sec. 3.7).</td>
</tr>
</tbody>
</table>

When a high level task is to be performed, the necessary tasks of lower order are automatically included. E.g. by setting the keyword propagation one implicitly also sets gendvr, genoper and geninwf.
The pes-files, created with the keywords genpes or gengmat (or the options -pes or -gmat), are special operator files which include only potential like operators. Note that all non-diagonal (i.e. kinetic energy) terms are automatically removed from the Hamiltonian when setting the keyword genpes. The pes-file is usually used in the context of the analyse routines vminmax or showsys (see Sec. 11.10). The showsys program can plot 2D-cuts through potential energy surfaces (or other multidimensional functions) provided in the form of a pes-file. The pes-file, created with the genpes-keyword, contains the potential energy surface of the system. If the keyword gengmat = I1,I2 (or the option -gmat I1 I2) is used then the pes-file contains the (I1,I2) matrix element of the G-matrix of the kinetic energy operator. A kinetic energy operator can always be written in the form:

$$T = \frac{1}{2} \sum_{i,j} p_i^\dagger G_{ij} p_j + \sum_{i=1}^f F_i p_i + V_{\text{extra}} \quad (3.1)$$

where $G_{ij}$, $F_i$ and $V_{\text{extra}}$ are potential like terms. This equation defines the G-matrix. Note that the kinetic energy, defined in the Hamiltonian-Section, does not need to be of this particular form.

A further useful task is performed if the keyword test is used. In this case all input files are checked and all output files are created (even a psi file for t=0) like in a propagation run, but no propagation step is done. A test run may hence be used to convert a restart-file into a psi-file. It is possible to start a continuation run (see Sec. 3.9) after a test run.

### 3.2 Specifying the desired output

The output produced by the MCTDH program is sent to a directory called the name-directory. The (absolute or relative) path of this directory is specified by the name keyword in the RUN-SECTION of the input file. The name-directory should already exist when the MCTDH program is started. Otherwise one may use the option -mnd (make name directory) to create a new directory.

During run-time, a number of files are generated in the name-directory. Some of these are always created while others have to be selected by the user. The most interesting file of the former category is the log file, which records what happens at the various stages of a run.

Frequently used files of the second category are the output, auto, and psi files. The output file contains some basic physical quantities of the wavefunction, such as norm, energy, state populations, and the position and momentum expectation value of each coordinate. The auto file contains the auto-correlation function as a function of time. Both files are in ASCII format. In the psi file the wavefunction as a function of time is stored in binary format. The three files are selected by placing the keywords output, auto, or psi in the RUN-SECTION. Examples for the RUN-SECTION are given in the following sections. A complete list of the available files and options can be found in the HTML documentation.

The files gridpop, check, steps, update, timing, and speed only serve to check the efficiency or accuracy of an MCTDH calculation. They are very useful during the test phase of your calculations. Since some of them might become rather large, however, you may turn them off for your production calculations. NB the files output, timing, speed, and (for CMF runs) update are opened by default. To turn them off, use the keywords no-output (or screen), no-timing, etc.
3.3 Propagating a wavepacket

For performing a wavepacket propagation using the MCTDH method you first have to set up the Hamiltonian in an operator file (see Sec. 6). This operator file must then be specified in the OPERATOR-SECTION of the input file (see Sec. 6.1). In the input file the primitive and single-particle basis (Secs. 4 and 5), as well as the initial wavefunction (Sec. 7), must also be defined. Finally, you may select an integration scheme different from the default (Sec. 8).

A wavepacket propagation is then initiated by placing the keyword `propagation` in the RUN-SECTION. A typical example is

```
RUN-SECTION
propagation
  tfinal = 50.0   tout = 1.0
  name = results
  psi auto gridpop
end-run-section
```

The parameters `tfinal` and `tout` denote the time the propagation will run up to and the time interval after which the data is output (in femtoseconds). The name-directory is `results` in our example. The other parameters have been established in Sec. 3.2. A number of additional options may be selected in the RUN-SECTION. We refer the reader to the HTML documentation for details.

3.4 Relaxing a wavepacket to produce the lowest eigenstate

In a relaxation calculation a wavepacket is propagated in imaginary time to produce the lowest ro-vibrational eigenstate. Analogous to a (real-time) propagation, an operator and an input file are required to define the Hamiltonian, the primitive and single-particle basis, the initial wavefunction, and possibly the integration scheme.

A relaxation calculation is selected in the RUN-SECTION by the keyword `relaxation` instead of `propagation`. The RUN-SECTION may read

```
RUN-SECTION
relaxation
  tout = 10.0
  tfinal = 100.0
  name = results
end-run-section
```

The parameters have been introduced in Secs. 3.2 and 3.3.

After a successful run, the desired lowest eigenstate is stored in the restart file. This eigenstate can then be used as initial wavefunction in following calculations, by using the file keyword in the INIT_WF-SECTION.

To check the convergence of a relaxation calculation with respect to the propagation time `tfinal`, you may look at the total energy being displayed in the output file. If this has not changed significantly during the last outputs, the eigenstate is converged. Otherwise, the calculation should be continued to longer times. See Sec. 3.9 for continuing calculations.
3.5 Advanced topic: Improved relaxation. Generation of excited eigenstates

This section may be difficult to understand, if one is not familiar with the constant mean field (CMF) integration scheme. For a brief discussion on CMF see Sec. 8.2 and 8.4.2. A more comprehensive discussion on CMF can be found in the MCTDH review. The Improved Relaxation algorithm is discussed in the MCTDH feature article [5] and more recently (and more comprehensively) in refs. [7, 24, 25].

Consider a relaxation run where the CMF integration scheme is adopted and the SIL integrator is used to propagate the A-vector. In this case it seems to be somewhat cumbersome to generate a relaxed (i.e. propagated in imaginary time) A-vector by taking a linear combination of the eigenvectors of the Lanczos matrix. It is more meaningful to replace the relaxed A-vector by the ground state of the Lanczos matrix, as one is interested in the ground-state but not in a proper propagation in imaginary time. This modification leads not only to a faster convergence but, more importantly, offers the possibility to converge to an excited state. One simply takes the n-th state of the Lanczos matrix as "relaxed" A-vector. However, the dimension of the Lanczos matrix is in general small compared to the length of the A-vector. The pseudo-spectrum of the Lanczos matrix is thus a poor simulation of the spectrum of the matrix representation (in the set of the single particle functions) of the Hamiltonian \( H \). Although the n-th eigenstate of the Lanczos matrix may be a good approximation to an eigenstate of the \( H \)-matrix, it may not approximate the n-th state of \( H \), but a higher one. Thus the algorithm of improved relaxation will converge to some eigenstate, but one is not sure, which one it is (except when the ground-state is sought).

To deal with this situation, the algorithm is set up such, that that eigenvector of the Lanczos matrix is taken, which has the largest overlap with the one of the previous CMF step. (A warning is written to the log file, if this overlap-squared is smaller than 0.66). There are two ways to select the initial Lanczos vector. The first way is to write relaxation = \( n \) to the RUN-SECTION. The n-th eigenvector of the Lanczos matrix is then taken as starting point of the relaxation. The counting starts from zero, i.e. \( n = 0, 1, 2, \ldots, 98 \). Note that the Lanczos matrix depends on the starting vector, which in this case is the initial wavefunction defined in the INIT_WF-SECTION.

The second way is to specify relaxation = follow. In this case the starting vector of the improved relaxation is that eigenvector of the Lanczos matrix, that has the largest overlap with the initial wavefunction defined in the INIT_WF-SECTION.

The Krylov space and thus the dimension of the Lanczos matrix grows with each step of the Lanczos iteration. This process is stopped when the accuracy criterion is satisfied (the SIL accuracy parameter is now interpreted as the tolerated error in milli Hartree of the energy of the desired Lanczos eigenvalue ) or when the specified maximal dimension of the Lanczos matrix is reached. If the keyword full is given as a second argument of the relaxation keyword, then during the very first build up of the Krylov space the iteration will be continued till the maximal dimension is reached. This feature is useful to ensure that the improved relaxation starts from the correct Lanczos eigenvector. There is also the keyword ortho, which may appear as an argument to the relaxation keyword. ortho forces the SIL integrator to perform a full re-orthogonalisation of the Krylov space. This often significantly improves the convergence, but for long A-vectors it may take some CPU–time. In short, the improved relaxation command may read relaxation=follow,full,ortho.
As the A-vector now changes discontinuously, the standard variant of the CMF step size control, CMF/var, does not work. One must either work with fixed CMF steps, CMF/fix, or let the step size control depend on the single particle functions only, CMF/varphi. The latter choice is usually to be preferred. For convenience CMF is set equivalent to CMF/varphi when the run-type is improved relaxation. In all other cases CMF is interpreted as CMF/var. Note that the integrator parameters, initial CMF step size, CMF accuracy (only for varphi), maximal Lanczos space and SIL accuracy, may have decisive effects on the convergence. The SIL accuracy and space should be chosen higher than for propagation. Use an SIL accuracy between $10^{-8}$ and $10^{-11}$ and a Lanczos space between 20 and 200 (the maximal size is 500). The higher the sought eigenstate lies, the larger must be the Lanczos (or Krylov) space. The convergence to higher lying states also requires more SPFs than needed for propagation (and as indicated by the natural weights).

The single-particle functions are propagated in imaginary time, i.e. they converge toward the lowest states of the mean-field operator, irrespectively whether they are needed for representing the wavefunction or not. If, for example, there is no node-less single-particle function approximating the ground-state of the mean-field operator, then the propagation in imaginary time will rapidly change the single-particle functions in order to generate such a function. Due to this rapid change, the method then fails to converge. One thus may have to select the initial single-particle functions somewhat more carefully. However, there is no implemented algorithm to do so and one may therefore be forced to use more single-particle functions for improved relaxation than finally needed to represent the converged wavefunction.

When the keyword orben is set in the RUN-SECTION, then the so called orbital energies are computed and output to the file orben. The orbital energies are obtained by diagonalising an appropriate mode-Hamiltonian in the set of the single-particle functions. The mode Hamiltonian is conveniently defined as the trace over the mean-fields of the particular mode under discussion. I.e.

$$H_{av} = \sum_{j=1}^{n_{av}} \langle H \rangle_{jj}^{(e)}.$$  

See the MCTDH-review for a definition of the mean-field operators $\langle H \rangle^{(e)}$.

The eigen-functions of $H_{av}$, called energy orbitals, allow to define energy weights as the diagonal values of the density matrix expressed in the basis of the energy orbitals. The energy weights are very useful for assigning quantum numbers to a relaxed wavefunction. If for each mode there is one weight that is close to one (larger than 2/3, say), then the full wavefunction is characterised by the quantum numbers of the dominant energy orbitals. For further information see the HTML documentation under Input-Documentation/Run-Section and Output-Documentation/Data-files.

More recently (Aug 2003) a Davidson diagonaliser has been implemented to replace the SIL for improved relaxation. The Davidson algorithm is much superior to Lanczos when the generation of a single excited eigenstate is asked for. For the Davidson, there is a new keyword relaxation=lock. This is similar to relaxation=follow but more stable, because lock searches for the eigenvector with the largest overlap with the initial wavefunction, whereas follow takes the eigenvector with the largest overlap with previous vector. The keyword ortho is not allowed when using the Davidson, and full is useful in special cases only. See the HTML documentation for more details. A typical input for improved relaxation with Davidson is provided by inputs/hono.dav.inp. Note that in this example one is computing the about 120th eigenstate in A' symmetry.

There are several versions of the Davidson diagonaliser implemented. With the keyword DAV one calls a routine which works for hermitian Hamiltonians. With the aid of the keyword
CDAV one may diagonalise non-hermitian (e.g. CAP augmented) Hamiltonians and compute complex resonance energies. However, in most applications the Hamiltonian will not only be hermitian, it will also be real. (A Hamiltonian is called real, if $H \Psi$ is real for every real $\Psi$). As MCTDH is written to propagate wave packets, almost all variables are declared as complex. Hence one is wasting memory and CPU time by not making use of the reality of the wave function. To reduce this waste, one may use the keywords rDAV or rrDAV. rDAV is similar to DAV, but it stores the Davidson vectors as real. This substantially reduces the memory demand. rrDAV performs in addition the matrix-vector operation $HA$ in real arithmetic, which speeds up this step. Note that rrDAV can be used only, if each single operator appearing in the Hamiltonian is real. This, in particular, excludes the use of the operator $p$. (One may replace it by the operator $dq$). Furthermore, rrDAV cannot be used, if there are so called muld-operators (i.e. multi-dimensional operators which act on more than one MCTDH particle simultaneously). The use of natural potentials, however, is allowed. Note that the relaxation of the SPFs is always done in complex arithmetic.

Turning more to the technical side we note that the integrator setting for improved relaxation is quite different from the one for propagation. For improved relaxation (with Davidson) the CMF accuracy is usually set to a rather low value, around $10^{-3}$, whereas high accuracies, $10^{-7} \ldots 10^{-8}$ and $10^{-8} \ldots 10^{-10}$, are to be used when propagating the SFPs and the A-vector, respectively. (See Sec. 8.4.2). For propagating the SFPs we recommend the RK8 integrator. In contrast to a propagation run one now prefers to have an output after each update step, i.e. after each diagonalisation. To achieve this, one sets $tout=all$. In this case one should also give $tpsi$, even if the $psi$ file is not written. The $tpsi$ time is taken as the upper limit for the update interval.

When computing the ground state, the Davidson diagonaliser is usually quite fast, i.e. requires only few iterations. Turning to excited states, however, the number of Davidson-iterations may become quite large. In order to speed up the calculation in this case, a better pre-conditioner was implemented. If one sets $precon=N$ then an $N \times N$ block of the Hamiltonian matrix is diagonalised and used to improve the pre-conditioner. One should be careful not to use a too large value for $N$, otherwise the build-up of the pre-conditioner takes more time as saved by performing a smaller number of Davidson iterations. It is usually not useful to use $precon$ when the ground state is computed. When computing higher excited states, on the other hand, the pre-conditioner can be very helpful.

When excited states are to be computed one usually uses $relaxation=lock$. This, however, requires that an initial state is provided, which has a decent overlap with the state to be computed. There are two convenient ways to generate such an initial state. The first one is to operate with some excitation operator on the ground state, or on some converged excited state. (Compare with the co2_*.inp input files on MCTDH_DIR/inputs). The other way is to diagonalise appropriate 1D-operators with eigenf or mode-operators with meigenf and build the initial state as Hartree product from those low-dimensional eigenstates. (Compare with the hono.dav.inp and H2CS.r1r2.inp input files on MCTDH_DIR/inputs). Furthermore, one may use the orthogonalise keyword to purify an initial state from contributions of already converged eigenstates. (See HTML documentation under INIT_WF-SECTION).

When the CI-space is too small (i.e. when the A-vector length is too short) the improved relaxation algorithm will not converge due to a “variational breakdown”. Not only the state requested but all states below this one must be representable by the SPF basis sets. For highly excited states this may require large CI-spaces. One must be careful when using mode combinations and avoid too strongly combined modes because over-combination will make the CI-space too small. Relaxation to the ground state is always unproblematic. If
a relaxation to an excited state does not converge, one has to use more SPFs, although the natural populations may already be very low. The natural population indicates how important an orbital is for representing the desired state. But again, here the orbitals have to represent in addition all states below the desired one.

Rather than (single) relaxation one may use block-relaxation. See Sec. 2.7 for an example. The keyword *split-rst* splits the restart file of the block-relaxation into individual restart files for each eigenstate. Block-relaxation does not allow the use of the keywords-arguments *follow* or *lock*, only *relaxation=0* is allowed. Hence one computes the $b$ lowest eigenstates, where $b$ denote the block size. However, one may set the keyword *rlxemin* (see the HTML docu for a full description). This will force the code to compute the $b$ lowest states above the argument of *rlxemin*.

An initial wavefunction for a block improved relaxation run can be read by using the keywords *block-spf*, *block-A*, or, if there is already a wavefunction in correct block form, simply by using the *file* keyword. The keyword *autoblock* is particularly useful. See the HTML docu for a comprehensive description of these keywords.

Finally we note that the *rlx_info* file is most conveniently read with the aid of the script *rdrlx*. Type *rdrlx -h* for obtaining more information. The script *plrlx* (or *plbrlx* for block-relaxation) plots the energy versus relaxation time.

### 3.6 Performing a numerically exact calculation

The MCTDH program — although originally developed for wavefunction dynamics within the MCTDH scheme — also allows one to perform numerically exact wavefunction propagations. Employing the MCTDH program for such calculations has the advantage that one can benefit from the easy way a Hamiltonian can be set up. A numerically exact calculation may also be useful for comparison with an MCTDH calculation. Of course, a numerically exact propagation is only feasible for rather small systems.

For a numerically exact wavepacket propagation an operator file and the same input sections as in an MCTDH calculation are required. A numerically exact calculation can be made by including the keyword *exact* in the RUN-SECTION in addition to the calculation type keyword (e.g. propagation or relaxation). Rather than using the low-dimensional MCTDH single-particle functions, this sets up a wavepacket (or for a non-adiabatic system one for each electronic state) on the full product primitive basis. The operator is also set up on this full grid.

To propagate this wavepacket any of the integrators listed in Tab. 8.1 can be used. Although the ABM integrator is the default, the best performance is typically obtained by the SIL integrator, because it exploits the fact that the equations of motion in a numerically exact calculation are linear. To select the SIL integrator, insert for instance

```
INTEGRATOR-SECTION
   SIL = 20, 1.0d-6
end-integrator-section
```

into your input file. The parameters are discussed in Sec. 8.3. The ABM, BS or RK5/8 integrator can be chosen as described in the examples in Sec. 8.
3.7 Diagonalising the Hamiltonian using the Lanczos algorithm

Although the MCTDH method is a time-dependent one, the MCTDH program is also capable of diagonalising a Hamiltonian using the Lanczos scheme. In such a diagonalisation run the wavefunction is automatically represented on a (primitive) product grid, i.e. in the same way as in a numerically exact calculation.

A diagonalisation run therefore requires the same input sections as a numerically exact calculation, and of course an operator file. A possible RUN-SECTION reads

```
RUN-SECTION
diagonalisation = 10000
name = results
end-run-section
```

The number associated with the `diagonalisation` keyword indicates the number of Lanczos iterations to be made. The `name` keyword has been explained in Sec. 3.2. Other keywords that may be specified in a diagonalisation run can be found in the HTML documentation.

The computed eigenenergies and intensities, together with an error estimate of the eigenenergies, are compiled in the `eigval` file. If the error of the desired eigenvalues is too large, the calculation can be continued to increase the number of iterations. See Sec. 3.9 for continuing calculations.

Note that one-dimensional Hamiltonians can be numerically exactly diagonalised using a DVR basis. See Sec. 7.6 for more details of this.

3.8 Starting a calculation

The MCTDH program is started by typing

```
mctdh85 myinput
```

on your console, where we assumed that your input file is named `myinput.inp`. If the input file is not stored in the current directory, add the correct path to the input file’s name. A variety of options may be used on starting the MCTDH program. Type

```
mctdh85 -h
```

to get an overview.

To find out during run-time how far a propagation is proceeded, you may look for instance at the last line of the `log` file, by typing

```
tail -1 results/log
```

where `results` has to be substituted by the name-directory.

Any error messages that might be raised during run-time are sent directly to the screen and additionally written to the `log` file. See there in case of problems. If the calculation was successful, the results can be analysed by the Analyse programs and scripts. This is detailed in Chap. 11.
3.9  **Advanced topic: Continuing or stopping a calculation**

A calculation that has been finished may be continued again, so propagating to longer times or performing a larger number of Lanczos iterations. This is done by adding the keyword `continuation` to the RUN-SECTION. (Alternatively, one may start a continuation run using the `-c` option on the command line, see the HTML documentation for details.) In a propagation run the final time must also be increased. For instance, if the previous calculation finished after 50 fs, then you may set $t_{\text{final}} = 75.0$ to propagate over the next 25 fs. Similarly, if 10,000 iterations were made in a diagonalisation run, set $\text{diagonalisation} = 15000$ for the next 5,000.

Again, it is often more convenient to use options. E.g. the command

```
mctdh85 -c -tfinal 75.0 <inputfile>
```

will start a continuation run where the final time is set to 75 fs. The use of other options like `-I`, `-tcpu` or `-tstop` may be useful. Type `mctdh85 -h` to see the list of options. Note that a continuation run reads only the RUN-SECTION and ignores all the other sections of the input file as well as the operator file. All the necessary information is read from the read-write files of the name directory.

Sometimes it is desirable that a calculation is continued with a different integrator setting. This can be accomplished by giving the keyword `continuation=integrator` or by the option `-ci`. In this case the INTEGRATOR-SECTION will be read.

Finally, the `continuation` keyword can be used in order to try to complete a crashed calculation. The program however does not check the output files for consistency. The continuation thus might fail if some relevant data was lost due to the crash.

Another option of the MCTDH program is to stop a calculation during run-time in a controlled manner, such that it can be resumed later. To this end include the stop file by adding the keyword `stop` to the RUN-SECTION. To halt a calculation after the next output, edit the stop file and write `stop` to its first line. Alternatively one may just remove the stop file. The stop is automatically deleted when the run finishes. It thus may serve as a lock-file. As long as the stop file exists, the run is not finished.

Rather then simply writing `stop` to the stop file one may supply more specific commands which will halt the program after a certain real-time or CPU-time has passed. This allows e.g. to outwit CPU-time limits. Rather than editing the stop file "by hand", one may let the MCTDH program do that. This can be accomplished by giving the keywords `tcpu` and/or `tstop` in the RUN-SECTION or by using the options `-tcpu` and/or `-tstop`. See the HTML documentation for details.

3.10  **Advanced topic: Using parallel shared memory hardware**

If shared memory hardware is available MCTDH can take advantage of it. The parallel features of MCTDH are used if the keyword `usepthreads = I` is set in the RUN-SECTION, where I stands for the number of processors that should be used. Further arguments can be added, which disable the parallelisation of the different MCTDH routines. The parallelisation of the following routines can be disabled: `phihiphi` (no-phihiphi), `calcha/funka2` (no-funka), `summf` (no-summf), `mfields` (no-mfields), `hlochphi` (no-hlochphi), `hlochphilm` (no-hlochphilm), `funkphi` (no-funkphi), `getdavmat` (no-getdavmat) and `dsyev` (no-dsyev). In the MCTDH code the funkphi routine calls several subroutines:
Defining the type of calculation to be made

hlochphi1m, mfsunphi1m, dichlphi1ms, hunphi1ms, addhunphi1ms and project1ms. (The parallelisation of the first routine is switched off by setting the no-hlochphi1m keyword. Setting the no-funkphi keyword switches off the parallelisation of the other routines all together.) The funka2 routine is only used in the case of a relaxation run with the rrDAV integrator.

There are more keywords. The mem-calcha and the mem-mfields keywords enable MCTDH to use more memory for a more efficient parallelisation of the calcha and mfields routines. In default mode the parallelisation is optimized for low memory requirements. Furthermore there is the summf2 keyword. If this keyword is set, a differently parallelised summf routine is used. This may increase the efficiency of the parallelisation if a very large combined mode is present, dominating the calculation of summf. A parallel version of the LAPACK dsysv-routine is used in MCTDH. The dsysv = I keyword can be used to set the minimum size of the matrix that is diagonalized in parallel.

The results of a parallel calculation may slightly differ from those of a non-parallel one due to numerical reasons. Furthermore a parallel calculation needs more memory. This is one of the reasons why the parallel use of the routines can be disabled. Depending on the type and the parameters of the calculation some routines may only marginally improve the performance of the parallelisation. They can be turned off to save memory. Moreover, depending on the calculations made, some routines can produce overhead which overcompensates the gain of their parallelisation. These routines should also be turned off.

Example:

```
RUN-SECTION
usepthreads = 4, no-funkphi
...
end-run-section
```

In this example a parallel calculation with 4 processors will be performed but the parallelisation of the funkphi routine is disabled. Sometimes disabling the parallelisation of routines even increases the computational speed, as it is the case for propagation of C$_2$H$_4$. If the calculation is made with all routines running in parallel mode the parallel part of the program, according to Amdahl’s law, is 61% whereas the parallel part is 75% if the parallelisation of the funkphi routine is disabled.

In general the parallelisation works better for larger systems, i.e. systems with many Hamiltonian terms and many single particle functions. This is due to the fact that either loops over the Hamiltonian terms or loops over the single particle functions are parallelised. We have observed that the performance of the parallelisation does not only depend on the system studied but also on the computer platform and compiler. The C$_2$H$_4$ system has been propagated on an Itanium cluster using the Intel compiler. This resulted in a speedup of 2.30 (4 processors) for which Amdahl’s law states a parallel parallel part of 75%. The same system, propagated on a quad-opteron, also with the Intel compiler, showed a speedup of 2.58 (4 processors) and hence a parallel part of 81.7%.

One of our better examples is H$_2$ + H$_2$ inelastic scattering. Here the Hamiltonian consists of many terms because there is a large potfit. A speedup of 6.19 is observed when running this system on 8 processors in parallel. This implies that 95.5% of the work is done in parallel. The memory used increased form 56 MB (one processor) to 60 MB (8 processors).

The performance of the parallelisation can be monitored using the keyword ptiming. If this keyword is set in the RUN-SECTION an additional timing file, called ptiming, is created containing information about the time spent in each thread and routine. This keyword also can be used if usepthreads is not set. In this case no ptiming file is created, but the timing
file contains information about the timing of the parallelised routines. This helps to decide what routine should be used in parallel mode if the parallelisation is turned on.

The ptiming file is structured in the following way (H$_2$O$_2$ propagation):

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Calls</th>
<th>cpu</th>
<th>sum</th>
<th>thread: 1</th>
<th>thread: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>phi phi</td>
<td>52</td>
<td>822.72</td>
<td>822.83</td>
<td>411.44</td>
<td>411.39</td>
</tr>
<tr>
<td>cal cha</td>
<td>168</td>
<td>11297.70</td>
<td>11363.55</td>
<td>5680.55</td>
<td>5683.00</td>
</tr>
<tr>
<td>m fields</td>
<td>51</td>
<td>8054.95</td>
<td>8055.37</td>
<td>4026.99</td>
<td>4028.38</td>
</tr>
<tr>
<td>sum mf</td>
<td>51</td>
<td>4991.97</td>
<td>9529.60</td>
<td>4764.80</td>
<td>4764.80</td>
</tr>
<tr>
<td>hloch phi lm</td>
<td>5617</td>
<td>11071.72</td>
<td>11204.26</td>
<td>5667.22</td>
<td>5537.04</td>
</tr>
<tr>
<td>funk phi</td>
<td>5237</td>
<td>716.71</td>
<td>726.63</td>
<td>329.93</td>
<td>396.70</td>
</tr>
</tbody>
</table>

The first column shows the name of the parallel subroutine, the next column gives the number of calls to this routine. The column “cpu” shows how much cpu time is spend for the computation. The columns “thread: p” give the real time spend in each thread (here 2) for the computations made, these values are summed up in column “sum”. In this example the parallelisation of the summf routine works badly. This can be seen, because the cpu time for the summf evaluation is much lower than the sum of the realtime spend for this task. Using the summf2 keyword this problem can be fixed. The corresponding line in a summf2 run indicates that the parallelisation now works better:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Calls</th>
<th>cpu</th>
<th>sum</th>
<th>thread: 1</th>
<th>thread: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>188394</td>
<td>3809.27</td>
<td>3868.90</td>
<td>1917.64</td>
<td>1951.25</td>
</tr>
</tbody>
</table>

Ignore the number of calls. It is increased because now not summf is timed but an internal routine called by summf. More things can be seen with help of the ptiming files. Sometimes the parallelisation creates cpu time overhead. This cannot be discovered by checking only one ptiming file. In the case of the C$_2$H$_4$ propagation the routines controlled with the no-funkphi keyword produce overhead. But the corresponding ptiming file is:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Calls</th>
<th>cpu</th>
<th>sum</th>
<th>thread: 1</th>
<th>thread: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>funk phi</td>
<td>121389</td>
<td>606.69</td>
<td>627.99</td>
<td>308.03</td>
<td>319.96</td>
</tr>
</tbody>
</table>

Here the columns “cpu” and “sum” compare very well. But the ptiming file for one thread, i.e. use pthreads=1, reads:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Calls</th>
<th>cpu</th>
<th>sum</th>
<th>thread: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>funk phi</td>
<td>120780</td>
<td>174.36</td>
<td>175.23</td>
<td>175.23</td>
</tr>
</tbody>
</table>

Comparing the “cpu” column of both files a strong increase of cpu time is discovered, 174.36s (1 proc) to 606.69s (2 proc). Hence the no-funkphi keyword should be used to avoid this overhead. Finally we give some overall timings:

<table>
<thead>
<tr>
<th></th>
<th>1 processor</th>
<th>4 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>H$_2$O$_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>9h 52m 20s</td>
<td>3h 50m 52s</td>
</tr>
<tr>
<td>no-summf</td>
<td>3h 52m 06s</td>
<td>3h 09m 49s</td>
</tr>
<tr>
<td>summf2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H$_2$O$_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>22m 21s</td>
<td>12m 20s</td>
</tr>
<tr>
<td>no-funkphi</td>
<td></td>
<td>8m 49s</td>
</tr>
</tbody>
</table>

If the ptiming keyword is used but the lrt library is not linked the following error message appears.
32 3 Defining the type of calculation to be made

###########################################
### --- no clock_gettime command ---
### lrt not linked!
### If lrt is available,
### modify the script compile.cnf
###########################################

If your compiler supports this library the script compile.cnf has to be modified.1 The option
-lrt must be added in the line MCTDH_ADD_LIBS in the section of the compiler that is
used. Then MCTDH must be compiled again.

A similar error message appears if the pthread library was not linked for compilation of
MCTDH.

###########################################
### --- no xxx_yyyy command ---
### pthread library not linked!
### If your compiler supports pthread,
### modify the script compile.cnf
###########################################

xxx_yyyy is replaced by the name of the routine the program tried to execute but could
not be found. If your compiler supports pthreads, the script compile.cnf has to be modified
(see above). The option -lpthread must be added in the line MCTDH_ADD_LIBS and the
option -pthread must be added in the line MCTDH_CFLAGS in the section of the compiler
that is used. Then MCTDH must be compiled again.

For the standard compilers, e.g. GNU or Intel, all the necessary extensions of the compile
scripts are already done.

To improve the efficiency of the shared memory parallel MCTDH on NUMA (non uni-
form memory access) machines the numa.h library can be linked 2. To do so the -u option
must be given for compilation:

compile -u mctdh

Doing so the POSIX-threads created during an MCTDH-run are distributed and bound
cyclicly to the available processors to prevent thread migration. Of course, this only works if
the numa library is available on your computer.

On NUMA machines (e.g. typical Opteron or Xeon clusters) we observed that execution
times of identical runs may vary by more than 20%. Using the numa.h library, however, these
surprising variations vanish and all runs take almost identical execution times. These times
agree with the shortest times observed without using numa.h.

3.11 Advanced topic: Using parallel distributed memory hardware

Beside parallel shared memory hardware, MCTDH can make use of parallel distributed
memory hardware. For the distributed memory parallelization of MCTDH the Message

1One should edit compile.cnf_be and compile.cnf_le as well because compile.cnf is overwritten by one of the
latter files when install_mctdh is executed.

2You may visit http://oss.sgi.com/projects/numa/ to find more information about NUMA API
3.11 Using parallel distributed memory hardware

Passing Interface (MPI) was used. To use the MPI-parallel MCTDH, the keyword usempi must be set in the RUN-SECTION and the MCTDH program must be started with the mpirun command. Further the MPI compilation of MCTDH has to be used (see below). To invoke an MPI-parallel run with 32 MPI-prozesses the command could look like this:

```
mpirun -np 32 mctdh85.mpi-g77 <inputfile>
```

Further arguments can be added to the usempi keyword, which disable the parallelisation of the MPI-parallel routines. These routines are: calcha (no-calcha), funka2 (no-funka2), mfields (no-mfields), getdavmat (no-getdavmat), dsyev (no-dsyev), phiophi (no-phiophi), hlochphi (no-hlochphi) and mpirdavstep/mpibdavstep (no-dav). These keywords are similar to those used in shared memory parallelisation with the usepthreads keyword but the last one, no-dav, is different. This keyword disables the usage of the MPI-parallel routines mpirdavstep/mpibdavstep. This means that the davidson vectors that are built during an improved relaxation step are stored on one node and are not distributed. Additionally there is the keyword dav=I, where I stands for the maximum number of davidson vectors that are stored on one node. If this keyword is not set the maximum allowed number of vectors (integration order) is equally distributed over the available nodes (intorder/# of MPI-prozesses). If I is set to a higher value this can lead to smaller communication costs because the vectors are held on a smaller number of prozessors.

As in the case of the shared memory parallelisation the results of a parallel calculation may slightly differ from those of a non-parallel one due to numerical reasons. An example for a RUN-SECTION in an MPI-parallel run is shown in the following Example:

```
RUN-SECTION
usempi, no-dsyev
...
end-run-section
```

Here the parallelisation of the diagonalisation routine DSYEV is turned off. The number of prozesses is not specified in the RUN-section as is must be done in the shared memory parallel case. This is done via the mpirun-command (see above).

In the case of the distributed memory parallelization not all parts of MCTDH are parallelized, only the mean-field computation and the A-vector propagation are parallel but not the SPF-propagation. This is due to the fact that large calculations are A-vector dominated ones in general. Hence calculations with a considerable contribution from the SPF propagation to the cpu time are not suited for the MPI-parallel MCTDH. The best tested case was the $\text{H}_2\text{O}_2^+$-propagation where up to 1024 prozessors were used. This calculation showed a maximum speedup of 118, this corresponds to a parallel part of over 99%.

If the keyword ptiming is set for an MPI-parallel run the file mpitiming is created. This file contains timing information for each MPI prozess. An mpitiming-file for a run with 8 prozesses is shown here:

<table>
<thead>
<tr>
<th>Process</th>
<th>cpu</th>
<th>%cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14501.84</td>
<td>12.40</td>
</tr>
<tr>
<td>1</td>
<td>15288.35</td>
<td>13.08</td>
</tr>
<tr>
<td>2</td>
<td>14399.39</td>
<td>12.32</td>
</tr>
</tbody>
</table>
Defining the type of calculation to be made

<table>
<thead>
<tr>
<th>Process</th>
<th>cpu</th>
<th>%cpu</th>
<th>sum real time ( 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>51811.34</td>
<td>50.31</td>
<td>51574.81</td>
</tr>
<tr>
<td>1</td>
<td>51168.49</td>
<td>49.69</td>
<td>51095.79</td>
</tr>
</tbody>
</table>

The first column denotes the MPI process, 0 being the master process, the second column shows the cpu time spent in each process and the third column gives the percentage of the cpu time. The Example shows a well parallellized case where the cpu time is equally distributed over the processes.

If the MPI parallelization is combined with the usage of POSIX threads the fourth column appears that shows the sum of real time spent in the threads of each process. This is similar to the summation done for the ptiming-files but here the summation was additionally done over the different routines. The following example is an MCTDH-run with 2 MPI-processes with 4 POSIX-threads each, denoted by (4):

If the sum of real time is comparable to the cpu time the shared memory parallelization works well for each prozess (see the comments to ptiming-file in sec. 3.10).

For combined calculations, distributed and shared memory parallelization, a further option for the ptiming-keyword can be set: ptiming=all. If this is done a ptiming-file is created for each MPI-prozess, containing the timing information for the parallel routines. Thereby the quality of the shared memory parallelization within the different MPI-prozesses can be checked in more detail as by the mpitiming-file. These files are denoted by ptiming0, ptiming1 and so on. This is mainly for testing.

To combine distributed and shared memory parallelization both, the usepthreads and the usempi keywords, must be set in the RUN-section and MCTDH must be started with the mpirun command. For a parallel calculation where each MPI-prozess uses 4 POSIX-threads the RUN-section may look like:

```
RUN-SECTION
usepthreads=4
usempi
...
end-run-section
```

Here the parallelization for all routines is working. The MPI-parallel ones are called in each MPI-prozess where they run in shared memory parallel mode with 4 POSIX-threads. Additionally in the master-process the other shared memory parallel routines for the SPF-propagation are used.

Example:

```
mpirun -np 5 mctdh85.mpi-g77 <inputfile>
```

The g77-built MCTDH is now run with 5 MPI-processes and 4 POSIX-threads per MPI-process i.e. 20 cores are used. The specific command to be given may depend on the queueing system of your computer and hence may differ from the example above.
To be able to start the MPI-parallel MCTDH the program must be compiled with some additional options. For the standard compilers (g77, gfortran, pgf77, and ifort) additional sections are included in the script compile.cnf. Depending on the compiler the command is:

\texttt{compile -m mctdh}

or

\texttt{compile -c mpi-xxx mctdh}

where \texttt{xxx} must be replaced by \texttt{g77, gfortran, pgf77, or intel}, respectively. If another compiler should be used an appropriate section in the compile.cnf script must be created. The \texttt{-m} option is a shorthand form. It requires that your default compiler is one out of the four above mentioned compilers. Note that it is essentially that your MPI package was created with a compiler compatible to the compiler you would like to use \(^3\).

\(^3\)Try to check \texttt{mpif77 \(-v\)} and see the manpages for \texttt{mpif77}.
Chapter 4

Selecting a DVR/FBR-representation for the primitive basis

DVR/FBR-representations are used to set up the single-particle functions of an MCTDH or the product grid of a numerically exact calculation. The DVR/FBR basis is also called primitive basis in the following. A comprehensive discussion of the DVR/FBR technique can be found in Appendix B of the review [1].

4.1 Available DVR/FBR-representations

The DVR/FBR-representations that are implemented in the MCTDH package are compiled in Tab. 4.1, along with examples for typical applications of them. Also given are the keywords by which the corresponding primitive basis types are selected in the input file.

The representations of the primitive basis are specified in the PRIMITIVE-BASIS-SECTION of the input file. The program evaluates this section and stores the information in the dvr file. The PRIMITIVE-BASIS-SECTION also serves to define the system coordinates, by allocating a label to each degree of freedom in the system. These labels are then used in other sections, such as the INIT_WF-SECTION, the SPF-BASIS-SECTION, and the HAMILTONIAN-SECTION to map the different sets of information onto the system coordinates.

A complete input file — for the photo-dissociation process of NOCl — is shown in Example 4.1. There, three Jacobian coordinates are defined, labelled rd (dissociative degree of freedom), rv (vibrational degree of freedom), and theta (angular degree of freedom). Their DVR-representations are sine, Hermite, and Legendre, respectively. The parameters associated with these representations are explained in the following sections.

4.2 Hermite and radial Hermite DVR

In the Hermite or harmonic oscillator DVR the harmonic oscillator functions

$$\chi_j(x) = (2^j j!)^{-1/2} (m_\omega/\pi)^{1/4} H_j \left( \sqrt{m_\omega} (x - \tilde{x}) \right) e^{-m_\omega(x-\tilde{x})^2/2}$$

(4.1)

are taken as basis functions. The Hermite DVR is thus typically used for vibrational modes. In the above equation $H_j$ denotes the $j$th Hermite polynomial and $j$ starts from zero,
4.2 Hermite and radial Hermite DVR

Table 4.1: The DVR/FBR-representations for the primitive basis which are available in the MCTDH package. Also displayed are the corresponding keywords that are used in the input file, and a typical application for each basis type.

<table>
<thead>
<tr>
<th>DVR/FBR-representation</th>
<th>Keyword</th>
<th>Typical applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermite (harmonic osci.) DVR</td>
<td>HO</td>
<td>Vibrational modes</td>
</tr>
<tr>
<td>Radial Hermite DVR</td>
<td>rHO</td>
<td>Vibrational modes</td>
</tr>
<tr>
<td>Legendre (rotator) DVR</td>
<td>Leg</td>
<td>Angular modes ($\theta$)</td>
</tr>
<tr>
<td>Restricted Legendre DVR</td>
<td>Leg/R</td>
<td>Angular modes ($\theta$)</td>
</tr>
<tr>
<td>Sine DVR</td>
<td>sin</td>
<td>Vibrational, angular, and dissociative modes</td>
</tr>
<tr>
<td>Laguerrre DVR</td>
<td>Lagu</td>
<td>Boundary condition $\varphi(r) \sim r^{1/2}$</td>
</tr>
<tr>
<td>Exponential (plane-wave) DVR</td>
<td>exp</td>
<td>Periodic boundaries, $\phi$-angular and diffractive modes</td>
</tr>
<tr>
<td>Fast Fourier transform (FFT)</td>
<td>FFT</td>
<td>Dissociative modes with large grids</td>
</tr>
<tr>
<td>Spherical harmonics FBR</td>
<td>sphFBR</td>
<td>Combined ($\theta, \phi$) angular modes</td>
</tr>
<tr>
<td>Extended Legendre DVR</td>
<td>KLeg</td>
<td>Combined ($\theta, \phi$) angular modes</td>
</tr>
<tr>
<td>Two–Dimensional Legendre DVR</td>
<td>PLeg</td>
<td>Combined ($\theta, \phi$) angular modes</td>
</tr>
<tr>
<td>Three–Dimensional rotational DVR</td>
<td>wigner</td>
<td>Combined ($\alpha, \beta, \gamma$) angular modes</td>
</tr>
</tbody>
</table>

\[ j = 0, 1, \ldots, N - 1. \] Note that the Hermite DVR depends only on the product $m\omega$, which determines the width, and on $\tilde{x}$, which defines the centre of the grid.

A variant of the Hermite DVR is the radial Hermite DVR, which is an appropriate DVR when the wavefunction is defined on a half-axis $[\tilde{x}, \infty)$ only, and satisfies the boundary condition $\psi(\tilde{x}) = 0$. The odd harmonic oscillator functions,

\[ \chi_j(x) = \sqrt{2} \chi_{2j-1}^{\text{HO}}(x), \]

are chosen as basis, with $j = 1, \ldots, N$ and $\chi_k^{\text{HO}}$ given by Eq. (4.1).

A Hermite DVR may be selected in two ways. One is given by the following PRIMITIVE-BASIS-SECTION:

```
PRIMITIVE-BASIS-SECTION
  X  HO  36  0.00  0.10  1822.89
  Y  HO  36  0.00  2.721,eV  1.0,AMU
end-primitive-basis-section
```

Here we have assumed that the system under consideration has two degrees of freedom, labelled $X$ and $Y$. The keyword $\text{HO}$ specifies the harmonic oscillator DVR. The next entry denotes the number $N$ of basis functions or, equivalently, grid points. The next three numbers define the equilibrium position $\tilde{x}$, the frequency $\omega$, and the mass $m$. If the mass entry is missing, the program sets $m$ to 1.

The above example also demonstrates the use of units. The default unit for times is femtoseconds, for all other input variables it is atomic units. (Hence exactly the same DVR is selected for the two modes $X$ and $Y$.) A complete list of the available units can be found in the HTML documentation.

The second way to choose a Hermite DVR is to specify the first and last grid point, by employing the keyword $\text{xi-xf}$:

```
PRIMITIVE-BASIS-SECTION
  X  HO  36  xi-xf  -0.528  0.528
  Y  HO  36  xi-xf  -0.528  0.528
end-primitive-basis-section
```
Example 4.1: An input file for a wavepacket propagation on the S1 surface of NOCl.

Here the two parameters define the grid points $x_1$ and $x_N$. The program then computes the corresponding product $m \omega$, as well as the equilibrium position $\tilde{x} = (x_1 + x_N)/2$. The above example is thus equivalent to the previous one.

A radial Hermite DVR can be selected in exactly the same way than a Hermite DVR, but with $rHO$ rather than $HO$ as keyword.

### 4.3 Legendre DVR

A Legendre or rotator DVR is employed for angular degrees of freedom $\theta$ because the associated Legendre functions $P_l^m(\cos \theta)$ are eigenfunctions of the angular momentum operator $\hat{l}^2$. The basis functions are thus the $L^2$-normalised associated Legendre functions,

$$ \chi_{l-m+1}(\theta) = \sqrt{\frac{2l + 1}{2}} \frac{(l-m)!}{(l+m)!} P_l^m(\cos \theta), $$

(4.3)
with \( m \geq 0 \) and \( l \) restricted to \( m \leq l \leq m + N - 1 \). The parameter \( m \) denotes the magnetic quantum number and is treated as a fixed parameter. The associated Legendre function \( P_l^m \) is given by the polynomial

\[
P_l^m(x) = \frac{(-1)^m}{2^m m!} \frac{d^{l+m}}{dx^{l+m}} (x^2 - 1)^l,
\]

for \( 0 \leq m \leq l \).

A Legendre DVR is selected for a coordinate named \( \text{theta} \) by the line

```
theta Leg 60 0 all
```

in the PRIMITIVE-BASIS-SECTION. The first number specifies the number \( N \) of basis functions or grid points, the second one denotes the magnetic quantum number \( m \). The last keyword controls which symmetry is to be used. For all, all values \( l = m, m + 1, \ldots, m + N - 1 \) are employed, for odd/even, only odd/even values of \( l \) are taken, i.e. \( l = m, m + 2, \ldots, m + 2N - 2 \) or \( l = m + 1, m + 3, \ldots, m + 2N - 1 \).

### 4.4 Sine DVR

The sine or Colbert-Miller DVR uses the particle-in-a-box eigenfunctions as a basis. The box boundaries are \( x_0 \) and \( x_{N+1} \), and \( L = x_{N+1} - x_0 \) denotes the length of the box. The basis functions are thus

\[
\chi_j(x) = \begin{cases} 
\sqrt{2/L} \sin \left(j \pi (x - x_0)/L\right) & \text{for } x_0 \leq x \leq x_{N+1} \\
0 & \text{else}
\end{cases}
\]

Note that the grid boundaries \( x_0 \) and \( x_{N+1} \) do not belong to the grid since the wavefunction vanishes there by construction. The sine DVR has been successfully used in MCTDH calculations for vibrational, angular, and dissociative degrees of freedom.

A sine DVR is turned on for a coordinate named, say, \( r \) by the line

```
r sin 24 1.0 24.0 short
```

in the PRIMITIVE-BASIS-SECTION. Again the first number denotes the number \( N \) of grid points. The meaning of the next two numbers depends on the last (optional) keyword, which can be short (the default) or long. When short is selected, the two numbers are the first and last grid point, \( x_1 \) and \( x_N \). If the keyword long is present, the two numbers specify the box boundaries, \( x_0 \) and \( x_{N+1} \). The line

```
r sin 24 0.0 25.0 long
```

is hence equivalent to the previous one. The grid spacing is \( \Delta x = (x_N - x_1)/(N - 1) \) in the case of short and \( \Delta x = (x_{N+1} - x_0)/(N + 1) \) in the case of long.

### 4.5 Exponential DVR and fast Fourier transform

The exponential DVR uses plane waves as basis functions. It is therefore often used for dissociative degrees of freedom. Moreover, exponential DVR and FFT are the only primitive-basis representations within the MCTDH program that satisfy periodic boundary conditions. These occur for instance for angular motion or motion parallel to a corrugated surface.
As our implementation of the exponential DVR requires an odd number of basis functions, we set \( N = 2n + 1 \). The basis functions are then written as

\[
\chi_j(x) = L^{-1/2} \exp \left( 2i\pi j(x - x_0)/L \right),
\]

with \(-n \leq j \leq n\) and \( L = x_N - x_0 \). The wavefunctions to be represented satisfy periodic boundary conditions, \( \psi(x_0) = \psi(x_N) \).

The fast Fourier transform (FFT) method may be considered as an exponential DVR where, however, the derivative matrices are not built but the action of them on the wavefunction is evaluated by two FFTs. The MCTDH program uses a Temperton FFT which allows one to use grids the length of which can be factorised into powers of 2, 3, and 5, i.e. \( N = 2^j3^k5^l \), where \( j, k, \) and \( l \) are non-negative integers. For optimal performance, one should work with a grid length of \( N = 2^j3^k \).

Although the exponential DVR and the fast Fourier transform (FFT) are formally equivalent, this does not hold for their efficiency. It is our experience that the exponential DVR performs faster than FFT for small grids \((N \lesssim 16)\), while FFT is faster for large grids \((N \gtrsim 64)\). Between these limits both representations are similarly fast.

FFT and exponential DVR have an identical set of input parameters. Supposed there are two degrees of freedom labelled \( X \) and \( Y \), an exponential DVR and an FFT representation are employed for these coordinates by a PRIMITIVE-BASIS-SECTION reading

```
PRIMITIVE-BASIS-SECTION
  X exp 25 2.2 5.8 linear
  Y fft 48 1.5 4.3
end-primitive-basis-section
```

As usual, the first number defines the number \( N \) of grid points. Remember that \( N \) must be odd for an exponential DVR and factorise into powers of 2, 3, and 5 (better 2 and 3 only) for FFT. If the last keyword is missing or linear, then the remaining two numbers, which will be called \( x_i \) and \( x_f \) in the following, are interpreted as the grid points \( x_0 = x_i \) and \( x_{N-1} = x_f \). The grid spacing is \( \Delta x = (x_f - x_i)/(N-1) \). Due to the periodic boundary conditions the first grid point, \( x_0 \), and the point \( x_N \), which is the one following the last point on the grid, are to be identified.

Instead of linear, the keywords periodic or s-periodic may be specified, which changes the interpretation of \( x_i \) and \( x_f \). In both cases, \( x_i \) and \( x_f \) are considered identical due to the periodic boundary conditions. The grid spacing is now \( \Delta x = (x_f - x_i)/N \). For periodic the parameter \( x_f \) is taken as \( N \)th grid point, \( x_N = x_f \). For s-periodic the grid points are additionally shifted by \( \Delta x/2 \), i.e. \( x_0 = x_i + \Delta x/2 \) and \( x_N = x_f + \Delta x/2 \). As an example, the lines

```
  X fft 32 0.00 3.0434 linear
```

and

```
  X fft 32 0.00 3.1416 periodic
```

are equivalent.

The keywords periodic and s-periodic are particularly useful to describe angular degrees of freedom. For angular modes ranging from 0 to some integer fraction of \( 2\pi \), one may alternatively use the keywords 2pi or s-2pi. The example above is thus equivalent to

```
  X fft 32 2pi/2
```

For a more detailed discussion see the HTML documentation.
4.6 Advanced topic: Spherical harmonics FBR

The spherical harmonics FBR is the appropriate choice when there is rotational motion which must be described by two angles, $\theta$ and $\phi$. The spherical harmonic functions

$$Y_{jm}(\theta, \phi) = \sqrt{\frac{2j + 1}{4\pi}} \frac{(j - m)!}{(j + m)!} P_j^m(\cos\theta) e^{im\phi}$$

serve as basis functions, where $P_j^m$ denotes the Legendre polynomial (4.4). The matrix elements of the angular momentum operators $\hat{j}^2$, $\hat{j}^+, \hat{j}^-$, and $\hat{j}_z$ are then given by simple formulas.

Examples for a PRIMITIVE-BASIS-SECTION defining a spherical harmonics FBR for the set of coordinates $\alpha$ and $\beta$ are

PRIMITIVE-BASIS-SECTION
alpha sphFBR 9 nosym
beta phiFBR
end-primitive-basis-section

and

PRIMITIVE-BASIS-SECTION
alpha sphFBR 8 sym
beta phiFBR 4 2
end-primitive-basis-section

The keyword sphFBR selects the spherical harmonics FBR and the label phiFBR indicates the second coordinate on which the FBR is based. The number $j_{\text{max}}$ after the keyword sphFBR or defines the maximum value for $j$. The $j$ values are $j = 0, 1, \ldots, j_{\text{max}}$ for nosym and $j = 0, 2, \ldots, j_{\text{max}}$ or $j = 1, 3, \ldots, j_{\text{max}}$ for sym, depending on the parity of $j_{\text{max}}$. Without the optional data $m_{\text{max}}$ (and possibly also $\Delta m$) given, $m$ takes the values $m = -j, -j+1, \ldots, -1, 0, 1, \ldots, j - 1, j$. With $m_{\text{max}}$ (and possibly also $\Delta m$) given, $m$ takes the values $m = -\min(m_{\text{max}}, j), -\min(m_{\text{max}}, j) + \Delta m, \ldots, \min(m_{\text{max}}, j) - \Delta m, \min(m_{\text{max}}, j)$. In the second example we thus have $j$-values of $j = 0, 2, 4, 6, 8$. The corresponding values for $m$ are $m = 0$ for $j = 0$, $m = -2, 0, 2$ for $j = 2$, and $m = -4, -2, 0, 2, 4$ for $j = 4, 6, 8$, giving an overall number of 19 basis functions.

The order in which the primitive basis sets are declared is, in general, arbitrary. However, a sphFBR line must be followed directly by a phiFBR line. Moreover, the corresponding degrees of freedom ($\alpha$ and $\beta$ in the example above) must be declared as combined in the SPF-BASIS-SECTION.

4.7 Advanced topic: Restricted Legendre DVR

The restricted Legendre DVR, $\text{Leg/R}$, is very similar to the (ordinary) Legendre DVR, but can make use of the fact that the angular motion may be restricted to an interval smaller than $(0, \pi)$. In such a case one may drop the unused grid-points, assuming that the wavefunction vanishes there. The FBR/DVR transformation matrix is now rectangular rather than square. The propagation, however, is performed exclusively in the (smaller) set of DVR grid-points and some speed-up is obtained. For example, one may replace the theta line in the PRIMITIVE-BASIS-SECTION of the noc11.inp input file (see Example 4.1 and the Tutorial) to
Selecting a DVR/FBR-representation for the primitive basis

The two last numbers define the range (in radians) to be covered by the grid-points. In the log file one finds

```
****** Primitive Basis ********
```

<table>
<thead>
<tr>
<th>mode</th>
<th>kappa</th>
<th>DVR</th>
<th>N</th>
<th>xi</th>
<th>xf</th>
<th>dx</th>
<th>p-max</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd</td>
<td>1</td>
<td>sin</td>
<td>36</td>
<td>3.800</td>
<td>5.600</td>
<td>0.051429</td>
<td>59.436</td>
</tr>
<tr>
<td>rv</td>
<td>2</td>
<td>HO</td>
<td>24</td>
<td>1.620</td>
<td>2.652</td>
<td>0.044849</td>
<td>70.171</td>
</tr>
<tr>
<td>theta</td>
<td>3</td>
<td>Leg/R</td>
<td>27/60</td>
<td>2.739</td>
<td>1.389</td>
<td>0.051924</td>
<td>m= 0, sym= 0</td>
</tr>
</tbody>
</table>

The last line tells us, that there are 60 FBR functions but only 27 DVR grid-points. For technical reasons, the locations of the first and last grid-points differs slightly from the inputted numbers. Comparing the timing files, one finds that the restricted Legendre DVR reduces the total CPU time by a factor of 1.4 and reduces the time spend for propagating the single-particle functions of the theta degree of freedom by more than a factor of 4.

The analyse programs `plgpop` and `showd1d85` are useful for determining an appropriate theta-range. Try `plgpop -z 1.e-10 2 3` and `showd1d85 -a -y 0.000001 f3`.

When using the restricted Legendre DVR it should be the last entry in the PRIMITIVE-BASIS-SECTION. Due to a bug it otherwise sometimes performs incorrectly.

### 4.8 Advanced topic: Extended Legendre DVR (KLeg) and Two-Dimensional Legendre DVR (PLeg)

Similarly to the spherical harmonics FBR, the extended Legendre DVR employs the spherical harmonics as basis set. It thus defines a two-dimensional representation. As the name indicates, there is a FBR/DVR transformation from the angular momentum indices \( \ell \) to grid points \( \theta_i \). There is, however, no such transformation for the \( \{m, \phi\} \) pair. The extended Legendre DVR will typically be used to describe angular motion of a molecule with total angular momentum \( J > 0 \). The projection of the angular momentum of the angular motion under discussion onto the body fixed frame is usually called \( K \) or \( \Omega \) (rather than \( m \)). The angle \( \phi \), which is the coordinate canonical conjugate to \( K \), is an Euler angle and does not appear in the potential. There is thus no point to perform a \( K \leftrightarrow \phi \) transformation. The 2D single-particle functions, \( \varphi(\theta, K) \), are hence given in a mixed DVR–FBR representation.

An example for a PRIMITIVE-BASIS-SECTION defining an extended Legendre DVR for the set of coordinates theta and \( K \) is given by

```
PRIMITIVE-BASIS-SECTION
theta  KLeg  31  even
K   K    -4   4
end-primitive-basis-section
```

The keyword `KLeg` selects the extended Legendre DVR and the label \( K \) indicates the second coordinate on which the KLeg representation is based. Similarly to the spherical harmonics FBR, the KLeg line **must** be followed directly by a \( K \) line, and the corresponding degrees of freedom (here theta and \( K \)) must be declared as combined in the SPF-BASIS-SECTION. The first number after `KLeg` specifies the number \( N \) of \( \theta \)-grid points, the following keyword
controls which symmetry is to be used. For all values \( \ell = K_1, K_1 + 1, \ldots, K_1 + N - 1 \) are employed, for odd/even, only odd/even values of \( \ell \) are taken, i.e. \( \ell = K_1, K_1 + 2, \ldots, K_1 + 2N - 2 \) or \( \ell = K_1 + 1, K_1 + 3, \ldots, K_1 + 2N - 1 \). Here, \( K_1 \) denotes the minimum of \( |K| \), i.e. \( K_1 = 0 \) if \( K_{\min} \cdot K_{\max} \leq 0 \) and \( K_1 = \min(|K_{\min}|, |K_{\max}|) \) else. The numbers following \( K \) are the minimal, \( K_{\min} \), and maximal, \( K_{\max} \), values for \( K \). The \( K \)-grid thus consists of 9 points for the example above.

The two-dimensional Legendre DVR, \( \text{PLeg} \), is similar to \( \text{KLeg} \), except that a Fourier-transformation from \( K \) to \( \phi \) is performed. An example for a PRIMITIVE-BASIS-SECTION defining a \( \text{PLeg} \) two-dimensional Legendre DVR for the set of coordinates \( \theta \) and \( \phi \) is given by

```plaintext
PRIMITIVE-BASIS-SECTION
  theta PLeg 31 even
  phi  exp 11 2pi
end-primitive-basis-section
```

The keyword \( \text{PLeg} \) selects the two-dimensional Legendre DVR and the label \( \phi \) indicates the second coordinate on which the \( \text{PLeg} \) representation is based. Similarly to \( \text{KLeg} \) and to the spherical harmonics FBR, the \( \text{PLeg} \) line must be followed directly by a \( \text{exp} \) line, and the corresponding degrees of freedom (here \( \theta \) and \( \phi \)) must be declared as combined in the SPF-BASIS-SECTION. The number of grid-points of the exponential DVR must be odd. The 11 points chosen here allow to represent \( K \) in the interval \(-5 \leq K \leq 5\).

### 4.9 Advanced topic: Three-Dimensional rotational DVR (Wigner)

Wigner-DVR uses the \( L^2 \)-normalized Wigner-D functions, \( \overline{D}^j_{m,k}(\alpha, \beta, \gamma) \), as a basis set. These are defined as:

\[
\overline{D}^j_{m,k}(\alpha, \beta, \gamma) = \sqrt{\frac{2j + 1}{8\pi^2}} D^j_{m,k}(\alpha, \beta, \gamma) \tag{4.8}
\]

\[
D^j_{m,k}(\alpha, \beta, \gamma) = e^{-im\alpha} d^j_{m,k}(\beta) e^{-ik\gamma} \tag{4.9}
\]

where \( \alpha \) and \( \gamma \) are the Euler angles representing rotation around the space-fixed (SF) and body-fixed (BF) z-axes, respectively, and \( \beta \) is the Euler angle between the SF and BF z-axes (where the z-y-z right-handed axis convention is used). The Wigner-(small)-d function, \( d^j_{m,k}(\beta) \), is defined as:

\[
d^j_{m,k}(\beta) = \langle j, m | e^{-i\beta J_y} | j, k \rangle \tag{4.10}
\]

Wigner-DVR defines a three-dimensional representation which can be used to model rotation of polyatomic molecules. An FBR/DVR transform is used to convert between the angular momentum index \( j \) and the grid points \( \beta \). The \( \{m, \alpha_i\} \) and \( \{k, \gamma_i\} \) momentum-coordinate conjugate pairs can be used in either momentum or coordinate representation in MCTDH; in the latter case a discrete Fourier transform switches between grid points and momentum indices.

A PRIMITIVE-BASIS-SECTION input block defining Wigner-DVR uses three input lines, one for each Euler angle. For example:
The keyword `wigner` is used to select the $L^2$-normalized Wigner-(small)-$d$ functions as the basis functions for the $\beta$-angle. The number following the `wigner` keyword denotes the number of $j$-values to be used, and the `all` keyword denotes that both odd and even $j$-values are to be included. For the $\gamma$ and $\alpha$ angles, the coordinate representation is declared by an `exp` line; alternatively, a `K` line can be used to declare the momentum representation. In the example above, the `alpha` DOF is used in coordinate representation, while the `gamma` DOF is used in momentum representation. The input line for the $\beta$ angle must appear first and be immediately followed by a line for the $\gamma$ angle, which must then be followed by a line for the $\alpha$ angle; that is, the order of the degrees of freedom must be given as $|J, K, M\rangle$. All three DOF must be declared as combined in the SPF-BASIS-SECTION.
Chapter 5

Defining the single-particle basis

In an MCTDH calculation (propagation or relaxation) the single-particle basis has to be defined. This is done in the SPF-BASIS-SECTION of the input file. The SPF-BASIS-SECTION also enables one to treat degrees of freedom as a combined mode, or to select only a subset of the system degrees of freedom.

5.1 Specifying the number of single-particle functions

In the SPF-BASIS-SECTION the number $n_\kappa$ of single-particle functions to be used for each degree of freedom $\kappa$ of the system are listed. Supposed that there are two degrees of freedom, the SPF-BASIS-SECTION

```
SPF-BASIS-SECTION
X = 3
Y = 3
end-sbasis-section
```

assigns three single-particle functions to each mode. Here $X$ and $Y$ are the mode labels which must coincide with those in the PRIMITIVE-BASIS-SECTION.

If only one single-particle function is used for each degree of freedom, a calculation actually employs the time-dependent Hartree (TDH) method rather than the MCTDH scheme. Another special case is obtained for equal numbers of single-particle and primitive basis functions in each degree of freedom $\kappa$, i.e. $n_\kappa = N_\kappa$, with $N_\kappa$ being the number of primitive basis functions. An MCTDH calculation is then equivalent to a numerically exact one, but note that numerically exact calculations can be performed much more efficiently by using the exact keyword (see Sec. 3.6).

Typical numbers of single-particle functions range from $n_\kappa = N_\kappa/20$ to $n_\kappa = N_\kappa/3$. Note that the numbers of single-particle functions should obey

$$n_\kappa^2 \leq \prod_{\kappa' = 1}^{f} n_{\kappa'}, \quad (5.1)$$

since otherwise there will be redundant configurations in the MCTDH wavefunction. Hence, one must choose $n_1 = n_2$ if there are two degrees of freedom.
5.2 Advanced topic: Selecting degrees of freedom from a large system

For some systems it may be useful to treat only a subset of the specified coordinates. The program allows one to select coordinates simply by listing only those required in the SPF-BASIS-SECTION. For instance, if five modes $V$, $W$, $X$, $Y$, and $Z$ are specified in the PRIMITIVE-BASIS-SECTION

```
PRIMITIVE-BASIS-SECTION
  V  HO  16  0.0d0  1.0d0  1.0d0
  W  HO  22  0.0d0  1.0d0  1.0d0
  X  HO  32  0.0d0  1.0d0  1.0d0
  Y  HO  21  0.0d0  1.0d0  1.0d0
  Z  HO  12  0.0d0  1.0d0  1.0d0
end-primitive-basis-section
```

and you wish to include only $W$, $X$, and $Z$ in your calculations, this can be accomplished by the SPF-BASIS-SECTION

```
SPF-BASIS-SECTION
  W = 7
  X = 8
  Z = 6
end-spf-basis-section
```

(or by commenting out the corresponding lines for $V$ and $Y$).

The approximation being made here is that the coupling between the included and excluded modes is negligible. As a result, the Hamiltonian is built simply ignoring all terms that include contributions from the excluded degrees of freedom.

5.3 Combining modes to produce multi-dimensional single-particle functions

For large systems, or when certain degrees of freedom are strongly coupled, it may be advantageous to combine degrees of freedom together and use multi-mode single-particle functions (see Sec. 4.5 in Ref. [1] for further details).

A combination scheme can be easily specified by grouping together the degrees of freedom to be combined in the SPF-BASIS-SECTION. As an example we consider a system consisting of eight degrees of freedom named $r_1$, $r_2$, $r_3$, $r_4$, $r_5$, $r_6$, $r_7$, $r_8$. The SPF-BASIS-SECTION

```
SPF-BASIS-SECTION
  r1, r4 = 15
  r2, r3, r7 = 10
  r5, r6, r8 = 12
end-spf-basis-section
```

then combines, for instance, the modes $r_1$ and $r_4$ together, and treats them as a single coordinate in the calculation. The single-particle functions for this coordinate are then two-dimensional functions in the system coordinates. The number of two-dimensional single-particle functions is 15 for the combined $r_1$--$r_4$--mode.

In the example given, eight modes have been combined together to produce three modes for the MCTDH calculation. As the length of the expansion coefficient vector grows exponentially with the number of modes included in a calculation this drastically reduces the computational resources required.
It is important to choose a "good" combination scheme. One should combine those DOF, which are most strongly coupled with each other. Then, these correlations are taken care of at the SPF-level, and the number of SPFs, necessary for convergence, is decreased. Sometimes physical intuition and knowledge of the system helps to identify the strongly coupled DOFs, but most of the time, this information is missing. Then one takes a more practical approach. Firstly, one combines those DOFs which have similar frequencies, because modes with very different frequencies are less likely to couple strongly. Secondly, one should set up a combination scheme such, that the combined grids have similar size.

One must neither over- nor under-combine. The A-vector length should be larger than the number of data points needed to represent the SPFs. Otherwise it is likely that one over-combines. It is always useful to inspect the timing file. The time needed to propagate the SPFs should take between 2% and 25% of the total effort (CMF scheme assumed).
Chapter 6

Setting up the Hamiltonian

For a quantum dynamical calculation a Hamiltonian operator has to be defined. This is typically done using the operator file, which is a text file that is read and interpreted by the MCTDH program. The MCTDH program is capable to parse a variety of mathematical expressions. This often allows the implementation of a new Hamiltonian without programming any routines.

6.1 The operator file

The operator file contains all the information the program needs to set up the Hamiltonian. It must have the extension .op, e.g. hamiltonian.op. Similarly to the input file, the operator file is divided into sections. In the OP_DEFINE-SECTION the Hamiltonian is characterised. Numerical constants can be specified in the PARAMETER-SECTION. The definition of the Hamiltonian is contained in the HAMILTONIAN-SECTION. Finally, any labels that are used to describe the Hamilton operator are compiled in the LABELS-SECTION. The operator file for the propagation of NOCl on the S1 surface is given as Example 6.1.

In order to select a particular operator file, an OPERATOR-SECTION is required in the input file. The keywords opname and oppath then point to this file. For instance, with

```
OPERATOR-SECTION
oppath = /usr/people/mctdh/operators
opname = noc1l
end-operator-section
```

the MCTDH or Potfit program would look in the directory /usr/people/mctdh/operators for the operator file noc1l.op. If the oppath item is not given, the program first looks in the startup directory (i.e. the directory where the input file is located), then in the directory specified by the default operator path. (The default operator path is displayed when typing mctdh85 -max). The operator files that are available in the MCTDH package are tabulated in the HTML documentation.

If you desire to create a new operator file, the first step is to write the OP_DEFINE-SECTION. In this section the Hamiltonian is given a title, between the keywords title and end-title. The title will be printed e.g. in the log file. In the NOCl example 6.1 the operator is labeled “NOCl S1 surface”, by employing the OP_DEFINE-SECTION
6.2 Defining numerical constants

There is no restriction on the format or number of lines used. The next step for creating a new operator file is described in the following section.

6.2 Defining numerical constants

A Hamiltonian typically contains some numerical constants, such as (reduced) masses, frequencies, or coupling strengths. In the PARAMETER-SECTION of the operator file, labels can be associated with these constants, which can then be used in the definition of the Hamiltonian. This makes the operator file not only easier to read but also very simple to change, as is depicted in Sec. 6.11.

For example, a PARAMETER-SECTION reading

```
PARAMETER-SECTION
mh = 1837.15 # hydrogen mass
mc = 21874.7 # carbon mass
end-parameter-section
```
defines the labels \(mh\) and \(mc\) as the masses of the hydrogen and carbon nuclei (in a.u.). Note that it is possible to specify the unit of a parameter. The just mentioned example can hence also be written

```
PARAMETER-SECTION
mh = 1.0, H-mass
mc = 12.0, AMU
end-parameter-section
```

The keywords \(H\)-mass and AMU stand for hydrogen mass and atomic mass unit. See the HTML documentation for available units. Furthermore, elementary algebraic operations can be performed in the PARAMETER-SECTION. Thus, the section

```
PARAMETER-SECTION
a = 1.0
b = 2.0
c = 3.0
d = 2.0*a+b*c+1.0
e = EXP[d/9.0-a]
end-parameter-section
```

sets \(d\) to 9.0 and \(e\) to 1.0. The algebraic expression must not contain spaces or brackets! Only the operators \(+\), \(-\), \(\star\), \(/\) are allowed. An exponent acts only on the label to which it is attached and \(\star\)/ are evaluated before \(+\)/-. Otherwise the order of operation is strictly from left to right. Exponents must be numbers (not parameters), they may be signed and real (e.g.: alpha\(^{-0.5}\) is a valid construct). The exponential form (e.g. 2.3d-5) is allowed for numbers only. The decimal exponent must be indicated by a \(d\), not by a \(D\), \(e\) or \(E\) ! See the HTML documentation (Hamiltonian-Documentation/Parameter-Section) for further details.

The last line of the example above demonstrates the use of functions in parameter arithmetic. Available functions are:

- \(\text{EXP}\), \(\text{LOG}\), \(\text{LOG10}\), \(\text{SIN}\), \(\text{COS}\), \(\text{TAN}\), \(\text{ASIN}\), \(\text{ACOS}\), \(\text{ATAN}\), \(\text{SINH}\), \(\text{COSH}\), \(\text{TANH}\), \(\text{ABS}\), \(\text{INT}\), \(\text{ATAN2}\), \(\text{MIN}\), \(\text{MAX}\).

The definitions of these functions are the usual FORTRAN definitions. Note that the last three functions depend on two arguments. The two arguments, which may be numbers, parameters or numerical expressions, must be separated by a blank or a comma. The keyword for the function must follow directly the equal sign, not even a minus is allowed in between. No operations must follow the closing bracket of the function, except possibly a unit.

Some labels have a special, pre-defined meaning. A complete list of these is given in the HTML documentation. Here we only mention the \(\text{mass}_x\) label (cf. Example 6.1). The value of this parameter is taken as the mass of the degree of freedom specified by \(x\) (as defined in the PRIMITIVE-BASIS-SECTION of the input file). By default the mass is otherwise set to 1.0 au. The \(\text{mass}_x\) label can then be used to define the kinetic energy, by employing the \(\text{KE}\) operator. The symbol \(\text{KE}\) belongs to the list of expressions the MCTDH program can interpret. These will be explained in the following section. Other parameters of special meaning are \(\text{PI}\), \(j\text{tot}\), and \(j\text{bf}\).

Remarks:

- Parameters cannot be re-defined. A second definition is simply ignored (but protocoted in the log file). E.g. if a parameter is defined on the command line, then a following re-definition in the operator-file will be ignored. This, however, makes operations like
6.3 Using symbolic expressions to define the Hamiltonian

The MCTDH and Potfit programs are able to parse a variety of mathematical symbols, such as powers, exponential and trigonometric functions, and first and second derivatives. A complete list of these built-in symbols can be found in the HTML documentation. A selection of these symbols is compiled in Table 6.1. For a complete list, please refer to App. C. The symbolic expressions are used in the HAMILTONIAN-SECTION to define the Hamiltonian.

Within the MCTDH framework the Hamiltonian must in general be given in product form,

\[ H = \sum_{r=1}^{s} c_r h_r^{(1)} \cdot \ldots \cdot h_r^{(f)}, \]  

(6.1)

where \( f \) and \( s \) denote the numbers of degrees of freedom and Hamiltonian terms, respectively. This structure is reflected in the HAMILTONIAN-SECTION, which is mainly a table with \( s \) lines and \( f+1 \) columns, containing the coefficients \( c_r \) and single-particle operators \( h_r \).

To make things concrete, let us explain the usage of the symbolic expressions with the aid of the following two examples. The first one is a fairly simple one, namely a modified Henon-Heiles Hamiltonian, i.e. two coupled anharmonic oscillators.

The Hamiltonian is

\[ H = -\frac{1}{2m} \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) + \frac{x^2 + y^2}{2} + \lambda \left( x y^2 - \frac{x^3}{3} \right) + \lambda^2 \left( \frac{x^4}{16} + \frac{x^2 y^2}{8} \right). \]  

(6.2)

The modification with respect to the original Henon-Heiles Hamiltonian is the last (quartic) term. It makes the system bound. The corresponding operator file is included as Example 6.2. As one can see, the Hamiltonian (6.2) is represented by the symbols in the HAMILTONIAN-SECTION, one product term per line. The mode labels have to match with those in the
Table 6.1: A selection of built-in symbolic expressions that can be used to define the Hamiltonian. (For a complete list please refer to the HTML documentation or to Appendix C.) The variables $x$ and $\theta$ represent the mode labels associated with the corresponding degrees of freedom.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>$1$</td>
<td>Unit operator</td>
</tr>
<tr>
<td>$q$</td>
<td>$x$</td>
<td>Multiply by position coordinate $x$</td>
</tr>
<tr>
<td>$q^r$</td>
<td>$x^r$</td>
<td>Multiply by $r$th power of $x$</td>
</tr>
<tr>
<td>$\sin$</td>
<td>$\sin x$</td>
<td>Sine of coordinate</td>
</tr>
<tr>
<td>$\cos$</td>
<td>$\cos x$</td>
<td>Cosine of coordinate</td>
</tr>
<tr>
<td>$\tan$</td>
<td>$\tan x$</td>
<td>Tangent of coordinate</td>
</tr>
<tr>
<td>$\exp$</td>
<td>$e^x$</td>
<td>Exponential of coordinate</td>
</tr>
<tr>
<td>$dq$</td>
<td>$\partial_x$</td>
<td>First derivative</td>
</tr>
<tr>
<td>$dq^n$</td>
<td>$\partial_x^n$</td>
<td>$n$th derivative$^a$</td>
</tr>
<tr>
<td>$KE$</td>
<td>$-\frac{1}{2m} \partial_x^2 x$</td>
<td>Kinetic energy term$^b$</td>
</tr>
<tr>
<td>$j^2$</td>
<td>$-\frac{1}{\sin^2 \theta} \partial_\theta \sin \theta \partial_\theta$</td>
<td>Angular momentum squared$^c$</td>
</tr>
</tbody>
</table>

$^a$Derivatives with $n > 2$ are only allowed for an FFT primitive-basis.

$^b$The variable $m$ stands for the $mass_x$ label defined in the PARAMETER-SECTION.

$^c$The given formula is only valid if a Legendre-DVR with magnetic quantum number $m = 0$ is used.

Example 6.2: An operator file for a wavepacket propagation using the modified Henon-Heiles Hamiltonian.
6.4 Defining labels

PRIMITIVE-BASIS-SECTION of the input file. The coupling parameter $\lambda$ and the masses used for the $\text{KE}$ keyword are defined in the PARAMETER-SECTION.

The second example is the kinetic energy of a three-atomic molecule with total angular momentum $J = 0$ described by Jacobian coordinates $r_1$, $r_2$, and $\theta$. The kinetic energy reads

$$T = -\frac{1}{2\mu_1} \frac{\partial^2}{\partial r_1^2} - \frac{1}{2\mu_2} \frac{\partial^2}{\partial r_2^2} - \frac{1}{2} \left( \frac{1}{\mu_1 r_1^2} + \frac{1}{\mu_2 r_2^2} \right) \frac{\partial}{\partial \theta} \sin \theta \frac{\partial}{\partial \theta}, \quad (6.3)$$

where $\mu_1$ and $\mu_2$ specify the reduced masses associated with $r_1$ and $r_2$. The representation of $T$ in the HAMILTONIAN-SECTION is given by

HAMILTONIAN-SECTION

\begin{tabular}{|c|c|c|c|}
\hline
modes & r1 & r2 & theta \\
\hline
1.0 & KE & 1 & 1 \\
1.0 & 1 & KE & 1 \\
0.5/mass_r1 & q^-2 & 1 & j^2 \\
0.5/mass_r2 & 1 & q^-2 & j^2 \\
\hline
\end{tabular}

end-hamiltonian-section

Here we have assumed that the keywords $\text{mass}_r1$ and $\text{mass}_r2$ have been correspondingly defined in the PARAMETER-SECTION. (See also Example 6.1). Note that all blank lines and all lines which start with ----- (5 minus) are ignored.

Another example, the operator file for a 4-mode model of pyrazine, is discussed in Sec. 9. This example also demonstrates how to treat non-adiabatic systems.

6.4 Defining labels

In the tableau of the HAMILTONIAN-SECTION there must appear only simple labels with or without exponents and products of those. I.e. $\cos^* q^{-2}$ is a valid entry but $q^* q^{-2}$ is not. Simply use two lines to perform the sum. Symbolic expressions with parameters must not appear in the tableau, one rather must link them to a simple label. This is done in a LABELS-SECTION. For example

LABELS-SECTION

bcw=expcos[1.1,theta0]
cap1=CAP[ 5.0 0.3 3 ]
dcos=cos1[2.0 5.0*t1]^2
V = natpot {name}
end-labels-section

See the Appendix C for a list of symbolic expressions. In the parameter list, $[\cdots]$, there may appear numbers, parameters or simple algebraic expressions of those. The different entries may be separated by a comma or a blank. The use of units is not allowed here. File names, etc. are given in curly brackets. If a path is relative it is interpreted as relative to the location of the input file. The entry name simply is a shortcut for the path of the name-directory.

A label may consist of upper or lower case letters (case sensitive!) and numbers. Even special characters like ".", ":", "$", "/", ",&", or ":?" are allowed. The underscore "." how-
ever has a special meaning. If one writes a label as

\[ \text{label}_{modelabel} \]

where modelabel denotes a modelabel of one of the dof, then the program puts the corresponding operator in the modelabel column of the HAMILTONIAN-SECTION, assuming unit operators for all other degrees of freedom. This feature is frequently used for complex absorbing potentials (CAPs). See Section 6.10 for an example. Remember not to make the underscore part of a label, except when using this modelabel feature.

### 6.5 Implementing user-defined 1D-operators

The examples in the previous section have underlined how easily a Hamiltonian operator can be implemented in many cases into the MCTDH program. Nevertheless, it might happen that a new 1D-operator is required for a particular problem. When the desired 1D-operator is a real function, i.e. a potential, then there are three simple ways to implement it. The first way is to define the function by a set of data points \( \{x_i, f(x_i)\} \). This data is written in ASCII format to a file called, say, data. When one includes in a LABELS-SECTION a statement like:

\[ V2=\text{external1d}\{\text{data}\} \]

then the data will be read by MCTDH, quadratically interpolated and assigned to the label \( V2 \). As usual, the path given as argument to external1d may be absolute or relative. In the latter case it is, as usual, relative to the location of the input file. MCTDH reads the data file (two columns, free format) till it finds an end-of-file. Blank lines and lines beginning with a hash '#' are ignored. The \( x \)-values have to be equally spaced. Extrapolation is not possible, hence the first \( x \)-point must be smaller than the first grid-point and the last \( x \)-point must be larger than the last grid-point. For optimal performance, there should be two data points below the first and above the last grid-point, respectively. The data points should be dense enough such that the interpolation error is negligible.

The second way is similar to the first one, but here the data must be given precisely at the grid points. A file of either ASCII or binary format must be generated such that the first line contains the data for the first grid point, the second line for the second point, etc. Let us call this file again data. The LABELS-SECTION should now contain the statement

\[ V2=\text{read1d}\{\text{data ascii}\} \]

where ascii is to be replaced by binary if the file data is in binary format. See also HTML documentation: “Hamiltonian Documentation”/“Labels-Section”.

The third way is to use the pre-defined label my1d and to edit the subroutine my1d which is on source/opfuncs/user1d.F. The label my1d is to be used in the same way as e.g. the label cos, i.e. with or without parameters. One may make use of up to five parameters.

Grid based 1D operators, i.e. matrix representations of 1D operators with respect to the DVR functions, may also be read in. See the HTML documentation “Hamiltonian Documentation” / “Labels-Section” for details.

### 6.6 Advanced topic: Defining new symbolic expressions

When a new operator with a new label is to be implemented, this operator must be added to the operators in the source/opfuncs directory. Operators are divided into four classes, which
are handled differently by the program. The first of these are grid based operators, such as the kinetic energy operator in a DVR basis, or a natural potential expansion operator, which are only defined with respect to a grid. The remaining three classes are all different types of analytic functions: complex functions (e.g. CAPs), multi-dimensional functions (e.g. non-separable potential energy surfaces), and real one-dimensional functions. In this and the following section we will give some examples as to how to implement real one-dimensional functions. The implementation of multi-dimensional functions is the topic of Sec. 6.8.

**Important note:** Since version 8.3.10 there are four new files: install/user_surfaces, install/user_surfsdef, opfuncs/user1d.F and opfuncs/usersrf.F. These were introduced to separate changes done by a user from changes done by the authors. This new procedure will simplify an update of the package. When implementing a new one-dimensional potential function please use now opfuncs/user1d.F. Otherwise the procedure remains unchanged.

There is a default example from MCTDH in the file opfuncs/user1d.F, showing how users can define their own one-dimensional potentials. There are three subroutines in this file: ufdef1d, ufunc1d, and my1d. Suppose we use the label “my1d” in the operator file

```
LABELS-SECTION
  my = my1d[p1,p2,p3,p4,p5]ˆp0
end-labels-section
```

The subroutine my1d will be executed during running MCTDH. Although there is no real potential coded in this subroutine, one can find instructions on how to implement the user-defined potentials. To call the subroutine my1d, the program has to identify the label “my1d.” This is done in the subroutine ufdef1d, where MCTDH reads the label and its parameters, e.g. p1. The arrays hoprpar and hopipar serve to pass real or integer arguments to the function definition in subroutine ufunc1d. Here we only use the real one. The program by default stores an exponent \( r \) in \( \text{hoprpar}(1) \) or \( \text{hopipar}(1) \), depending on its type. If any parameters \([p1,p2,...]\) given in square brackets are present, this is indicated by the counter \( np \) being greater than zero, and the parameters are automatically pushed to the array elements \( \text{hoppar}(i,np). \text{hoppar}(2,np)...... \). Since \( \text{hoprpar}(1) \) is reserved for the exponent, there is a shift of arguments between \( \text{hoprpar} \) and \( \text{hoppar} \). If no parameters have been specified, i.e. \( np=0 \), default values are taken for the parameters. Besides the issue of parameters, each label is given a different function number \( \text{ifunc} \) and the program will call different subroutine according to the \( \text{ifunc} \) number. For example, we have defined \( \text{ifunc} \) equals to 1 if the label is my1d in subroutine ufdef1d and in subroutine ufunc1d, subroutine my1d will be called if \( \text{ifunc} \) equals 1.

As an example, if you wish to add the cotangent function \( \cot(a \ast (x - b)) \) to the program, using the label \( \cot \). You would then have to edit both the subroutines ufdef1d and ufunc1d in the file opfuncs/user1d.F, and additionally write a subroutine of the cotangent function. First, the new function must be coded:

```
subroutine cot(x,v,a,b)
  C cot: \( \cot(a \ast (x - b)) \)
  C this subroutine is called in subroutine ufunc1d
  C the parameters are defined in subroutine ufdef1d
  real*8 x, v, a, b, r

  r=a*(x-b)
  v=1.d0/tan(r)
end subroutine
```
Any valid FORTRAN expression can be employed to define a new function. Then you have to define the label “cot” in the subroutine ufdef1d and give it a function number ifunc. Go to the end of the subroutine ufdef1d where the code reads like (near line 60 of file opfuncs/user1d.F)

```fortran
ifunc=1
C newfunc       ! This is, of course, just an example.
C elseif(label(1:ilbl) .eq. 'newfunc') then
C     ifunc=2

C-----------------------------------------------------------------------
C end of if loop
C-----------------------------------------------------------------------
endif
```

The next free function number is 2, so you should replace the three out-commented lines with

```fortran
C cot: cot(a*(x-b))
    elseif (label(1:ilbl) .eq. 'cot') then
        if (np .gt.0) then
            hoprpar(2)=hoppar(1,np)
            hoprpar(3)=hoppar(2,np)
        else
            hoprpar(2)=1.0d0
            hoprpar(3)=0.0d0
        endif
        ifunc=2
```

Finally, you have to modify the subroutine ufunc1d. Find the following comments

```fortran
C newfunc EXAMPLE
C Set here the routine you want to call
C
C elseif (ifunc .eq. 2) then
C     call newfunc(x,v)
```

and replace those comments by the following lines

```fortran
C cot: cot(a*(x-b))
    elseif (ifunc .eq. 2) then
        call cot(x,v,hoprpar(2),hoprpar(3))
```

This completes the modifications for implementing the cotangent function. After recompiling the program (type compile mctdh), the new label cot may be used in the HAMILTONIAN-SECTION. When the label cot is used with parameters, it must – as usual – be assigned to a simple label in a LABELS-SECTION (see Sec. 6.4).

### 6.7 Advanced topic: Implementing separable potentials

In favorable cases the potential energy surface may be expressed by the built-in symbols known to the program, as has been discussed in Sec. 6.3. If this is not possible, new symbolic
expressions have to be introduced as detailed in the previous section. In some cases, however, it may be more convenient to set up a set of labels specifically for a complicated separable potential, i.e.

\[ V(x_1, \ldots, x_f) = \sum_{k=1}^{s} \prod_{i=1}^{f} V_{ki}(x_i), \quad (6.4) \]

where \( f \) and \( s \) denote the numbers of degrees of freedom and Hamiltonian terms, respectively. Let us further assume that you provide a FORTRAN-routine, say `mypot`, stored in a file `source/opfuncs/mypot.f`, to evaluate the one-dimensional potentials \( v = V_{ki}(x) \) in dependence of \( k, i, \) and \( x \) (\( v \) and \( x \) in a.u.):

```fortran
subroutine mypot (k,i,x,v)
    integer k,i
    real*8 x,v
    ... 
    end
```

The first step then is to add this file to the operator library, by inserting the line

\[
$(AR_OPFUNCS)($(PATH_OPFUNCS)/mypot.o) \ 
\]

at the corresponding position into the install/Makefile.

The second step is to establish labels for the one-dimensional potentials. Although the choice of these labels is arbitrary, we strongly recommend the use of some systematic nomenclature, e.g. `mypot1:1`, `mypot2:1`, `mypot1:2`, etc., where the first number denotes \( k \) and the second \( i \). For instance, with \( s = 3 \) and \( f = 2 \) the potential would then be defined by the lines

1.0 | mypot1:1 | mypot1:2
1.0 | mypot2:1 | mypot2:2
1.0 | mypot3:1 | mypot3:2

in the HAMILTONIAN-SECTION.

Next, a link between these labels and the potential routine is needed. This is done by adding the lines

\[
8 \text{ call mypot(ipar(2),ipar(3),x,v)} \ 
\]

return
to subroutine `callanld` in the file `source/opfuncs/callanld.f`. The file number 8 should be the next free number in that subroutine. For this to work properly, you have to store the indices \( k \) and \( i \) in `ipar(2)` and `ipar(3)` before. (Remember that `ipar(1)` is reserved for an exponent, if present.) This can be accomplished by inserting a new subroutine `defmypot` into your `mypot.f` file:

```fortran
subroutine defmypot (label,file,k,i)
    integer file,k,i,a,b,c
    character(*) label
    if (label(1:5) .eq. 'mypot') then
        a = 5
        b = index(label,':')
        c = index(label,' ') 
        read(label(a+1:b-1),*) k 
        read(label(b+1:c-1),*) i 
    end if
```


The number 5 must match the number of characters in the name mypot, and file must be 300 plus the file number introduced above. Finally, add the lines

```fortran
    call defmypot(buffer,ifile,hopipar(2),hopipar(3))
    if (ifile .ne. 0) go to 99
```

to the subroutine defanld in the file source/opfuncs/callanld.f, and recompile.

We close in noting that it is possible — analogous to what was said in Sec. 6.6 — to pass additional parameters to the potential routine mypot, by employing the arrays hopipar, hoprpar, and hoppar.

### 6.8 Advanced topic: Implementing non-separable potentials (potential surfaces)

It is also possible to include non-separable potentials into the MCTDH program, i.e. potentials that cannot be written in the product form (6.4). Because the direct evaluation of a non-separable potential makes an MCTDH calculation extremely inefficient, they are typically used in numerically exact calculations (propagation, relaxation, or diagonalisation) or to generate a separable potential fit using the Potfit program. The MCTDH program can however use them as they are, which may be useful for checking purposes. Note that the use of a multi-dimensional potential is not of disadvantage, if it operates on the coordinates of one MCTDH particle (combined mode) exclusively. See Section 6.13.

Another application of non-separable potentials is the MCTDH method in combination with the CDVR scheme (see Sec. 8.4.3). If the CDVR method is to be used during the propagation, the keyword analytic_pes should be included in the OPERATOR-SECTION. This means that the generated potential operator will not be explicitly calculated on the primitive grid points, but will be stored in the oper file in an “analytic” form which can be evaluated on-the-fly at any point in coordinate space.

For the convenience of the user, there is already a dummy routine source/surfaces/mysrf.f and one merely places the code of the new potential energy surface there. When editing mysrf.f one finds a brief description of how to make the necessary modifications on the file source/opfuncs/usersrf.F. This simplified procedure is convenient for a quick implementation of a new surface. However, it does not allow to pass surface options to the program. If options are needed or if more than one potential energy surface is to be implemented, one has to go the proper, slightly more elaborate way described next.

However, the most easiest way to include a multi-dimensional potential into MCTDH or POTFIT is via the readsrf keyword. In the LABELS-SECTION of an MCTDH operator file there would appear the statement

```fortran
    Vmd=readsrf{data ascii}
```

where data gives the path of the data file and ascii might be replaced by binary, if the file is in binary format. For POTFIT one would write

```fortran
    pes=readsrf{data ascii}
```
6.8 Implementing non-separable potentials (potential surfaces)*

in the OPERATOR-SECTION of the POTFIT input file.

The file data must contain the potential energy values at the grid points, just the energies and one energy per line. The order is implicitly defined by the Primitive-Basis-Section, or (in mctdh) by a $|i&j&k\rangle$ (i,j,k=1,2,...) construct of the Hamiltonian-Section. Note: first index runs fastest. There is no check for the consistency of the data.

**Important note:** Since version 8.3.10 there are four more files, which were introduced to separate changes done by a user from changes done by the authors. This new procedure will simplify an update of the package. Rather than editing the Makefile, please edit now the two files `install/user_surfdef` and `install/user_surfaces`. Similarly, rather than editing `opfuncs/funcsrf.F`, please edit now the file `opfuncs/usersrf.F`. For implementing new 1D functions, please use the file `opfuncs/user1d.F`.

The implementation of a non-separable potential follows the same philosophy as that of separable potentials, now with `source/opfuncs/usersrf.F` being the relevant file to modify.

Let us again assume that you provide a FORTRAN-routine, say `newsurf`, stored in a file `source/surfaces/newsurf.f`, to evaluate the non-separable potential $v = V(\vec{x})$ in dependence of the coordinate vector $x(i) = \vec{x}_i$ ($v$ and $\vec{x}$ in a.u.):

```fortran
subroutine newsurf (x,v)
real*8 x(*),v
... 
end
```

As a first step one has to provide a default routine, in case the new surface is not linked. I.e. one creates a file `DEF_newsurf.f` on the directory `source/surfaces`, and writes to it e.g. (see `source/surfaces/` for examples)

```fortran
subroutine newsurf (x,v)
real*8 x(*),v
write(6,'(a,/a)') '###########################################',
+    'newsurf is not linked. Run compile with -i option.'
write(2,'(a,/a)') '###########################################',
+    'newsurf is not linked. Run compile with -i option.'
stop 
end
```

Note that the file `DEF_newsurf.f` may also need to include dummy routines for other subroutines which are on the `source/surfaces/newsurf.f` file. In particular, if the routine `newsurf` contains an entry point, the dummy routine must have a corresponding entry point as well. If the program compiles but does not link, after having added a new surface, it is likely that there is a mistake in the dummy routine.

The next step then is to ensure that these two file will be compiled, by inserting the line

```
SURFDF1 = $(AR_SURFDEF)($(PATH_SURFACES)/DEF_newsurf.o)
```

into `install/user_surfdef`. Finally insert

```
newsurf.o : $(PATH_SURFACES)/newsurf.f
 $(FC) $(FFLAGS) -c -o $@ $?
```

into `install/user_surfaces`. Note that the second line starts with a tab and not with a series of blanks. The two files `install/user_surfdef` and `install/user_surfaces` will be sourced (i.e. read) by the Makefile. They hence contain the personal additions to the Makefile. Similarly, the
files opfuncs/user1d.F and opfuncs/usersrf.F contain the personal additions to the Fortran code.

The subroutine newsurf will be called from the MCTDH or Potfit program via the subroutine uvpoint (i.e. user V point) which is located on the file opfuncs/usersrf.F.

At the end of the subroutine uvpoint a call to the new surface routine must be added. For this in-comment

```fortran
elseif(ifunc .eq. 2) then
```

(near line 107 of file opfuncs/usersrf.F) and replace the line following this if-statement with

```fortran
call newsurf(gpoint,v)
```

The array gpoint contains the coordinates in the order specified in the Hamiltonian-section, via the mode-line and the |i&j&k| construct. (See Section 6.13).

The surface number hopilab, 2 in our example, then has to be defined. (Note, hopilab is called ifunc in some routines). This is done in subroutine udefsrf, which also is stored on source/opfuncs/usersrf.F. Add lines such as

```fortran
else if (label(1:ll) .eq. 'newsurf') then
    hopilab = 2
```

to this subroutine. If the surface depends of parameters or options, these options have to be read here. See subroutine defsrf on opfuncs/funcsrf.F for examples.

Finally, we want the program to write some information to the log file, when the multi-dimensional potential energy surface is used. To this end one has to edit the subroutine usersurfinfo on the file opfuncs/usersrf.F. One should briefly describe the surface and then name the coordinates. The string mlabel(j) contains the modelabel of the j-th coordinate of mysurf, as assigned in the Hamiltonian-Section. For example, the code added to usersurfinfo may read

```fortran
elseif( hopilab .eq. 2 ) then
    write(ilog,'(a)') 'newsurf. V(x,y,z) = beta-function'
    write(ilog,'(2a)') 'x : ',mlabel(1)
    write(ilog,'(2a)') 'y : ',mlabel(2)
    write(ilog,'(2a)') 'z : ',mlabel(2)
    jj=4 ! This is dimension+1
```

Finally, recompile. Include the new surface by running compile with the option -i newsurf. (See HTML documentation Installation and Compilation/Compiling the Programs)

Please be reminded again, that only real, multi-dimensional potential functions should be implemented on usersrf.F. For one-dimensional real functions please use user1d.F (or funcanld.F), for complex functions use funcanlz.F, and for grid based operators use funcgrd.f.

The new potential surface is selected in the Hamiltonian-section using the V (or any other not pre-defined) label and the Labels-section. Assuming that the Hamiltonian is given by $H = -1/2m \left( \partial_x^2 + \partial_y^2 + \partial_z^2 \right) + V(x,y,z)$, the Hamiltonian-section reads

```
HAMILTONIAN-SECTION
-------------------------------
modes | x | y | z
------------------------------
```
6.9 Incorporating natural potentials

With the aid of the Potfit program potential surfaces can be fitted to the product form 6.4. These fits are known as natural potential fits. Natural potentials are the method of choice to employ non-separable potential surfaces in an MCTDH calculation. How such a fit is generated will be discussed in Sec. 12.1. We only mention that the Potfit program requires the (non-separable) potential to be implemented in the operator library in exactly the same way as described in the previous section.

After having constructed a natural potential fit, it may be used in the MCTDH program by correspondingly defining the label \( V \) in the above example:

\[
\begin{align*}
  \text{LABELS-SECTION} \\
  V &= \text{natpot\{directory\}} \\
\end{align*}
\]

Here \textit{directory} denotes the path to the directory containing the \textit{natpot} file, which is created by the Potfit program. Replacing the path by the keyword \textit{name}, i.e. \textit{natpot\{name\}}, indicates that the \textit{natpot} file is in the name-directory, i.e. the directory the output is directed to.

Note that MCTDH uses the modelables to let the potential operate on the degrees of freedom (or on combined modes, i.e. MCTDH particles) in the correct order. It is hence
recommended to use an unnumbered Hamiltonian line (e.g. | V ) rather than a numbered one (e.g. |1&2&3 V ) when V refers to a natural potential. One may put the symbol | V in any column, and it may be convenient to place it in the first column. If a numbered Hamiltonian line is used, however, the numbers must be consistent with the model labels. Otherwise the program will stop. If one gives the keyword _ignore_ as argument to natpot{⋯} in the Labels-Section, then the model labels are ignored and the assignment of the modes or DOFs is done exclusively by using the numbers. As a final remark we note that the model labels of a natpot may be altered by running chnpot85.

There are some restrictions when using natural potentials (natpots). Natpots cannot be multiplied with another operator (except a constant). Hence a construct like | cos*V is not allowed. However, if the natpot does not operate on all DOFs or particles, it may be combined with operators acting on the remaining DOFs or particles. E.g. if the natpot V does not operate on the first DOF, the following Hamiltonian line is allowed

\[ \text{const} \mid dq^2 \mid V \]

This, however, does not hold if the other operator is also a natpot. I.e.

\[ \text{const} \mid V1 \mid V2 \]

is not a valid Hamiltonian line if V1 and V2 are both natpots. One cannot multiply natpots with each other.

### 6.10 Using complex absorbing potentials (CAPs)

Complex absorbing potentials (CAPs) can be employed to reduce the lengths of the primitive grids. CAPs are also useful for computing S-matrix elements in scattering processes. Please refer to Chaps. 4.7 and 8.6 of the review [1] for details.

The CAPs \(-iW\) that can be employed in the MCTDH program are one-dimensional and monomial, i.e. of the form

\[ -iW(x) = -i\eta |x - x_c|^b \Theta(\pm (x - x_c)). \]  

(6.5)

The parameters \(x_c\), \(\eta\), and \(b\) denote the starting point, strength, and order of the CAP, respectively. \(\Theta\) specifies Heaviside’s step function. When the positive sign is used, the CAP lies to the right of \(x_c\), otherwise it is located to the left of \(x_c\).

Let us assume that your system under investigation has three degrees of freedom, labeled \(x\), \(y\), and \(z\). To turn on CAPs for, say, the first two modes, add the lines

\[
1.0 \mid \text{cap1} \mid 1 \mid 1 \\
1.0 \mid 1 \mid \text{cap2} \mid 1
\]

to the HAMILTONIAN-SECTION. The labels \text{cap1} and \text{cap2} (or any other labels you have chosen) are then defined in the LABELS-SECTION:

```
LABELS-SECTION
\text{cap1} = \text{CAP} [ 5.0 \ 0.375 \ 3 \ +1] \\
\text{cap2} = \text{CAP} [ 1.0 \ 0.240 \ 2 \ -1] \\
end-labels-section
```

The parameters in square brackets are from left to right the starting point \(x_c\), strength \(\eta\) (both in a.u.), and order \(b\) of the CAP. The last number specifies whether the CAP lies to the right (+1, which is also the default), or left (−1) of \(x_c\). Note, the input [⋯] may consist of numbers, parameters or algebraic expressions containing numbers and parameters. Hence
6.10 Using complex absorbing potentials (CAPs)

\[ \text{cap1} = \text{CAP} \left[ 3.0 + x_0 \ 1.0 \times 10^{-3} \times \text{strength} \ 3 \ +1 \right] \]

is a valid statement, provided the parameters \( x_0 \) and \( \text{strength} \) are defined.

The described way of including CAPs has the disadvantage that the CAPs are hard-wired in the operator file. The MCTDH program therefore offers the opportunity to switch on CAPs from the input file, without any change of the HAMILTONIAN- or LABELS-SECTION in the operator file. To this end include the lines

\[
\begin{align*}
\text{alter-labels} \\
\text{cap_x} = \text{CAP} \left[ 5.0 \ 0.375 \ 3 \ +1 \right] \\
\text{cap_y} = \text{CAP} \left[ 1.0 \ 0.240 \ 2 \ -1 \right] \\
\text{end-alter-labels}
\end{align*}
\]

in the OPERATOR-SECTION of the input file. The special keyword \( \text{cap}_x \), where \( x \) stands for one of the mode labels, tells the program to add a CAP to the corresponding degree of freedom. See Section 6.4 for more details on the special meaning of the underscore (\( _{} \)) within a label.

The optimal values of the CAP parameters, i.e. the Cap length, CAP strength, and CAP order, need to be determined. It is our experience, that the optimal CAP order is 2 or 3 (the larger the energy range, \( E_{\text{max}}/E_{\text{min}} \), the larger the optimal CAP order). The Cap length should be as small as possible in order not to waste grid points. On the other hand, a short CAP requires a large CAP strength which in turn, produces unwanted CAP reflections. The program \text{reflex85} computes an estimate of the CAP transmission and CAP reflection. This estimate, derived in Ref. [26], is very accurate for a free particle. To determine the optimal CAP parameters, one needs to know the lowest and highest kinetic energy component for the CAP degree of freedom with which the particle enters CAP. These energies are sometimes difficult to estimate. If one has used FFT (or exponential DVR) for the CAP degree of freedom, the command \text{showd1d85 -a pop2 -y 0.01 fx} (\( x \) denotes the number of the degree of freedom of the CAP) displays the momentum distribution from which the desired energies can be calculated.

The program \text{reflex85} is most conveniently called through the shell script \text{plcap}. The parameters necessary for the calculation are given as options and arguments. The program prompts for missing input. To give an example, let us turn to NOCl. Type

\text{plcap -n 3 -m 16 -l 0.6 -e 0.3 0.1 2.0 ev}

This computes the reflection– and transmission–probability for a CAP order of 3, a (reduced) mass of 16 atomic mass units, a CAP length of 0.6 a.u., a CAP strength of 0.3 a.u. and for the energy interval 0.1 – 2.0 eV. A GNUPLOT window pops up which displays the reflection– and the absorption–probabilities and their sum. \text{plcap} then prompts you for new options. Type \(-h\) to see the list of options, or type \(-z 1.e-7\) to arrive at a more convenient scale. If one inputs \(-e ？\), the program will compute the optimal CAP strength for the given energy interval.

The distribution of kinetic energies of dissociating NOCl lies between 0.2 eV and 1.6 eV, thus the reflection– and absorption–probabilities are below \(10^{-4}\), which is a fairly good value. The precise values of the CAP parameters are usually not very critical, except when very low kinetic energies are present. Very low kinetic energies may appear when an internal excitation takes almost all of the available energy. The low kinetic energy contributions are more strongly reflected from the CAP and these reflections lead to artificial oscillatory structures in e.g. the reaction probability. Fig. 3 of Ref. [12] shows such an small artificial structure near 1.25 eV, i.e. close to the \( v = 2 \) threshold. As discussed in Ref. [12] this small artificial structure disappears when using a longer and weaker CAP.

6.11 Advanced topic: Altering a Hamiltonian from input file or command line

The concrete form of the Hamiltonian, i.e. the values of parameters defined in the PARAMETER-SECTION and the meaning of labels specified in the LABELS-SECTION of the operator file, can be overridden in different ways. Let us first consider the change of parameters.

One possibility is to write the parameters to be changed into a file. Example 6.3 shows an example for such a parameter file, again for the Henon-Heiles system. To read this file and use its settings insert the keyword

parfile = mypara/hh.par

into the OPERATOR-SECTION of your input file. Here it was assumed that the parameter file is named hh.par and resides in the directory mypara (relative to the path of the input file).

A second way is to include the alter-parameter and end-alter-parameter keywords in the OPERATOR-SECTION of the input file. All parameter definitions in-between replace the corresponding parameters in the PARAMETER-SECTION of the operator file. For instance, the lines

alter-parameter
lambda = 0.4
end-alter-parameter

in the OPERATOR-SECTION of the input file set the coupling parameter \( \lambda \) of the Henon-Heiles potential, Example 6.2, to 0.4 a.u. The format of the parameters is the same as in the PARAMETER-SECTION of the operator file.

The third method makes use of command line parameters. Starting a calculation employing the -p option, e.g.

mctdh85 -D new -p lambda 0.4 hh

would run a new calculation with the input file hh.inp, but with the coupling parameter \( \lambda \) twice as strong as defined in the operator file. The -D option means that the results are written this time to the name-directory new. There may be more than one parameter definition and the parameters may carry a unit. Thus

mctdh85 -p lambda 10.9,eV -p mass_Y 1.5 hh

is also a valid command.

The order of precedence of parameters defined from different sources is command line, input file, parameter file, operator file. Thus parameters in the operator file are the default values, which can be altered in a run in a variety of ways.

The labels in the LABELS-SECTION of the operator file may also be modified without altering the operator file. This is done using the alter-labels keyword. One example for this was presented before in Sec. 6.10, where complex absorbing potentials were added to the Hamiltonian. Another typical example is to switch between different implementations of a potential. Supposed the potential to be used is represented by the label V in the line

# Parameter file for the Henon-Heiles system
lambda = 0.4
end-parameter-file
# Uwe Manthe fit NOCl S1 surface

PARAMETER-SECTION
theta0 = 127.4, deg

c000 = 0.0384816 c001 = 0.0247875 c002 = 0.0270933 c003 = 0.00126791
c004 = 0.00541285 c005 = 0.0313629 c006 = 0.0172449
...

end-parameter-section

HAMILTONIAN-SECTION

<table>
<thead>
<tr>
<th>modes</th>
<th>rd</th>
<th>rv</th>
<th>theta</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1</td>
<td>v:NO</td>
<td>1</td>
</tr>
<tr>
<td>c000</td>
<td>bcrd`0*1qd</td>
<td>bcrv`0</td>
<td>bcw`0</td>
</tr>
<tr>
<td>c001</td>
<td>bcrd`0*1qd</td>
<td>bcrv`0</td>
<td>bcw`1</td>
</tr>
<tr>
<td>c002</td>
<td>bcrd`0*1qd</td>
<td>bcrv`0</td>
<td>bcw`2</td>
</tr>
<tr>
<td>c003</td>
<td>bcrd`0*1qd</td>
<td>bcrv`0</td>
<td>bcw`3</td>
</tr>
<tr>
<td>c004</td>
<td>bcrd`0*1qd</td>
<td>bcrv`0</td>
<td>bcw`4</td>
</tr>
<tr>
<td>c005</td>
<td>bcrd`0*1qd</td>
<td>bcrv`0</td>
<td>bcw`5</td>
</tr>
<tr>
<td>c006</td>
<td>bcrd`0*1qd</td>
<td>bcrv`0</td>
<td>bcw`6</td>
</tr>
<tr>
<td>c010</td>
<td>bcrd`1*1qd</td>
<td>bcrv`0</td>
<td>bcw`0</td>
</tr>
<tr>
<td>c011</td>
<td>bcrd`1*1qd</td>
<td>bcrv`0</td>
<td>bcw`1</td>
</tr>
</tbody>
</table>

....

end-hamiltonian-section

LABELS-SECTION

bcrd = exp[-1.5,4.315]
lqd = exp[-1.5,4.315]
bcrv = q[2.136]
bcw = expcos[-1.1,theta0]

end-labels-section
end-operator

**Example 6.4:** A surface file containing the MANTHE analytic fit to the NOCl S1 potential energy surface.

1.0 | V

of the HAMILTONIAN-SECTION and defined as

LABELS-SECTION

V = mysurf

end-labels-section

in the LABELS-SECTION of the operator file. Then you may add

alter-label
V = natpot{name}
end-label-parameter

to the OPERATOR-SECTION of your input file. The program then uses the natural potential fit stored in the name-directory rather than the potential surface labeled mysurf.

It is also possible to move some part of the Hamiltonian section, e.g. the potential, to a separate file, called surface file. Use of this is made in Example 6.1, where only the kinetic energy part is defined in the HAMILTONIAN-SECTION. The (separable) potential is stored in the surface file nocl1.srf, part of which is displayed in Example 6.4. (See
Setting up the Hamiltonian

For a complete listing and compare it with Eqs. (32,33) of Ref. [4]).

The surface is defined in the Hamiltonian by

```
LABELS-SECTION
V = srffile {noc11, directory}
end-labels-section
```

Here directory denotes the path to the directory containing the noc11.srf file. Replacing the directory by the keyword oppath indicates that the surface file is in the same directory as the .op file. The keyword default will make the program look for the .srf file in the default operator directory. Using again the alter-label keyword in the OPERATOR-SECTION of the input file, one may select a different potential with a minimum of effort.

Finally we note that is is possible to impose an energy cut-off on a non-separable potential energy surface by using the v keyword in the operator section of the input file. This is detailed in the HTML documentation.

### 6.12 Setting up Auxiliary Operators

In addition to the system Hamiltonian, other operators may be required, e.g. to generate eigenfunctions of a zero-order Hamiltonian for the initial wavepacket (see Sec. 7.6), or to calculate the time-evolution of an expectation value (either using the expect keyword in the RUN-SECTION, or the ANALYSE program EXPECT). Operators needed during a run must be included in the .op file. They are defined exactly as the system Hamiltonian, but are delimited by

```
HAMILTONIAN-SECTION_XXX
  .
  .
  .
end-hamiltonian-section
```

where XXX is a label to distinguish the operator. Operators to be used in any post-propagation analysis can also be set up in a separate .op file, i.e. one not containing the system Hamiltonian. The RUN-SECTION keyword genoper = S can then be used to set up the operators in the file S.op to the read-write file oper.S.

To check that these operators are used correctly it is necessary to understand the working of the program internal flag diag assigned to each operator of which the total operator is composed. If this flag is set to .true. then the operator is a unit operator and it will not be explicitly evaluated. This has an obvious advantage for the efficiency of the program. The program also uses these flags to determine which operator terms are separable, i.e. product terms in which all operators except for one are unit operators.

In some cases, however, unit operators must be explicitly evaluated. An example is when the matrix elements \( \langle \tilde{\varphi}_a \mid h \mid \varphi_b \rangle \) are required, where \( \{ \tilde{\varphi} \} \) and \( \{ \varphi \} \) are different basis sets, which happens when e.g. the operate keyword is used. For this reason there is a flag nodiag assigned to each operator, which when set to .true. turns off the use of the diag flag. This flag is set by the program, but can also be set by hand using nodiag or usediag as the very first keyword in the HAMILTONIAN-SECTION_XXX. How this flag is set is listed in the log and op.log files. Note that the system Hamiltonian and the operators used for eigenf, meigenf, expect, and pexpect must be of usediag type. For operate,
6.13 DOF, mode, and muld potentials

The potential functions from which the Hamiltonian is built are mostly one-dimensional. They are accessed through symbols like q, sin, or vh2. However, MCTDH can also use multi-dimensional potential functions. These are defined in `funcsrf.F` of `usersrf.F`. If the multi-dimensional potential operates on one particle (i.e., on one combined mode) it is a particle (or mode) operator. Otherwise it is a so-called muld-potential. The use of muld potentials makes the propagation slow and in general one will use potfit to generate a natural potential, which is a sum of products of DOF (or mode) potentials. However, there is no disadvantage in using (multi-dimensional) mode potentials. The MCTDH program will recognise a multi-dimensional potential as mode potential, if it operates on all DOFs of one mode, but on no other DOF. Assume that there is a symbol $f_{xy}$ which refers to a potential function $f_1(x,y)$. A Hamiltonian-Section may then read

```
HAMILTONIAN-SECTION
-----------------------------------------
modes | R | x | y | z | theta
-----------------------------------------
   .
const | 2 & 3 fxy | 5 cos
   .
-----------------------------------------
end-hamiltonian-section
```
or equivalently

```
HAMILTONIAN-SECTION
-----------------------------------------
modes | R | x | y | z | theta
-----------------------------------------
   .
const | 1 & fxy | 1 | cos
   .
-----------------------------------------
end-hamiltonian-section
```

If $x$ and $y$ are uncombined, then $f_{xy}$ will be treated as a muld potential, but when $x$ and $y$ are combined to form a MCTDH particle, then $f_{xy}$ will be treated as mode operator. Inspect the op.log file. After the operator terms are summed (if possible), they are listed under the heading

```
Hamiltonian Operator Terms [h, htdof, htmmode, htmmuld, htmtype, htsym, string]
No. f m md Typ Sym Term
```

A non-zero entry in the column $f$ or $m$ denotes that the operator term acts on the DOF $f$ or the particle $m$, respectively.

If however $x,y$, and $z$ would form one MCTDH particle, then $f_{xy}$ would not be recognised as a mode operator, because it does not act on all DOFs of the mode. One has to include $z$ as a dummy DOF.
Assume that there is another symbol \( fzx \), which refers to the function \( f_2(z,x) \). An inclusion of this function may look like

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{modes} & R & x & y & z & \theta \\
\hline
\text{const} & 4 & 2 & 3 & fzx & 5 \cos \\
\hline
\end{array}
\]

Note that the dummy variable(s) must be the last one(s). The symbol \( fzx \) is simply interpreted here as \( f_2(z,x,y) \) with no dependence on \( y \). Note that one can freely re-order the arguments of a multi-dimensional function when using a numbered Hamiltonian line.

The use of muld potentials is restricted as there must be no “holes” in the order of modes and of DOFs within a mode. I. e. a muld potential may operate on mode 2, 3, and 4 but must not operate on mode 2 and 4 only. In the latter case there is a hole (mode 3). Holes are only allowed at the beginning and the end of a muld potential. One may fill the holes by incorporating dummy variable as done above. However, this will make the application of a muld potential even more expensive. One rather should try to re-order the modes and DOFs to avoid the holes. In any case, it is advisable to transform a muld potential to a separable natpot by using `potfit` (see Section 12.1). How a natpot is incorporated is discussed in Section 6.9.
Finally an important note. It is not possible to re-order the arguments or to add dummy variables if the special multi-dimensional “function” readsrf is used. In this case the re-ordering and/or addition of dummy variables must be done on the readsrf data file.

6.14 Golden rules for writing operator files

General remarks
File-names, modelabels, labels and parameters are case sensitive! Hence most parts of the operator file are case sensitive (in contrast to the input file). All input is assumed to be in atomic units. In contrast to the input-file this holds also for times. One hence has to explicitly give the unit $\text{fs}$ when a parameter value is given in femto-seconds.

Parameter-Section  (See also Section 6.2)
Parameters are real numbers. When real a number appears in the Parameter-Section it should include a dot. E. g. one should use $2.0$ rather than $2$. Numbers in exponential format, e. g. $1.0d-2$ should be avoided, if the exponent is not large. Use $0.01$ in this case. The letter indicating the exponent must be a lower case $d$; a $D$, $e$, or $E$ will not work. One may perform simple arithmetic with the parameters (see Section 6.2). In particular one may exponentiate a parameter. The exponent, however, must be a number and cannot be another parameter. (Use the function $\text{EXP}$ for exponentiation). If the exponent is integer, write it as integer, e. g. write $\text{par}^3$ rather than $\text{par}^3.0$. Note that the exponent can be real and also negative, e. g. $\text{par}^{-0.5}$ is possible. The use of brackets is not allowed. Rather than writing
\[ \text{cent} = j*(j+1)/(2*\text{mass}) \]
one has to write
\[ \text{cent} = 0.5*j^2/\text{mass} + 0.5*j/\text{mass} . \]

The string which is used to specify a parameter may consist of upper or lower case letters, numbers, and the special characters

\[ . _ \ - \ @ \ $ \ % \ & \ ? \]

Note in particular that the colon (:) is not allowed to be part of a parameter name. It is recommended to choose names which start with a letter. Note that there is a pre-defined parameter $\text{PI}$ with obvious meaning.

There is a special parameter, called $\text{mass}_\text{modelabel}$ where $\text{modelabel}$ is a label which was assigned to one of the degrees of freedom in the Primitive-Basis-Section. This special parameter should be set to the (reduced) mass of the indicated degree of freedom. This parameter is used in connection with the $\text{KE}$ keyword and strange results may occur if $\text{KE}$ is used but $\text{mass}_\text{modelabel}$ is not set. Do not use this construct when the second part is not a valid modelabel. E. g. for the total mass do not use $\text{mass}_\text{tot}$ but rather use $\text{mass@tot}$ or $\text{mass.tot}$ or $\text{MassTot}$ instead.

Labels-Section  (See also Section 6.4)
One may use the same set of letters, numbers and special characters for labels, as are allowed for parameters. Again, the colon (:) is not allowed to be part of the name, although pre-defined labels (i.e. those listed in Appendix C) may contain a colon. Moreover, the underscore has a special meaning for labels. If a label has the structure $\text{label}_\text{modelabel}$ then the mctdh program will put the corresponding operator in that column of the Hamiltonian-Section which refers to $\text{modelabel}$. One must not put it explicitly there. Unit operators are assumed for all other degrees of freedom. This feature, which is often used to include CAPS,
excludes the general use of the underscore in a label. E. g. defining a label as \( \exp_{-1} \) may produce an error, because \( -1 \) may not be a modelabel.

Note that there must not be a parameter and a label which have the same name. E. g. \( q \) cannot be used as parameter because it is pre-defined as a label. The program checks that parameter and label names are disjoint.

**Hamiltonian-Section**  (See also Section 6.3)

Only simple labels may appear in a Hamiltonian-Section. Operators with arguments must be assigned to a simple label in the Labels-Section.

With the aid of the caret \(^\wedge\) one may apply a power to operators. The power may be integer or real and may carry a sign. This, however, works only for potential like operators. Inspect Appendix C to learn, which operators can be exponentiated. Note, that symbols like \( dq^2 \) or \( j^2 \) are operator labels of their own right, they do not denote that the second power of the operators \( dx \) or \( j \) is taken literally.

Time-dependent operators can easily be implemented. The time is simply treated as an additional DOF of the Hamiltonian-Section. The modelabel of this additional DOF must be Time. See the HTML-documentation ("Hamiltonian/Liouvilleian Documentation" and then "Time-dependent Operators") for further details. See also the operator file `noc1T.op` on `$MCTDH_DIR/operators`. 
Chapter 7

Generating the initial wavepacket

For a quantum dynamical calculation an initial wavepacket $\Psi(0)$ is required. This is done in the INIT_WF-SECTION of the input file. The initial wavepacket must have a particular form, depending on the method to be used. If the MCTDH scheme is employed, $\Psi(0)$ has to be represented as a multi-configurational Hartree product (i.e. a linear combination of products of single-particle functions), while in a numerically exact calculation it must be mapped onto a product of DVR grids. Usually, $\Psi(0)$ is a simple Hartree product, i.e. a product of one-dimensional functions (unless spherical harmonics are employed, which are two-dimensional). The MCTDH program offers a number of function types to be used for each factor in the product, i.e. each degree of freedom.

7.1 Building Gaussian functions as initial functions

A possible choice for the one-dimensional initial functions are Gaussian functions. These can be defined in two manners which differ only by the way the width is specified, namely either as

$$\varphi(x) = N e^{-1/4((x-x_0)/\Delta x)^2} e^{i p_0 (x-x_0)} \quad (7.1)$$

or as

$$\varphi(x) = N e^{-1/2 m \omega (x-x_0)^2} e^{i p_0 (x-x_0)}, \quad (7.2)$$

with corresponding keywords gauss and HO. Here $N$ is a normalisation constant, $x_0$ and $p_0$ are the centre and initial momentum, $\Delta x$ is the width, and $m$ and $\omega$ denote mass and frequency.

Suppose there are two degrees of freedom $X$ and $Y$, then the initial wavepacket may be defined by an INIT_WF-SECTION reading

```
INIT_WF-SECTION
build
  X  gauss  4.315  0.0  0.0794
  Y  HO  2.151  0.0  0.218,eV   13615.5
end-build
end-init_wf-section
```

The keywords build and end-build enclose the lines that specify how to build the initial wavefunction. The first and second number in each line denote $x_0$ and $p_0$, respectively. The next numbers are $\Delta x$ in the first case, and $\omega$ and $m$ in the second. As the example shows, one may add a unit to the parameters. Note that $\Delta x$ and $\omega$ may be complex.
plane waves may be generated by setting the frequency within the \text{HO} line to zero. E.g.

\begin{verbatim}
build
  X  HO  0.0  0.0  0.0
  Y  HO  0.0  2.5  0.0
end-build
\end{verbatim}

will generate a flat function for the X degree of freedom and a plane wave with momentum 2.5 au for the Y degree of freedom.

In a numerically exact calculation, the initial wavepacket is simply the product of the functions (7.1) or (7.2) for the degrees of freedom involved. In an MCTDH calculation, however, the program interprets each line in the \text{INIT\_WF-SECTION} as first single-particle function for that degree of freedom. Higher single-particle functions are then constructed by multiplying the preceding function by $x$ (so producing a series of powers of $x$), followed by Schmidt-orthogonalisation onto the lower functions. The set of all products of the functions of the included modes then defines the initial configurational space.

The initial wavefunction in an MCTDH calculation is then chosen as one of these configurations. The default is to use the product of the first single-particle functions of each mode, thus arriving at the same initial wavefunction as in a numerically exact calculation: the product of the functions (7.1) or (7.2). One may however also populate a different configuration, with the aid of the \text{pop} keyword, e.g.

\begin{verbatim}
INIT\_WF-SECTION
build
  X  gauss 4.315 0.0 0.0794  pop = 2
  Y  HO  2.151 0.0 0.218,eV  13615.5  pop = 3
end-build
end-init_wf-section
\end{verbatim}

The initial wavepacket is in this example the product of the second single-particle function in X and the third in Y.

### 7.2 Setting up Legendre functions as initial functions

An initial function frequently employed for angular modes is an associated Legendre function

$$
\phi_{l-m+1}(\theta) = \sqrt{\frac{2l+1}{2}} \frac{(l-m)!}{(l+m)!} P_l^m(\cos \theta),
$$

(7.3)

with $0 \leq m \leq l$. The parameter $m$ denotes the magnetic quantum number and is treated as a fixed parameter. $P_l^m$ is the (unnormalised) associated Legendre function (4.4).

The following example defines the initial wavepacket of a system with two degrees of freedom \text{rd} and \text{theta} as product of a Gaussian function in \text{rd} and a Legendre polynomial in \text{theta}:

\begin{verbatim}
INIT\_WF-SECTION
build
  rd  gauss  4.50d0  -8.76d0  0.18d0
  theta Leg  0  0  sym
end-build
end-init_wf-section
\end{verbatim}
7.3 Setting up extended Legendre functions as initial functions

The numbers after the keyword `Leg` denote $m$ and $l$, respectively. If the corresponding type of the primitive basis is not `Leg`, then $m$ must be zero; if it is `Leg`, then $m$ must coincide with the value in the PRIMITIVE-BASIS-SECTION.

In an MCTDH calculation the program uses the Legendre polynomial specified in the INIT_WF-SECTION to define not only the initial wavepacket but also the first single-particle function. Which higher single-particle functions are used depends on the last parameter, which can be `sym` or `nosym`. In the latter case all values of $l$ (both even and odd) are employed, in the former case only those Legendre functions having the same symmetry as $l$ (either even or odd) are taken.

7.3 Setting up extended Legendre functions as initial functions

When the extended Legendre DVR is used an initial associated Legendre function and the corresponding K-function (which is a Kronecker $\delta$) should be generated via the `KLeg` and `K` keywords.

```plaintext
INIT_WF-SECTION
build
............
  theta  KLeg 2  sym
  K       K 1
............
end-build
end-init_wf-section
```

The number after the keyword `KLeg` denotes the initial $\ell$, and the keyword `sym` accomplishes that for the generation of higher single-particle functions only every second one will be taken. I.e. there will be only even or only odd $\ell$'s depending on whether the initial $\ell$ is even or odd. The keyword `sym` may be replaced by `nosym` with obvious meaning. The number following `K` denotes the initial value of $K$. Note that the `KLeg` and `K` keywords of the INIT_WF-SECTION may also be used when the `PLeg` DVR is employed.

7.4 Generating spherical harmonics as initial functions

If spherical harmonics have been employed as primitive basis functions for a combined mode `alpha` and `beta`, a normalised spherical harmonic

$$Y_{jm}(\alpha, \beta) = \sqrt{\frac{2j+1}{4\pi}} \frac{(j-m)!}{(j+m)!} P^m_j(\cos \alpha) e^{im\beta} \quad (7.4)$$

is the appropriate initial function for that mode. Here $P^m_j$ denotes the associated Legendre function (4.4).

Spherical harmonics can be selected similarly to the primitive basis by the INIT_WF-SECTION

```plaintext
INIT_WF-SECTION
build
............
  alpha  sphfbr  0
  beta   phifbr  0
```
The two numbers specify $j$ and $m$. Of course, the initial spherical harmonic must be part of the primitive basis set.

In an MCTDH calculation, this again defines not only the initial wavepacket but also the first single-particle function. Higher single-particle functions are other spherical harmonics whose quantum numbers are as close as possible to the quantum numbers of the first one.

### 7.5 Generating Wigner functions as initial functions

For rotational motion of polyatomic molecules in three dimensions, Wigner-D functions can be used for the initial wave function. These are defined as:

$$D^j_{m,k}(\alpha, \beta, \gamma) = e^{-ima} d^j_{m,k}(\beta) e^{-ik\gamma}$$  \hspace{1cm} (7.6)

$$d^j_{m,k}(\beta) = \langle j, m | e^{-i\beta J_y} | j, k \rangle$$  \hspace{1cm} (7.7)

where $D^j_{m,k}(\alpha, \beta, \gamma)$ is the normalized Wigner-D function and $d^j_{m,k}(\beta)$ is the Wigner (small)-$d$ function, and $-j \leq m, k \leq j$.

A Wigner-D function can be generated as an initial wave function by specifying the `wigner` keyword, followed by two $K$ lines, as in the example below:

```plaintext
INIT_WF-SECTION
build
-----------------------------
# mode type q.n.
-----------------------------
... beta wigner 5 nosym excite=mkj print # j of initial wigner
gamma k -3 7 1 # initial-k, k-range, k-step size
alpha k 1 -2 2 1 # initial-m, m-range, m-step size
...
-----------------------------
end-build
END-INIT_WF-SECTION
```

The Wigner-(big)-D function is generated in the above example as the product of a Wigner (small)-$d$ function for the $\beta$ Euler angle and a Kronecker-$\delta$ for the associated momentum quantum number of each of the $\alpha$ and $\gamma$ angles. The first number following `wigner` denotes the $j$-value of the initial wavefunction; initial values of the $k$ and $m$ quantum numbers along with their ranges and step sizes are given in the first and second $K$ lines, respectively. The corresponding DVR for `wigner` initial wavefunctions must be `wigner` for the first degree of freedom in the combined mode; either `exp` or `K` are allowed DVR/FBR types for the second and third degrees of freedom. The degrees of freedom are assumed to be given in
the order $|J, K, M\rangle$. The \texttt{excite} keyword can be used to choose between two different schemes for generating unoccupied single-particle functions: \texttt{excite=mkj} preferentially excites $m$-states, then $k$, then $j$, while \texttt{excite=kmj} preferentially excites $k$, then $m$, then $j$-states.

### 7.6 Generating eigenfunctions of a one-dimensional Hamiltonian

It may be useful to start a calculation with the wavepacket in a particular eigenstate of a zero-th order Hamiltonian. This occurs, for example, in an atom-diatom scattering calculation when the diatom starts in a particular vibrational eigenstate. To do this, one must first define the zero-th order operator. This is done by including a \texttt{HAMILTONIAN-SECTION\_OPER} section in the operator file (see Sec. 6.12) to define an operator labelled \texttt{OPER} (any other string except \texttt{SYSTEM} can be chosen for this name).

As an example, a one dimensional $\text{H}_2$ Hamiltonian operator can be defined by adding the section

\begin{verbatim}
HAMILTONIAN-SECTION_H2
usediag
-------------------------------------------
modes | rd | rv | theta
-------------------------------------------
  1.0  | 1  | KE | 1
  1.0  | 1  | vh2 | 1
-------------------------------------------
end-hamiltonian-section
\end{verbatim}

to the operator file. In fact, the simpler input

\begin{verbatim}
HAMILTONIAN-SECTION_H2
------------------
modes | rv
------------------
  1.0  | KE
  1.0  | vh2
------------------
end-hamiltonian-section
\end{verbatim}

works as well, because \texttt{usediag} is default for \texttt{eigenf}, and for DOF’s which are not listed a unit operator is assumed by default.

The desired functions are then generated by using the \texttt{eigenf} keyword in \texttt{build} block of the \texttt{INIT\_WF\_SECTION}, e.g.

\begin{verbatim}
INIT_WF-SECTION
build
  rd  gauss  4.50d0  -8.76d0  0.18d0
  rv  eigenf  H2  pop=2
  theta  leg  jbf  sl0  sym
end-build
end-init_wf-section
\end{verbatim}

generates a three dimensional wavepacket with a Gaussian along mode \texttt{rd} and the second eigenfunction (i.e. the first excited state) of the operator $\text{H}_2$ for \texttt{rv}. (NB. \texttt{pop=1} is default and may be dropped). For the theta degree of freedom an associated Legendre function is
Generating the initial wavepacket

The associated Legendre function is specified by the value of the parameters \( jbf \) and \( s10 \).

If the \texttt{veigen} keyword has been included in the RUN-SECTION, then the eigenfunctions and eigenvalues are written to the file \texttt{veigen}. In this way this procedure can be used to numerically exactly diagonalise a one-dimensional operator. If the \texttt{veigen} keyword is not given, the eigenvalues are still written to the log file.

NB. This is only possible if the primitive basis for the degree of freedom is a DVR basis (i.e. not an FFT) as the program generates the eigenfunctions by diagonalising the operator represented as a real matrix.

7.7 Reading the initial wavepacket from file

Instead of building a new initial wavepacket, one may also read a wavefunction that has been created in a previous calculation from the \texttt{restart} file. This is done by the \texttt{file} keyword in the INIT_WF-SECTION:

```
INIT_WF-SECTION
file = oldrun, orthopsi
end-init_wf-section
```

Here \texttt{oldrun} is the path of the directory where the \texttt{restart} file is stored. If no path is specified, the \texttt{restart} file is searched for in the name-directory. The second (optional) parameter, which can be \texttt{orthopsi} (the default) or \texttt{noorthopsi}, specifies whether or not the single-particle functions are Schmidt-orthogonalised after being read.

The primitive basis must be defined in the PRIMITIVE-BASIS-SECTION identically to the one of the previous calculation from which the initial wavepacket is being read. This can be ensured by reading the definition of the primitive basis of the previous run from file using the \texttt{readdvr} keyword in the RUN-SECTION, rather than defining the primitive basis in a PRIMITIVE-BASIS-SECTION. The number of single-particle functions, however, may differ.

Since version 8.3.10 there is also a \texttt{Read-Inwf} ... \texttt{end-read-inwf} block. In contrast to the simple \texttt{file} keyword, this allows to distribute the SPFs and the blocks of the A-vector freely among the electronic states. In particular, the current system and the wavefunction read in do no longer need to have the same number of electronic states. Example:

```
INIT_WF-SECTION
Read-Inwf
   file = gs
   SPF 1 -> 1,2,3
   A   1 -> 2
end-read-inwf
end-init_wf-section
```

Here, the file which is read in, \texttt{gs/restart}, has only one electronic state. Its SPFs are copied to all the three states of the current system and its A-vector is copied to state 2. The A-vector blocks for state 1 or 3 are hence zero. See the HTML documentation for more information.
7.8 Advanced topic: Diagonalising a multi-dimensional operator to create multi-dimensional SPFs

With the \texttt{meigenf} feature it is possible to diagonalise one– or multi–dimensional hermitian Hamiltonians to create one– or multi–dimensional SPFs. As \texttt{meigenf} uses the Lanczos algorithm (with full re-orthogonalisation) for diagonalisation, it needs some initial guess for the SPF. Hence the keyword \texttt{meigenf} must not be given in a build-block, it may come after a build block. However, \texttt{meigenf} can also alter the SPFs of a wavefunction which is read from file.

Example: \texttt{meigenf = 3,oper,0}

Here, \texttt{meigenf} will diagonalise the operator \texttt{oper}, which must be defined in a Hamiltonian-Section (similar to \texttt{eigenf}). The eigenfunctions of \texttt{oper} will then replace the SPFs of the third mode (first argument). The third argument, 0, finally indicates that the ground state is taken as the first SPF. (In contrast to \texttt{eigenf}, \texttt{meigenf} counts the eigenfunctions from zero). The Lanczos iteration is stopped, when the selected state (here the ground state) is converged. Adding the argument \texttt{full} forces \texttt{meigenf} to perform as many iterations as there are grid points, leading to a numerically exact full diagonalisation of the operator. (This is not recommended if the particular mode under discussion is represented by many grid points, more than 300 say). With an additional integer argument one may limit the number of iterations. Finally, if the integer argument for the selected eigenstate is replaced by the argument \texttt{follow}, then that eigenfunction, which has the largest overlap with the initial function, will be taken as first SPF.

Example: \texttt{meigenf = 3,oper,follow,full,select,write,125}

In this example the maximum number of arguments is given. See the HTML documentation for explanation and more information.

7.9 Advanced topic: Generating an initial wavepacket using an operator

It is also possible to first generate an initial wavepacket, and then to apply an operator to this wavepacket before starting the propagation. This is required, e.g., when the initial wavepacket to be generated is the dipole operator acting on a ground state wavefunction.

To do this, a wavepacket must be build or read in as described in this section above. The operator must also be defined in the operator file using a \texttt{HAMILTONIAN-SECTION OPER} section and setting \texttt{nodiag} (see Sec. 6.12). This sets up an operator labelled OPER (any other string except \texttt{SYSTEM} can be chosen for this name). Once this has been done, adding the keyword \texttt{operate=OPER} to the \texttt{INIT WF-SECTION} generates a wavepacket by applying this operator to the initial packet.

For a numerically exact wavefunction this procedure is simple. For an MCTDH wavefunction however the optimal single-particle functions for the final wavepacket may be different from those of the initial wavepacket. To optimise the basis functions for the new wavefunction, an iterative procedure is used. Details of the iterations are output in the log file.
### 7.10 Advanced topic: Creating a set of initial wavepackets

Instead of propagating only a single wavepacket, one may also define a set of $P$ initial wavepackets $\Psi_1, \ldots, \Psi_P$, which are then propagated simultaneously. This is called a multipacket calculation.

For example, to propagate $P = 2$ wavefunctions with coordinates $X$ and $Y$, one first has to add the line

```
packets = 2
```

to the SPF-BASIS-SECTION in order to specify the number $P$ of packets. The definition for the initial wavefunction has to be given for each wavepacket, e.g.

```
INIT_WF-SECTION
build
  X  gauss  4.315  0.0  0.0794  pack = 1
  Y  gauss  3.2    0.0  0.053   pack = 1
  X  HO     2.151  0.0  0.218,eV 13615.5 pack = 2
  Y  gauss  3.840  0.0  0.1378  pack = 2
end-build
end-init_wf-section
```

The `pack` keyword defines the packet to which an input line belongs. It is possible to specify more initial packets in this section than given by the `packets` argument. All data with `pack > packets` will then be ignored.

Note that the `auto` file now contains the cross-correlation matrix

$$c_{\alpha\beta}(2t) = \langle \Psi_{\alpha}^*(t) | \Psi_{\beta}(t) \rangle, \quad \alpha, \beta = 1, \ldots, P,$$

(7.8) rather than the auto-correlation function. See the HTML documentation for the exact format of the auto file.

### 7.11 Advanced topic: Setting up (a)diabatically corrected initial wavepackets

The flux-analysis and similarly `twprob` require the knowledge of the energy distribution of the (initial) wavepacket. This energy distribution is written to the `enerd` file, if the keyword `correction` is given in the `INIT_WF-SECTION`. (NB: The file `enerd` is called `adwkb` in older versions.) The keyword `correction` requires an argument. If `correction = edstr` is given, it is assumed that the wavepacket is located far outside such that the interaction potential can safely be neglected. The energy distribution is then given by Eq.(140) of the MCTDH review [1], i.e. essentially by a fourier transform of the single-particle function of the translational degree of freedom. It is assumed that the translational degree of freedom is the dof number 1, i.e. is the first item in the `PRIMITIVE_BASIS-SECTION`. Otherwise the keyword `trans = I1 (,I2)` must be given, where $I1$ denotes the number of the translational dof and (optional) $I2$ its electronic state.

The influence of the interaction potential is (partly) corrected for when giving `correction = dia`. The energy distribution is now evaluated as the overlap of the translational single-particle function with a distorted wave (rather than a plane wave). The
distorted wave is the solution of a 1D Schrödinger equation employing the translational mean field as interaction. (See the MCTDH review [1] Chapter 7.2 for details). The distorted wave is no longer calculated through the WKB approximation, but evaluated numerically using the Numerov method.

The quality of the energy distribution may be further improved by replacing the argument dia by ad. An adiabatic correction is now performed which modifies the single-particle functions of the internal degrees of freedom. This improves the mean-field and in turn the distorted wave. Adiabatic correction, however, is presently only implemented for the H+H\textsubscript{2} system and its isotopic variants.

Remarks:

- The translational degree of freedom must not be combined with other dof’s.

- There are special routines for the H+H\textsubscript{2} system (and its isotopic variants). These are used when the argument hh2 is additionally given with the keyword correction, e.g. correction = hh2, ad.

- Presently, the adiabatic correction works only in combination with the hh2 argument.
Chapter 8

Choosing an integration scheme

In a propagation or relaxation calculation an ordinary differential equation has to be solved. This can be accomplished by different integration methods. The first two sections, 8.1 and 8.2, are dealing with integration techniques developed especially for the MCTDH method. Section 8.3 describes general integrators, which can be used in both MCTDH and numerically exact calculations.

8.1 Using the VMF integration scheme in an MCTDH calculation

In an MCTDH calculation one possible integration method is the variable mean-field, or VMF scheme, which is described in Sec. 5.1 of Ref. [1] and in Ref. [27]. As the name implies, in the VMF scheme the mean-fields are determined in each integration step. The VMF scheme is the default in an MCTDH calculation.

As can be seen from Tab. 8.1, the equations of motion in the VMF scheme can be solved with an ABM (the default), BS or RK\(x\) integrator. ABM stands for Adams-Bashforth-Moulton predictor-corrector method, BS for Bulirsch-Stoer extrapolation scheme and RK\(x\) for a Runge-Kutta integrator or fixed order \(x\), where \(x = 5\) and \(x = 8\) are available. We recommend the use of the ABM method because it is generally more efficient.

To choose a VMF calculation employing the ABM integrator, the INTEGRATOR-SECTION in the input file should read, e.g.,

\[
\text{INTEGRATOR-SECTION} \\
\text{VMF} \\
\text{ABM} = 6, \ 1.0d-7, \ 0.01d0 \\
\text{end-integrator-section}
\]

The parameters after the \text{ABM} keyword are explained in Sec. 8.3. When the BS integrator is desired, a possible INTEGRATOR-SECTION is

\[
\text{INTEGRATOR-SECTION} \\
\text{BS} = 8, \ 1.0d-6 \\
\text{end-integrator-section}
\]

Here we have omitted the \text{VMF} keyword since it is the default.

Note that one may also not define at all the INTEGRATOR-SECTION. This is equivalent to specifying the \text{VMF} and \text{ABM} keywords, together with some default parameters for the ABM integrator that can be found in the HTML documentation.
8.2 Using the CMF integration scheme in an MCTDH calculation

In many cases an MCTDH calculation is more efficient if the VMF scheme is replaced by the constant mean-field, or CMF scheme. In the CMF scheme the numerical effort is reduced by holding the mean-fields, density matrices, and Hamiltonian matrix elements constant for some time, rather than evaluating them in each integration step. Note that the CMF scheme does (presently) not work in combination with the CDVR approximation. The CMF scheme is detailed in Sec. 5.2 of Ref. [1] and in Ref. [27].

Table 8.1 displays the integrators being compatible with the CMF method. Since the MCTDH coefficients (i.e. the $A$-vector) and the single-particle functions (i.e. the $\varphi$-vector) are propagated separately in the CMF scheme, different integrators can be chosen for each of them. This is indicated by appending /A or /spf to the ABM, BS, RK5, RK8 or SIL keyword. The default is SIL/A and BS/spf, which is in general the most efficient combination.

An example for the INTEGRATOR-SECTION in the input file is

```
INTEGRATOR-SECTION
  CMF = 0.5d0, 1.0d-6
  SIL/A = 15, 1.0d-7
  BS/spf = 9, 1.0d-7
end-integrator-section
```

This starts a CMF calculation with an initial stepsize of 0.5 fs and an error tolerance of $10^{-6}$. The parameters for the SIL and BS integrator are described in Sec. 8.3. If the same integrator (e.g. ABM) is to be used for the MCTDH coefficients and the single-particle functions, the shortcut /all can be appended to the integrator keyword:

```
INTEGRATOR-SECTION
  CMF = 1.0d0, 1.0d-5
  ABM/all = 5, 1.0d-4, 0.05d0
end-integrator-section
```

Note, however, that the ABM integrator typically will not give you the optimal performance of the CMF scheme. As a final example, the CMF scheme may also be selected by

```
INTEGRATOR-SECTION
  CMF
end-integrator-section
```

The program then uses default integrators and parameters, which are compiled in the HTML documentation.
In the above examples, two (optional) parameters are used to concretise the CMF calculation. The first one is the initial stepsize (in fs). A good guess for the initial stepsize is to use the output interval $t_{\text{out}}$ specified in the RUN-SECTION. Whether the initial stepsize was chosen reasonably can be checked by looking at the update file, which will be generated in a calculation if the `update` keyword in the RUN-SECTION is set. The update file indicates whether repetition steps were necessary in the beginning of the propagation. If so, one should use a smaller initial stepsize in the following calculations.

The second parameter defines the CMF error tolerance, which controls the stepsizes during the propagation. Typical values lie between $10^{-4}$ (very low accuracy) and $10^{-8}$ (very high accuracy). For many applications an error tolerance of $10^{-5}$ or $10^{-6}$ will be sufficient.

The convergence of a calculation with respect to the CMF error can be checked by comparing the results of two calculations performed with different error tolerances.

We finally note that it is possible to perform a CMF calculation with fixed or variable stepsizes. To choose among the possible options use the keywords `CMF/var`, `CMF/varphi`, `CMF/vara` or `CMF/fix`, respectively. With the extension `var` the stepsize becomes variable and is controlled by both the single particle functions and the A-vector. As `var` is default, `CMF/var` is identical to `CMF`. Using the extension `varphi` or `vara`, the stepsizes are controlled only by the single particle functions or only the A-vector, respectively. Finally, the extension `fix` enforces the use of a fixed stepsize. To discriminate these CMF stepsizes from the integrator step sizes, the former are often called update times.

### 8.3 Description of the available integrators

The integrators that are available are an Adams-Bashforth-Moulton (ABM) predictor-corrector method with fixed order and variable stepsize, a Bulirsch-Stoer (BS) extrapolation scheme with polynomial extrapolation and variable order and stepsize, two Runge-Kutta (RK5/8) integrators with adaptive stepsize and fixed order (5 or 8, respectively), and a short iterative Lanczos (SIL) algorithm with variable order and stepsize. Note that the (hermitian) SIL integrator is automatically replaced by a complex SIL integrator, also known as Lanczos-Arnoldi integrator, if the Hamiltonian is complex. More precisely, the Lanczos-Arnoldi routine is automatically chosen, if there is a complex potential (e.g. a CAP) in one of the separable parts of the Hamiltonian. However, the Hamiltonian may be non-hermitian for various other reasons. In these case one has to replace the `SIL` keyword by `CSIL`, which enforces the use of the Lanczos-Arnoldi integrator. Note that the integration will be incorrect, if the (hermitian) SIL integrator is used for a non-hermitian Hamiltonian! Which integrator has been used is protocoled in the log-file.

Each integrator is associated with up to three parameters. In case of the ABM, BS and SIL integrators the first one is the integration order and the second one the error tolerance, while the last one depends on the integrator. Typical error tolerances range from $10^{-3}$ or $10^{-4}$ (low accuracy) over $10^{-5}$ or $10^{-6}$ (normal accuracy) to $10^{-7}$ or $10^{-8}$ (high accuracy). For the RK5 and RK8 integrators (where the order is fixed), the first parameter specifies the error tolerance and the second one the initial stepsize. For all integrators it is in general not useful to work with an error tolerance less accurate than $10^{-5}$.

When performing a calculation, one should first select the desired error tolerance. The second step is to define the integration order. The meaning of this parameter is slightly different for the three integrators which provide it. For the ABM integrator, which runs with a fixed order, the order-parameter in the INTEGRATOR-SECTION is the true integration or-
Table 8.2: Optimal orders for the ABM and BS integrators in dependence of the error tolerance. The optimal ABM order was found empirically and might differ slightly in other cases. The values for the BS integrator, on the other hand, can be proved to be the optimal orders. (What is called “optimal BS order” in this guide is actually the maximum number of extrapolations.)

<table>
<thead>
<tr>
<th>Error tolerance</th>
<th>Optimal ABM order</th>
<th>Optimal BS order</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-3}$</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>$10^{-7}$</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

For the BS and SIL integrators, which continuously adapt their integration order during a run, the order-parameter denotes the maximum number of extrapolations and the maximum integration order, respectively. What the order-parameter hence defines is actually the memory being allocated, as all three integrators have in common that with each increase of the order-parameter by one, one additional wavefunction vector must be stored.

The order-parameter of the ABM and BS integrators should be chosen according to Tab. 8.2. Larger values do not increase the efficiency but only enlarge the memory requirements. (In the case of the ABM integrator a larger value in fact decreases the efficiency.) Smaller values for the order-parameter lead to longer CPU times. They might however be used if memory must be saved.

The optimal order-parameter of the SIL integrator unfortunately cannot be predicted but has to be found out empirically for each system. Typical values range from 6 to 16. After a calculation the largest order the SIL integrator has used is given in the log file. If this value is smaller than the order-parameter, you should decrease the order-parameter accordingly, to avoid the waste of memory in future calculations. If the largest order equals the order-parameter, this indicates that the efficiency might become higher if a larger order-parameter is chosen, so increase the order-parameter for optimal performance. When memory-intensive systems are investigated, it again might become necessary to use a smaller than optimal order-parameter, at the price of a longer computation time.

If the ABM, BS or RKx integrator is employed, the last parameter to be specified is the initial stepsize (in fs). In the case of the BS integrator, the output interval $t_{out}$ defined in the RUN-SECTION is normally a good choice. For the ABM integrator, the initial stepsize should in general be by a few orders of magnitude smaller than the output interval. This is because the ABM integrator — although being a multi-step method — has to be started as a one-step method, i.e., with an order of two, since initially the wavefunction is given for a single point of time only. In case of the RKx integrators, the initial stepsize can also be omitted or set to zero. The integrator then tries to guess a suitable value for the initial stepsize by employing a single explicit Euler step and estimating the second derivative of the solution. (However, in our experience this guess is often too conservative.) Whether the initial ABM, BS or RKx stepsize was chosen reasonably can be decided with the aid of the steps file, which is generated when the steps keyword in the RUN-SECTION is set. From this ASCII file it can be seen how large the first (successful) step actually was. This value may then be used as initial stepsize in future calculations.

For the complex SIL method, two different error estimates, called standard and improved estimate, are implemented, which can be specified by the third parameter. The standard error
criterion is based on the product of the sub-diagonal elements of the Lanczos matrix. The improved one uses the norm of the difference between the wave functions propagated with two consecutive orders. The improved estimate requires slightly more computation time but is more reliable when the stepsize is large. For details we refer the reader to Refs. [1,27]. The estimates can be activated by the keywords standard or novel, respectively. The former is the default.

8.4 Fine-tuning the Equations of Motion and the Integration Scheme.

There are a number of keywords that can be added to the INTEGRATOR-SECTION that change the form of the equations of motion, or change the way the integration is performed. Examples of these are given in this section.

8.4.1 Advanced topic: Propagating in natural or interaction picture orbitals

Instead of the standard single-particle functions one may employ natural or interaction picture orbitals. Natural orbitals are those single-particle functions that diagonalise the MCTDH density matrices. Interaction picture orbitals are obtained by moving from the Schrödinger to the interaction picture. For details see Secs. 3.3 and 3.4 of the review [1]. In normal use natural orbitals have no advantages over normal single-particle functions: they span the same space, and may even force the integrator to take smaller steps. The interaction picture may allow the equations of motion to be integrated more efficiently than the standard VMF scheme. This is especially true if an operator has a large separable part. It is however usually less efficient than the CMF scheme.

The type of orbitals to be used is selected in the INTEGRATOR-SECTION of the input file, since the orbital type affects the form of the equations of motion to be integrated. Place the keyword natorb or interpic into the INTEGRATOR-SECTION, i.e.

INTEGRATOR-SECTION
 .
 .
  natorb
end-integrator-section

or

INTEGRATOR-SECTION
 .
 .
  interpic
end-integrator-section

in order to move from standard to natural or interaction picture orbitals, respectively.

8.4.2 Suitable integrator settings for improved relaxation

The integrator settings for improved relaxation are somewhat different from those for propagation. Improved relaxation requires a CMF/fix or CMF/varphi integration scheme. The
best is simply to use CMF, this defaults to CMF/var for propagation runs and to CMF/varphi for improved relaxation. Improved relaxation furthermore requires a Davidson "integrator" (actually a diagonalizer), i.e. the keyword DAV, rDAV, rrDAV, or cDAV. A typical setting might read:

```
INTEGRATOR-SECTION
   CMF = 1.0, 3.0d-3
   RK8/spf = 1.0d-9
   rrDAV/A = 200, 1.0d-8
   natorb
      eps_inv=1.0d-10
end-integrator-section
```

Note that the CMF-accuracy is rather low, whereas the accuracy of the integrators is rather high. Note also that the parameter for regularizing the density matrices, \( \text{eps}\_\text{inv} \), is also set to a low value (its default value is \( 10^{-8} \)). This is because the lowest natural SPF populations are in the range \( 10^{-6} \ldots 10^{-10} \) for improved relaxation runs, whereas they are typically in the range \( 10^{-3} \ldots 10^{-6} \) for propagation runs.

The RK8 integrator was found to perform best for SPF relaxation. If a high accuracy of the results is not required, one may set the RK8 accuracy to \( 1.0d-8 \) and \( \text{eps}\_\text{inv}=1.0d-9 \), or even remove the \( \text{eps}\_\text{inv} \) line.

As orbital-type natorb was chosen in this example. The default for improved relaxation is energyorb (may be abbreviated to enorb). Energy orbitals make the Hamiltonian matrix more diagonal dominant than other orbital choices, which accelerates the convergence of the Davidson diagonalizer. However, the computation of the energy orbitals is a bit costly and we noticed that, in particular when a preconditioner is used, it is often more efficient to use natural orbitals. Standard orbitals, stdorb, which are default for propagation, can also be used. Note that the use of energy orbitals requires that the keyword orben is set in the Run-Section.

### 8.4.3 Advanced topic: Evaluating potentials using the TDDVR or CDVR method

The time-dependent DVR (TDDVR) and Correlation DVR (CDVR) methods offer the possibility of employing non-separable potentials within the MCTDH scheme, without the strong increase of computational labour that arises when such potentials are evaluated directly. The disadvantage of these procedures is their introduction of an additional, unpredictable error in the calculation. For a discussion of the TDDVR and CDVR methods see Ref. [1].

The TDDVR or CDVR method can be used by simply inserting either the keyword

```
TDDVR
```

or the keyword

```
CDVR
```

into the INTEGRATOR-SECTION of the input file. Presently, both TDDVR and CDVR work only in combination with the VMF integration scheme.
Chapter 9

Treating non-adiabatic systems

To treat a non-adiabatic system, i.e. a system which involves a manifold of coupled potential energy surfaces, the Hamiltonian operator has to be set up appropriately. Furthermore, the primitive and the single-particle basis, as well as the initial wavepacket, have to be defined in a special way when the system is non-adiabatic.

9.1 Setting up the Hamiltonian for a non-adiabatic system

The Hamiltonian of a non-adiabatic system with \( \sigma \) electronic states can be written as \((\sigma \times \sigma)\)-matrix

\[
H = \begin{pmatrix}
H_{11} & \cdots & H_{1\sigma} \\
\vdots & \ddots & \vdots \\
H_{\sigma 1} & \cdots & H_{\sigma \sigma}
\end{pmatrix},
\]

(9.1)

For the sake of simplicity we assume in the following that only two electronic states have to be accounted for, i.e. \( \sigma = 2 \). The generalisation to larger numbers of states is straightforward. To implement the Hamiltonian matrix, which is now two-dimensional, into the MCTDH program, it must have the product form required by the MCTDH method:

\[
H = \sum_{k=1}^{f} c_k \left\{ h_k^{(1)} \cdots h_k^{(f)} \begin{pmatrix}
\gamma_{11}^{(k)} & \gamma_{12}^{(k)} \\
\gamma_{21}^{(k)} & \gamma_{22}^{(k)}
\end{pmatrix}
\right\},
\]

(9.2)

where \( f \) denotes the number of molecular degrees of freedom and \( h_k^{(\kappa)} \) is a one-dimensional operator acting exclusively on the \( \kappa \)th degree of freedom. It is no restriction to assume that the elements of the \( \gamma \)-matrices are only zero and one, i.e. \( \gamma_{ij}^{(k)} \in \{0, 1\} \).

The program knows a number of built-in symbolic expressions that can be used to define the \( \gamma \)-matrices in the Hamiltonian section of the operator file. These symbols are compiled in Tab. 9.1. For instance, the symbol \( S1&1 \) specifies the symmetric matrix that couples states 1 and 1, while \( Z1&2 \) stands for the unsymmetric matrix that couples initial state 2 with final state 1. Note that the symbol \( S1&2 \) implies that initial state 2 couples with final state 1 and vice versa, because the corresponding matrix is symmetric.
9.2 Defining the primitive basis for a non-adiabatic system

Table 9.1: The built-in symbolic expressions that can be used to define the couplings of a non-adiabatic Hamiltonian. (Two electronic states are assumed.)

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Symbol</th>
<th>Matrix</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 0)</td>
<td>1</td>
<td>(0 1)</td>
<td>S1&amp;2 (or S2&amp;1)</td>
</tr>
<tr>
<td>(0 1)</td>
<td>S1&amp;1</td>
<td>(0 1)</td>
<td>Z1&amp;2</td>
</tr>
<tr>
<td>(0 0)</td>
<td>S2&amp;2</td>
<td>(0 0)</td>
<td>Z2&amp;1</td>
</tr>
</tbody>
</table>

As an example, the operator file for the 4-mode 2-state model of the pyrazine molecule is shown in Example 9.1. The Hamiltonian for this system reads

$$H = \sum_i \omega_i (\Omega_i^2 + Q_i^2) \left( \begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right) + \left( \begin{array}{cc} -\Delta & 0 \\ 0 & \Delta \end{array} \right) + \sum_{i \neq 10a} \left( \begin{array}{cc} \kappa_i^{(1)} & 0 \\ 0 & \kappa_i^{(2)} \end{array} \right) Q_i + \left( \begin{array}{cc} 0 & \lambda \\ \lambda & 0 \end{array} \right) Q_{10a}, \tag{9.3}$$

with $i = 10a, 6a, 1, 9a$. The numerical parameters are defined in the PARAMETER-SECTION, e.g. $\omega_{10a}$ corresponds to $\omega_{10a}$ and $\kappa_{6a1}$ to $\kappa_{6a1}^{(1)}$. This Hamiltonian is represented in the operator file by the HAMILTONIAN-SECTION (see Example 9.1). The modelabel $\text{el}$ labels the electronic states. For more details of the pyrazine calculations see Ref. [9].

9.2 Defining the primitive basis for a non-adiabatic system

For the treatment of a non-adiabatic system not only the operator but also the input file has to be set up appropriately. One modification concerns the PRIMITIVE-BASIS-SECTION, where an electronic basis has to be specified for the mode labelled $\text{el}$ in the operator file:

PRIMITIVE-BASIS-SECTION

```
  \begin{verbatim}
  v10a  HO  22  0.0  1.0  1.0
  v6a   HO  32  0.0  1.0  1.0
  v1    HO  21  0.0  1.0  1.0
  v9a   HO  12  0.0  1.0  1.0
  \text{el}  \text{el}  2
  \end{verbatim}
end-primitive-basis-section
```

The primitive-basis type for the electronic basis is $\text{el}$ and the number denotes the number $\sigma$ of states in the system, in this case two. This sets up a discrete (vector) representation for the $\sigma$ states.

The complete input file for the 4-mode 2-state pyrazine model, from which the above lines were taken, is displayed in Example 9.2.

9.3 Defining the single-particle basis for a non-adiabatic system

In an MCTDH propagation or relaxation a single-particle basis is needed for the representation of the wavefunction. For non-adiabatic systems there are two possible representations, termed single- or multi-set (see Sec. 3.5 of Ref. [1] for details).
OP_DEFINE-SECTION
  title  
    Pyrazine 4-mode model
  end-title
end-op_define-section

PARAMETER-SECTION

w10a = 0.09357, ev
w6a  = 0.0740 , ev
w1   = 0.1273 , ev
w9a  = 0.1568 , ev
delta = 0.46165, ev
k6a1 =-0.0964 , ev
k6a2 = 0.1194 , ev
k11  = 0.0470 , ev
k12  = 0.2012 , ev
k9a1 = 0.1594 , ev
k9a2 = 0.0484 , ev
lambda = 0.1825 , ev

end-parameter-section

HAMILTONIAN-SECTION

| modes | v10a | v6a | v1 | v9a | el |
|-----------------------------|
| 1.0*w10a | KE | 1 | 1 | 1 | 1 |
| 0.5*w10a | q^2 | 1 | 1 | 1 | 1 |
| 1.0*w6a  | 1 | KE | 1 | 1 | 1 |
| 0.5*w6a  | 1 | q^2 | 1 | 1 | 1 |
| 1.0*w1   | 1 | 1 | KE | 1 | 1 |
| 0.5*w1   | 1 | 1 | q^2 | 1 | 1 |
| 1.0*w9a  | 1 | 1 | 1 | KE | 1 |
| 0.5*w9a  | 1 | 1 | 1 | q^2 | 1 |
| -delta   | 1 | 1 | 1 | 1 | S1&1 |
| delta    | 1 | 1 | 1 | 1 | S1&2 |
| k6a1     | 1 | q | 1 | 1 | S1&1 |
| k6a2     | 1 | q | 1 | 1 | S2&2 |
| k11      | 1 | 1 | q | 1 | S1&1 |
| k12      | 1 | 1 | q | 1 | S2&2 |
| k9a1     | 1 | 1 | 1 | q | S1&1 |
| k9a2     | 1 | 1 | 1 | q | S2&2 |
| lambda   | q | 1 | 1 | 1 | S1&2 |

end-hamiltonian-section
end-operator

Example 9.1: An operator file for the pyrazine 4-mode 2-state model system.

In the single-set formalism, which is the default, the wavepackets on each surface are represented by the same single-particle function basis. As there is thus only one single-particle basis, the SPF-BASIS-SECTION has the same form as for adiabatic systems, e.g.

SPF-BASIS-SECTION
  v10a = 5
  v6a  = 6
  v1   = 4
  v9a  = 4
end-spf-basis-section
Example 9.2: An input file for the pyrazine 4-mode 2-state model system.

Note that no single-particle basis needs to be specified for the electronic “degree of freedom”,
as this is a complete basis set.

In the multi-set formalism, which is often more efficient than the single-set formalism,
the wavepackets on each surface are represented in a different single-particle function basis.
The number of functions desired for each state must therefore be given. The SPF-BASIS-
SECTION may read
The keyword `multi-set` selects the multi-set formalism. For instance, the line

```
v10a = 4, 3
v6a = 5, 4
v1  = 3, 3
v9a = 3, 3
```

requests four functions to be used for the wavepacket in the lower state 1, and three functions for the wavepacket in the upper state 2. The multi-set formalism usually requires fewer single-particle functions (per state) than the single-set formalism. This makes the former more efficient in most cases.

### 9.4 Building the initial wavepacket for a non-adiabatic system

For a non-adiabatic system with \( \sigma \) electronic states also \( \sigma \) initial wavefunctions have to be built (unless they are read from file). When generating the initial wavepacket the MCTDH program however assumes that only one electronic state is initially populated and hence sets all wavefunctions on other states to zero.

The initial wavefunction can therefore be defined in the same way as for adiabatic systems. The program has solely be supplied with the information which state is to be populated at the beginning. This is achieved using the `init_state` keyword in the INIT_WF-SECTION, as shown in Example 9.2. If this keyword is missing, state 1 is initially populated.

When a multi-set wavefunction is to be read from file it is convenient to use a `Read-Inwf` block, because then the wavefunction read and the system wavefunction do not need to have the same number of electronic states. See Section 7.7.
Chapter 10

Treating bosonic systems

While MCTDH is designed for distinguishable particles, it also allows for the treatment of indistinguishable particles. The ‘only’ conceptual complication that needs to be taken care of is the permutation symmetry encoded in the general rules of quantum mechanics: More precisely, \( \Psi \) is a valid wave function only if for any permutation \( P_{ij} \) of two particles we have \( P_{ij} \Psi = \pm \Psi \). The + sign holds for bosonic particles, which are the subject of this chapter; for fermions (−) there already exist specially modified versions of MCTDH. \(^1\) The consequence is that bosons live only in the symmetry-restricted Hilbert space

\[
\mathbb{H}_+ = \{ \Psi \mid P_{ij} \Psi = \Psi \quad \forall i,j \} \subset \mathbb{H}.
\]

Obviously, the most elegant way to extend the MCTDH ansatz

\[
\Psi(Q,t) = \sum J A_J(t) \Phi_J(Q,t)
\]  

(10.1)

would be to include only basis function \( \Phi_J \in \mathbb{H}_+ \), a demand clearly not met by the Hartree products employed in MCTDH. However, it is possible to circumvent this by simply projecting any wave function onto \( \mathbb{H}_+ \). This amounts to keeping the expansion coefficients \( A_J \) symmetric rather than the basis vectors themselves. In fact, any wave function will usually stay permutation symmetric under (real or imaginary) time evolution if both the initial state and the Hamiltonian are chosen as outlined below and if numerical errors are kept at bay.

10.1 Setting up the Hamiltonian

The Hamiltonian for identical bosonic atoms should of course be symmetric in all particles. For bosonic atoms with no more than binary interactions, it usually has the form

\[
H = \sum_i h(p_i, x_i) + \sum_{i<j} V(x_i - x_j),
\]

so the one-particle operator \( h \) (including kinetic energy) has to be listed in the operator file for any boson \( i \), while the interaction potential \( V \) requires a manual entry for any combination

\(^1\)For simplicity, we assume a system of spin-polarized one-dimensional bosons. The extension to higher dimensions is slightly more complicated insofar as one has to distinguish between particles \( (i = 1, \ldots, N) \) and degrees of freedom \( (\kappa = 1, \ldots, f) \). To match these two, different modes \( \kappa \) belonging to one and the same physical particle \( \#i \) have to be combined, cf. Sec. 5.3.
Treating bosonic systems

\[ i < j \] (Altogether, these are \( N(N-1)/2 \) terms, which naturally limits the application to few particles.)

As an example, the operator file for \( N = 3 \) one-dimensional bosons in a harmonic trap is shown in Example 10.1 (for more details, the reader is referred to [28]). In that case, \( h(p,x) = \frac{1}{2}p^2 + \frac{1}{2}x^2 \); the two-body potential \( V(x) = g\delta_\sigma(x) \), shaped as a normalized Gaussian of width \( \sigma \), has to be fitted to the direct-product form imposed by MCTDH. This is carried out as usual via \texttt{potfit} (set there e.g. \texttt{pes = gauss1d\{width=0.05\}}), the resulting \texttt{natpot} is included in the \texttt{LABELS-SECTION}. The numerical parameters are defined in the \texttt{PARAMETER-SECTION}, even though some of the values are conveniently reset in the input file (see below).

10.2 Modifying the input

The symmetrization mentioned above brings about some minor modifications of the way the wave function \( \Psi \) is handled. The MCTDH ansatz (10.1) is now simplified insofar as the single-particle functions are now identical, i.e., we have \( \Phi_J \equiv \varphi_j \otimes \cdots \otimes \varphi_j \) with a single set of functions \( \{ \varphi_j \mid j \leq n \} \).

This reflects in the input file as illustrated in Example 10.2: The \texttt{SPF-BASIS-SECTION} only gives the first orbital \( \varphi_1 \), while all others are mapped via the entry \( x2 = \texttt{id},1 \), etc.

It goes without saying that, by extension, the \texttt{primitive basis} also has to be identical for every boson. Again this is reflected in the input file, where the \texttt{PRIMITIVE-BASIS-SECTION} contains repetitions of the very same line

\[
x1 \ HO \ 125 \ xi-xf \ -4.0 \ 4.0
\]

for any \( x_i \).

A little less trivial is the choice of the initial wave function \( \Psi_0 \). As stated above, it must be permutation symmetric, a demand which can be met by the following standard choices.

- A Hartree state \( \Psi_0 = \varphi \otimes N \) is implemented trivially by including the following lines in the \texttt{INIT.WF-SECTION}:

  \[
  \texttt{build}
  \]
  \[
  \texttt{x1 eigenf spo}
  \]
  \[
  \texttt{x2 map x1}
  \]
  \[
  \texttt{x3 map x1}
  \]
  \[
  \texttt{end-build}
  \]

  Here \texttt{eigenf spo} specially selects the (lowest) eigenfunction \( \varphi \) of the single-particle operator \( \texttt{spo} \) previously defined in the operator file. This particular feature is not essential—in this case, one may as well use harmonic-oscillator functions (\texttt{HO}) instead—but we have included it simply to give a realistic example.

- More generally, a number (or Fock) state \( |n_1, n_2, \ldots \rangle \) can be selected, which denotes how many particles \( n_a \in \mathbb{N} \) occupy a given single-particle mode \( \varphi_a \) (where \( \sum_a n_a = N \)). Cast in standard MCTDH form, this is a sum over all permutations of the single configuration

  \[
  J = (j_1, \ldots, j_1, \ldots, j_n, \ldots, j_n).
  \]

  This is a little more cumbersome, since we have to keep track of all permutations of \( J \). A typical statement from the \texttt{INIT.WF-SECTION} now reads:
Note that the last block is the same as for the Hartree product above, telling MCTDH to select eigenstates $\varphi_a$ of the single-particle Hamiltonian spo. On top of that, the block A-coeff defines just which orbitals $a$ should be selected. In our example, this produces a single configuration $J = (2, 2, 3)$ (forget about normalization.) To make this permutation symmetric —i.e., a number state $|n_1 = 0, n_2 = 2, n_3 = 1\rangle$— the statement symcoeff has been added.

- If one performs a series of relaxations, e.g. with increasing interaction parameter $g$, then the most convenient choice for the initial state is one already obtained in some previous MCTDH calculation. This is done by simply casting the INIT.WF-SECTION as follows:

```plaintext
file= p3d1_19
symcoeff
```

This ensures that the restart file from the directory p2d1.19 is read as initial wave function. The symcoeff statement is added just to be one the safe side and make sure that the initial state is absolutely symmetric.

The input for potfit is very simple. One usually uses as many natpot terms as grid points. The fit is therefore exact.
### 3 bosonic particles, 1D, in a harmonic trap

**Example 10.1:** An operator file for $N = 3$ one-dimensional bosons in a harmonic trap.
Example 10.2: An input file for $N = 3$ one-dimensional bosons in a harmonic trap.
Chapter 11

Analysing the results employing the Analyse programs

The set of Analyse programs can be used to analyse the information from a calculation, which is stored in the various data files. For a complete list of programs, see the HTML documentation. If a program is started using the \(-h\) option, i.e.

\[
\text{analyse85 \ -h}
\]

where \textit{analyse} is the name of the program, e.g. \texttt{rdgpop}, a brief description of how to use the program, and a list of options will appear.

The programs are designed to be used together with the \texttt{GNUPLOT} program. In many cases, the option \(-g\) will produce a file complete with \texttt{GNUPLOT} commands, ready for immediate plotting. Some programs also support interactive plotting in conjunction with \texttt{GNUPLOT}. In these cases, starting the program with

\[
\text{analyse85 \ -inter}
\]

brings up a menu with options to choose what is to be plot, to change plotting boundaries, etc.

Here, a brief overview of only the most important programs will be given. As an example, we take the results from the wavepacket propagation of the NOCl system. First the system is relaxed on the S\(0\) surface using the input file \texttt{inputs/nocl0.inp}. Propagation is then made using the file \texttt{inputs/nocl1.inp}. The data files containing all the information about the calculation and the system evolution are then contained in the directory \texttt{nocl1}. This is the system used in the first tutorial.

11.1 The Analysis Interface

The \texttt{analysis} program provides a menu driven interface for running many of the \texttt{ANALYSE} programs. On typing

\[
\text{analysis85}
\]

a menu appears, as shown in Example 11.1.

An option is selected by entering the appropriate number. This may lead to further menus which allow the examination or plotting of various quantities of interest.
11.2 Interpreting the MCTDH output

During the propagation of a wavepacket, information about the system evolution is output to allow an easy visual check of how the calculation is progressing. If the keyword output was included in the RUN-SECTION of the input file, this information is written to the file output in the directory specified by the name keyword. If this keyword was omitted, this information is written to the screen during the calculation.

A section of the output from our example NOCl propagation is shown in Example 11.2. After information about the program version used, and where and when the calculation was run, starts the information about the system. This is output every \( t \) fs, where \( \text{tout} = t \) is the time specified in the RUN-SECTION of the input file.

At each output step the following information is give:

- Norm: The norm of the wavefunction expansion coefficients \( \sqrt{\sum J A_J^* A_J} \). This should remain close to 1.0, unless a CAP is used in which case the norm will disappear with the wavepacket.
**Example 11.2**: A section of an output file from the wavepacket propagation on the S1 surface of NOCl.

- **E-tot**: The total energy, i.e. expectation value of the Hamiltonian. This should remain constant, unless a CAP is used when the energy will disappear with the wavepacket.

- **E-corr**: The correlated energy, i.e. the expectation value of the terms in the Hamiltonian which correlate the degrees of freedom. (The correlated and uncorrelated Hamiltonian terms are listed in the op.log file).
• Delta-E: The loss in energy during the calculation, i.e. difference between the energy at time \( t \) and at time 0.

• Natural weights: The natural weights, i.e. eigenvalues of the one-dimensional density matrices, are given for each mode in the calculation. A weight of 1.0000 given here indicates that the least important natural orbital is present in 0.1% of the wavefunction.

• Mode expectation values: \( \langle q \rangle \) and \( \langle dq \rangle \) are the expectation values and variances for the position operator, where \( \langle dq \rangle = \sqrt{\langle q^2 \rangle - \langle q \rangle^2} \). This gives an idea of the spread of the wavepacket and a check on the grid used. If a DVR is used, \( \langle n \rangle \) and \( \langle dn \rangle \) are the expectation values and variances of the number representation in the corresponding FBR basis, a measure of which functions are populated. If a Legendre DVR is used (i.e. Leg, Leg/R or KLeg) the symbol is changed to \( \langle j \rangle \) and \( \langle dj \rangle \) to indicate, that the number representation is in this case just the expectation and variance of the angular momentum. If an FFT basis or the exponential-DVR is used, \( \langle p \rangle \) and \( \langle dp \rangle \), the expectation value and variance of the momentum operator are given.

At the end of the file should be written:

Propagation was successful.

and the CPU-time used for the calculation.

11.3 Checking the accuracy of a calculation

The accuracy of an MCTDH calculation depends on both the size of the primitive and the single-particle function bases. Analyse programs are available for both these tasks. Note that one has to be in the name directory when applying the analyse routines and scripts as shown below.

11.3.1 Checking the primitive basis size

The program **rdgpop** reads and evaluates the populations of the primitive basis functions, e.g. the grid points. This is used to check that enough primitive basis functions have been used for the calculation. The program requires the **gridpop** file, which is obtained by specifying the **gridpop** keyword in the RUN-SECTION of the input file.

The program can be used either to calculate the maximum population, or to evaluate the change of population with time of the points at the ends of the grid.

In the directory containing the data files, typing

```
rdgpop85
```

results in some information about the calculation and the primitive basis used for each degree of freedom in the calculation. The question

```
Number of grid-points to be summed over: nz =?
```
	hen appears. If 1 is input, then the populations output are those on the end grid points. If 2 is input, the output population for the beginning of the grid is the sum of the populations of the first and second points, while the output population for the end of the grid is the sum of the populations of the last and last-but-one points. And so on.

The next question asked is
100 11 Analysing the results employing the Analyse programs

Running number of degree of freedom: dof = ?

dof = 0 -> Print only maximum over time

Selecting 0 here results in the maximum population at the end grid points being displayed.

Maximal values (all times); final time: 30.00 fs

<table>
<thead>
<tr>
<th>#</th>
<th>dof</th>
<th>grid(begin)</th>
<th>grid(end)</th>
<th>basis(begin)</th>
<th>basis(end)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rd</td>
<td>.00045781</td>
<td>.000000064</td>
<td>.260884702</td>
<td>.000002414</td>
</tr>
<tr>
<td>2</td>
<td>rv</td>
<td>0.000000000</td>
<td>.000000000</td>
<td>.972981513</td>
<td>.000017179</td>
</tr>
<tr>
<td>3</td>
<td>theta</td>
<td>0.000000000</td>
<td>0.000000000</td>
<td>.150721535</td>
<td>.000005773</td>
</tr>
</tbody>
</table>

We see that the beginning of the \(rv\) grid is unpopulated, and this grid point could be removed without affecting the propagation quality. Likewise, the ends of the \(theta\) grid are unpopulated.

To avoid answering the questions, this result could be obtained by calling the program as

```
rdgpop85 1 0
```

If the number of grid points to be summed over is changed to 2, i.e.

```
rdgpop85 2 0
```

then the output is

```
Maximal values (all times); final time: 30.00 fs

<table>
<thead>
<tr>
<th>#</th>
<th>dof</th>
<th>grid(begin)</th>
<th>grid(end)</th>
<th>basis(begin)</th>
<th>basis(end)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rd</td>
<td>.001164114</td>
<td>.000002005</td>
<td>.593797892</td>
<td>.000013545</td>
</tr>
<tr>
<td>2</td>
<td>rv</td>
<td>0.000000000</td>
<td>.010178351</td>
<td>.993075095</td>
<td>.000088905</td>
</tr>
<tr>
<td>3</td>
<td>theta</td>
<td>0.000000000</td>
<td>0.000000000</td>
<td>.316521645</td>
<td>.000008258</td>
</tr>
</tbody>
</table>
```

The start of the \(rv\) grid is still unpopulated, and so 2 grid points could be removed at the start of this degree of freedom. By increasing the number of points over which the sum is made, it is possible to evaluate how many grid points can be removed. Whether it is possible to remove the grid points depends on the primitive basis being used. Thus the Leg-DVR used for the \(theta\) degree of freedom by definition runs from \(\pi\) to 0 (if symmetry is used \(theta\) runs from \(\pi/2\) to 0), and these points cannot be removed. (However, the use of the restricted Legendre DVR, Leg/R, allows to remove unused angular grid-points).

To chart the population of the end grid points, select a degree of freedom. Thus

```
rdgpop85 1 1
```

produces the file \(gpop.pl\), which contains the populations of the end of grid points as a function of time. The option \(-g\) writes information for GNUPlot to this file, and so

```
rdgpop85 -g 1 1
gnuplot -persist gpop.pl
```

produces a plot of the time-dependence of the end of grid points. A more convenient way to plot the time evolution of the population of the end grid points is provided by the \(plgpop\) script.

```
plgpop 1 1
```
The quality of the single-particle function basis is reflected in the populations of the natural orbitals (see Sec. 3.3 in Ref. [1]). If the calculation contains natural orbitals with a low population, these are not significant for the representation of the wavefunction, and the calculation is of a reasonable quality. Unfortunately, different properties have different convergence criteria, and it is not possible to give absolute figures for when the natural orbitals are insignificant. As a general rule of thumb, when the population of the highest (least populated) natural orbital is below 1% (i.e. a population below 0.01), the calculations will be reasonable, although convergence may be a way off.

Experience has shown that it is important that the single-particle function bases for all the modes are balanced, i.e. the lowest natural orbital populations are similar for all. There is little point spending effort on converging the single-particle function basis for one mode when the dynamics can be seriously affected by the poor representation of another mode.

The program `rdcheck` is used to check the natural orbital populations, and so show where more functions are required. Two basic pieces of information are required by the program: a state and a mode for analysis. If no arguments are given, the program prompts for what it requires. The basic information is provided by typing

```
rdcheck85 0 0
```

when in the directory containing the data files from a calculation. The arguments 0 0 select no particular state or mode. The program then prints some information about the system, and most importantly the maximum population of the highest natural orbital (lowest natural weight) is displayed.

```
Maximum over time of lowest nat.-weight; final time : 30.00 fs
mode   s = 1
  1 1.810E-03
  2 2.472E-04
  3 9.870E-05
```

This information says that the calculation should be of reasonable quality, as all mode contain natural orbitals that remain fairly insignificant.

If a mode is selected, the populations as a function of time can also be graphically displayed. The modes are numbered in the order in which they are listed in the SPF-BASIS-SECTION of the input file. For NOCl the order corresponds to the degrees of freedom rd, rv, theta, and so 2 selects the vibrational mode rv. The NOCl system has only one state, and so

```
rdcheck85 1 2
```

produces a file, `nat.pl` which contains the natural populations as a function of time. The `GNUPLOT` program can be conveniently used by including `GNUPLOT` data in this file, i.e.

```
rdcheck85 -g 1 2
gnuplot -persist nat.pl
```

would produce the plot shown in Fig. 11.1. Again, a more convenient way to produce this plot is provided by a `pl`-script. Just type:

```
plnat 1 2
```
11.4 Checking the efficiency of a calculation

The timing file, which is obtained by adding the keyword `timing` to the RUN-SECTION, contains information about how much time is spent in the various sections and subroutines of the program. This information can be used to improve the efficiency of a calculation. For instance, if in a CMF run the BS-integrator (used to propagate the single-particle functions) takes less than one or two percent of the total effort, one should combine more single-particle functions. If, on the other hand, the BS-integrator takes more than 80% of the total effort, one should remove some of the combinations. If the propagation of one certain mode takes much longer than the propagation of the other modes although the (combined) grid sizes are comparable, then check whether the (DVR) representation is appropriate. The information listed in the timing file can be extremely helpful. It is a good practice to always include the timing file.

11.5 Watching the system’s evolution

The program `showd1d85` is designed to plot the evolution of the system density along a degree of freedom. It reads the `gridpop` file, which can be created by adding the keyword `gridpop` in the RUN-SECTION of the input file. This file contains the one-dimensional

![Figure 11.1: The natural orbital populations for the single-particle function basis for the vibrational degree of freedom as a function of time for the photo-dissociation of NOCl.](image)
densities output at intervals specified by the \texttt{tout} keyword. This density can be plotted using the \texttt{showd1d85} program in conjunction with \texttt{GNUPLOT}.

For example, the NOCl photo-dissociation calculation has been run using the example input file 4.1. The command

\begin{verbatim}
showd1d85 -a -T f1
\end{verbatim}

requests that the density for the first degree of freedom (as listed in the PRIMITIVE-BASIS-SECTION of the input file, i.e. the dissociative mode, \texttt{rd}) is written to a file \texttt{den1d_f1}. The options make this file into a \texttt{GNUPLOT} grid file, complete with commands. The option \texttt{-a} (automatic) lets \texttt{showd1d85} call \texttt{GNUPLOT}. The 3D plot of density is shown in Fig. 11.2. Try also the other format options. A complete list of options is obtained through the command \texttt{showd1d85 -h}.

The similar program is \texttt{showspf85} which displays the single-particle functions (uncombined modes only, of course). Note that \texttt{showd1d85} reads the \texttt{gridpop} file while \texttt{showspf85} reads the \texttt{psi} file. Finally, there is \texttt{showrst85} which plots the single-particle functions of the restart file.
11.6 Determining photo-dissociation and photo-absorption spectra

The spectrum of a system can be generated using `autospec85`. This reads the autocorrelation function from the file `auto`, which is created if the keyword `auto` is included in the RUN-SECTION of the input file. The spectrum is then created over a chosen interval in energy space by Fourier transform. The shell script `plspec` calls `autospec85` and then `GNUPLOT`. It is often more convenient to use this script.

For example, the NOCl spectrum can be displayed by typing

```
plspec 0.6 2.0 ev
```

This spectrum was produced in the tutorial, and is shown in Fig. 2.1.

Before the autocorrelation function is Fourier transformed, it gets modified. To reduce artifacts of the Gibbs phenomenon, the autocorrelation function is multiplied with a filter function $\cos^n(\pi t/2T)$, where $n = 0, 1, 2$ and where $T$ denotes the final time (plus one time step) of the autocorrelation function. Due to the $t/2$-trick, see Eq. (167) of the MCTDH review, the propagation time is only $T/2$. The columns 2–4 of the `spectrum.pl` file list the results for the different $n$’s. When the option `-lin` is set for `autospec85`, then a second set of filters is used. In `plspec` the choice of the filter is made through the option `–g0 · · · –g5`, the filter `–g1` is default. See the HTML docu. For a comprehensive discussion of the filters see the lecture notes `INTRODUCTION TO MCTDH`, chapter 1.3. (The lecture notes can be downloaded from the literature downloads site, which is part of the MCTDH web-site). Here we only list the full-widths at half maximum (FWHM) of the energy filters, which are the Fourier transforms of the time filters. The entries in Table 11.1 are to be divided by the length of the autocorrelation function to yield the FWHM. Note that the filters $\tilde{g}_0'$ and $\tilde{g}_1'$ (which are called $g4$ and $g5$ in `plspec`) are non-negative. The energy-filters may be inspected by running `plspec -gn -r -t 100 -0.1 0.1 ev` with $n = 0, 1, ..., 5$.

<table>
<thead>
<tr>
<th>$\tilde{g}_0$</th>
<th>$\tilde{g}_1$</th>
<th>$\tilde{g}_2$</th>
<th>$\tilde{g}_3$</th>
<th>$\tilde{g}_0'$</th>
<th>$\tilde{g}_1'$</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.49</td>
<td>3.38</td>
<td>4.14</td>
<td>4.78</td>
<td>3.66</td>
<td>4.91</td>
<td>eV fs</td>
</tr>
<tr>
<td>20.1</td>
<td>27.3</td>
<td>33.2</td>
<td>38.6</td>
<td>29.5</td>
<td>39.6</td>
<td>cm$^{-1}$ ps</td>
</tr>
</tbody>
</table>

Table 11.1: FWHM values of the window functions $\tilde{g}_k$ times the length of the autocorrelation function. Remember that the length of the autocorrelation function is twice the propagation time, if the $t/2$-trick is used.

To introduce an additional damping, i.e. a Lorentzian or Gaussian broadening of the spectrum, the autocorrelation function may be further multiplied with $\exp(-((t/\tau)^i))$ where $\tau$ and $i = 1, 2$ are the two last arguments of `autospec85`. The exponential is ignored if $\tau = 0$. (For `plspec` this is the default).

Finally, if the option `-EP` is set, the Fourier transform is multiplied with the photon energy $\omega$ to arrive at an absorption spectrum. This multiplication in general requires to shift the spectrum (option `-e`) by the ground state (or initial state) energy. The multiplication with $\omega$ is omitted if the option `-FT` is given (this is the default). Note that only in this case one may use an energy interval containing negative energies (compare with Fig. 2.4).
11.7 Computing excitation and reaction probabilities

The analysis of scattering processes, i.e. the computation of excitation and reaction probabilities, is performed by evaluating the quantum flux going into a particular channel. The quantum flux is determined by the interaction of the time-dependent wavepacket with a CAP. The program \texttt{flux85} performs the necessary analysis. The \texttt{psi} file is read and the energy resolved flux is computed and written to the \texttt{flux} file. Additionally, this flux is divided by the energy distribution of the initial wavepacket to obtain the transition or reaction probabilities.

The latter step requires in general that the \texttt{mctdh85} program has computed the energy distribution of the initial wavepacket and has written it to the \texttt{enerd} file (see Section 7.11). (NB: The file \texttt{enerd} is called \texttt{adwkb} in older versions). Besides the \texttt{flux} file, \texttt{flux85} creates the files \texttt{flux.log}, \texttt{gtau} and \texttt{wtt}. When the file \texttt{gtau} is present, the flux program skips the time-consuming evaluation of the integrals $\langle \Psi(t) \mid W \mid \Psi(t+\tau) \rangle$ but reads the function $g(\tau)$ from the \texttt{gtau} file. (See the MCTDH review, section 8.6.3, Eq.(199), for more details). This allows to re-do the Fourier integrals for another energy interval very quickly. The option \texttt{-w} enforces the re-calculation of $g(\tau)$.

The file \texttt{wtt} contains the expectation values $W_{tt} = \langle \Psi(t) \mid W \mid \Psi(t) \rangle$. This information is useful for checking that the absorption process has finished. The shell script \texttt{plwtt} visualises the function $W_{tt}$, while \texttt{plflux} and \texttt{plflux -r} visualise the flux and the transition probability, respectively.

The program \texttt{flux85} cannot only determine the total energy resolved flux going into a particular arrangement channel (i.e. going into a particular CAP) but can also determine the flux which is projected onto final quantum states or which is weighted by an operator. This is probably best demonstrated by an example. Copy the file $\$MCTDH\_DIR/operators/nocl1.op$ to your tutorial directory and add the following lines to this operator file.

\begin{verbatim}
HAMILTONIAN-SECTION_vib
usediag
-----------------------------------------------------
modes  |  rd  |  rv  |  theta
-----------------------------------------------------
1.0    |  1   | KE   | 1
1.0    |  1   | v:NO | 1
-----------------------------------------------------
end-hamiltonian-section
\end{verbatim}

Then edit the input file \texttt{nocl1.inp} and set the propagation time to $t_{\text{final}}=60$ and re-run. Since \texttt{flux85} analyses the wavepacket as it is absorbed by the CAP, a longer propagation time is required as for converging the spectrum. Then execute the commands

\begin{verbatim}
flux85 -w -s 19 -lo 12 0.61 2.0 ev rd
mv flux flux.0
flux85 -w -s 19 -lo 12 -O vib -u 200. 0.61 2.0 ev rd
mv flux flux.op
flux85 -ed flux.0 -s 19 -lo 12 -O vib -u ev 0.61 2.0 ev rd
mv flux flux.op_r
flux85 -w -s 19 -lo 12 -P 2 eigenf vib 1 % 0.61 2.0 ev rd
mv flux flux.1
flux85 -w -s 19 -lo 12 -P 2 eigenf vib 2 % 0.61 2.0 ev rd
mv flux flux.2
flux85 -w -s 19 -lo 12 -P 2 eigenf vib 3 % 0.61 2.0 ev rd
mv flux flux.3
\end{verbatim}
The first flux-run evaluated the total flux. To display it, type

```
plflux -f flux.0
```

As you will notice, the plot looks very similar to the absorption spectrum shown in figure 2.1. It is not identical, though, as the definition of an absorption spectrum contains a factor $\omega$, the energy of the absorbed photon. This multiplication is omitted when `autospec85` (or `plspec`) is run with the `-FT` option, as we have done above. Now the spectra are identical as one observes when typing

```
plflux -G -f flux.0 -d spectrum.pl
```

This shows the flux and the Fourier transform of the autocorrelation function on top of each other.

Next we modify the flux by letting the operator of the vibrational energy act on it. Type

```
plflux -G -f flux.0 -d flux.op
```

and you will see the total flux in comparison with the vibrational energy weighted flux. It is now clear that the structures in the spectrum are due to vibrational excitation of the NO fragment. By the way, via the option `-u 200` the modified flux was multiplied by the factor 200. This was done to make it comparable with the total flux. Usually the option `-u` is followed by an energy keyword, e.g. `-u ev`, to transform the weighted flux from a.u. to a desired energy unit.

One may divide the weighted flux by the total flux to observe the vibrational energy content. The option `-ed flux.0` was used to input the file `flux.0` as energy distribution (the default is the file `enerd`, the generation of which, however, is only useful for a scattering – not half-scattering – problems). To visualise the quotient weighted-flux/total-flux, type
plflux -G -r -f flux.op_r

The structures below 0.8 eV are numerical noise because one divides a very small number by another very small number. Obviously, the higher the energy of the absorbed photon, the larger is the vibrational energy of the NO fragment. Compare this plot with the eigenenergies displayed in the flux.log file.

Finally we demonstrate the use of projectors. Type

plflux -G -f flux.0 -d flux.1 -e flux.2
plflux -G -a 1.0 -y 25. -f flux.0 -d flux.2 -e flux.3

The first plot shows the flux projected onto the vibrational ground state and the first excited state, respectively, in comparison with the total flux. The second plot is similar, but shows the flux projected onto the first and second excited state, respectively. The Fig. 11.3 displays the total and projected projected flux, similarly to the plots generated above.

11.8 Monitoring state populations of non-adiabatic systems

11.8.1 Diabatic populations

The (electronic) state populations of a multi-set run are written to the check file and are read by rdcheck85 which writes them to the file chk.pl. One then may use GnuPlot to plot the state populations, but it is easier to use plstate:

plstate

Note that plstate may also be used for runs evolving on one single potential energy surface. In this case the norm-squared of the wavefunction is plotted. In the presence of CAP’s this quantity is non constant.

In case of a multi-state calculation which uses the single-set formulation, the state populations can be plotted with showd1d85. Use the command:

showd1d85 -a -E f<el.DOF>

where <el.DOF> must be replaced with the DOF-number of the electronic state. The numerical values of the populations can be found on the file den1d,f<el.DOF>, which is created by showd1d85.

Alternatively, the populations may be read from the output file. Use the command:

fgrep population output | sed ‘s/population : //’ > dpopfile

to write the state populations to the file dpopfile.

11.8.2 Adiabatic populations computed with adpop

Because mctdh usually works in the diabatic representation the diabatic populations can be easily calculated (see above). But if one is interested in the adiabatic state populations, these are more difficult to obtain. There are two possible ways.

The first possibility is to use the analyse program adpop. This program reads the psi file of an mctdh run and the so-called pes file, which is a special operator file in which all derivative operators are ignored. The pes file is conveniently generated by setting the option -pes, i.e. running “mctdh85 -pes inpfile”. Moving to the name directory, one may then
calculate the diabatic and adiabatic state populations by running \texttt{adpop85}. If needed, one- and two-dimensional adiabatic densities can be calculated by adding the modelabels to the program call:

\texttt{adpop85 v9a v10a,v6a}

In this case the one dimensional adiabatic density for the \texttt{v9a} degree of freedom and the two dimensional density for the \texttt{v10a-v6a} degrees of freedom will be created as well as the diabatic and adiabatic state populations. The created files are: \texttt{adp} (for the populations), \texttt{adp,v9a} (for the one dimensional density), \texttt{adp,v10a,v6a} (for the two dimensional density) and the log file \texttt{adp.log}. To create two-dimensional densities, the two modelabels must be separated by a comma. The modelabels for one-dimensional densities stand alone. To visualise the density files created by \texttt{adpop} use the shell script \texttt{pladpop} that can plot one-and two-dimensional densities. For plotting the state populations simply use \texttt{plgen}.

As the calculation runs over the full primitive grid, the calculation is slow. Analyzing one wavefunction takes about \([\text{grid-dimension}] \times [\text{A-vector length}] \times 1.5 \times 10^{-8} \text{ s} \) on a 3 GHz P4. The adpop calculations can be accelerated by setting the options \(-q\) or \(-mc\). The first option enables the \textit{quick modus} where all points are ignored for which the product of the one-particle grip-populations (i.e. the 1D-densities) are smaller than some threshold. The loss in accuracy is usually negligible (and can be controlled by setting the parameter \texttt{qtol}). The Monte-Carlo integration is, of course, less accurate, but allows to attack problems which are not feasible for adpop using direct integration.

\subsection*{11.8.3 Adiabatic populations computed with \texttt{adproj}}

If one is interested in the adiabatic state populations alone one should use the second variant, which is more efficient, especially for larger systems (\(\geq 5\) degrees of freedom). However, this way is more cumbersome than the \texttt{adpop} calculation. The procedure is explained in the following.

First a \texttt{pes} file must be created, often this has already been done by the previous \texttt{mctdh}-run. After this, running \texttt{adproj} generates several \texttt{vpot} files which contain the adiabatic surfaces and the projection operator matrix elements for each electronic state. As an example the file \texttt{apr.p1.12} contains the matrix element \((1,2)\) of the projector (\(p = \text{projector}\)) for the first adiabatic electronic state. In contrast, the file \texttt{apr.v2} contains the second adiabatic potential energy surface (\(v = \text{potential}\)). Then the \texttt{vpot} files must be fitted by \texttt{potfit} to bring them into the MCTDH product representation. To calculate the expectation value of the projector one needs the projector in form of an MCTDH \texttt{oper} file. This \texttt{oper} file can be created by an \texttt{mctdh}-run with the keyword \texttt{genoper} in the RUN-SECTION. After the \texttt{oper} file has been created, the expectation value of the projector can easily be calculated with the \texttt{expect} analyse routine. This requires that the wavefunction has been stored (\texttt{psi} file). Alternatively one may compute the adiabatic populations \textit{on the fly} by setting the keyword \texttt{expectation} in the RUN-SECTION of the \texttt{mctdh} propagation.

Here’s a list of all steps needed to compute the adiabatic populations:

1. Create an \texttt{pes} file by an \texttt{mctdh-genpes}-run.

2. Create \texttt{vpot} files with \texttt{adproj} as explained above. (Just run \texttt{adproj} in the name directory of step 1. \texttt{adproj} will read the \texttt{pes} file.)
Example 11.3: The input file, projfit.inp, for the potfit calculation.

3. Fit the vpot files of the projector with potfit to create natpot files. As potfit requires an input file, an example is shown in Example (11.3). Here potfit will read the vpot file apr_pl11 and will create the natpot file in the name directory projector111. In the OPERATOR-SECTION the keyword pes = none must be given. Note that there must not appear an electronic degree of freedom in the PRIMITIVE-BASIS-SECTION. A similar input file must be generated for each matrix element of the projector.

4. Create a projection operator file by performing an mctdh-genoper-run using the just created natpot files. For this mctdh-run an input and an operator file must be written. As an example, these files for the creation of a projection operator are shown in Example (11.4) and (11.5). In the operator file, projector1.op, one uses the natpot files which are in the directories projector111, projector112 and projector122. Note that the SPF- and PRIMITIVE-BASIS-SECTIONs must be identical to the ones used in the mctdh-propagation-run where the psi-file was created that will be used for the calculation of the adiabatic populations.

5. Calculate the expectation value of the projection operator with the analyse routine expect. This requires the wave function that is contained in the psi file. Alternatively the adiabatic populations can be computed on the fly by setting the expectation keyword in the RUN-SECTION. In this case it is not necessary to store the wavefunction.

Notice that the adproj run provides also the vpot files of the adiabatic surfaces. This makes it possible to perform propagation in the adiabatic approximation.

Typical calculation times for the adpop-run are about 20s (for 3D) up to 9h (for 6D). Each calculation was performed on a 2.6GHz processor and the psi files contained 121 timesteps (3D and 6D pyrazine models). In the case of using adproj it takes about 10s (3D) up to
RUN-SECTION
name = proj1 genoper
title = pyrazine 3D projector.
end-run-section

OPERATOR-SECTION
opname = projector1
end-operator-section

SPF-BASIS-SECTION
multi-set
  v10a = 7, 7
  v6a = 10, 10
  v1 = 6, 5
end-spf-basis-section

PRIMITIVE-BASIS-SECTION
  v10a HO 20 0.0 1.0 1.0
  v6a HO 30 0.0 1.0 1.0
  v1 HO 20 0.0 1.0 1.0
  el el 2
end-primitive-basis-section
end-input

Example 11.4: The input file, proj1.inp, for the mctdh-genoper-run.

OP_DEFINE-SECTION
title
projection-operator for 3D pyrazine
end-title
end-op_define-section

LABELS-SECTION
P11 = natpot{projector1111}
P12 = natpot{projector1122}
P22 = natpot{projector1222}
end-labels-section

HAMILTONIAN-SECTION_projector1
usediag
----------------------------------------------------------
modes | el | v10a | v6a | v1
----------------------------------------------------------
1.0   | S11 | P11
1.0   | S22 | P22
1.0   | S12 | P12
----------------------------------------------------------
end-hamiltonian-section
end-operator

Example 11.5: The operator file, projection1.op, for the mctdh-genoper-run.
11.9 Plotting 2D cuts through the system density

0 = stop
1 = plot to screen
2 = print plot
3 = save plot to a postscript file
4 = save plot to a gnuplot file (use after 1 or 2)
5 = save data to an xyz file
9 = toggle re-plot (get new set of contours)
10 = change plot task (plot reduced density)
20 = change coordinate section ( rd=x rv=y theta=1.545 Time=0.000)
30 = change coordinate bounds
40 = show coordinate info
50 = change a single coordinate
80 = change coordinate units
90 = change Z-axis units (au)
110 = toggle contour mode (linear)
120 = change number of contours (21)
150 = toggle grid (off)
160 = toggle surface (off)
170 = toggle contour lines (on)
240 = toggle key (on)
245 = toggle title (on)
250 = toggle show points (off)
260 = toggle stick spectrum (off)
270 = toggle smooth curve (off)
280 = change time-slice format, e.g. movie(step-through)
285 = change time-step ( 1)
290 = toggle no-weights (off)
400 = Overlay plots (off)
900 = toggle gnuplot output format (on)
910 = change printer (lpr)
920 = change GNUplot command

Example 11.6: The start-up menu in the showsys program to enable interactive plotting of the system density and potential energy surfaces.

5min (6D) for the same Hamiltonians and wavefunctions. Note that in the potfit step of the second variant there are, as usual, less natpot terms than grid points (See Example 11.3). The numbers in the NATPOT-BASIS-SECTION were chosen such that there was virtually no difference to the numerically exact adpop-run.

11.9 Plotting 2D cuts through the system density

The program showsys can be used to display one- and two-dimensional cuts through the system density using the psi file. This must have been generated during a propagation run by including the psi keyword in the RUN-SECTION of the input file. If this file has not been generated, one-dimensional densities may be still be plotted from the gridpop file using the showd1d program (see Sec. 11.5).

The options are accessed by typing in the number given and responding to the questions. The words in brackets indicate what the present option is. Some options lead to a change in options being displayed.
Option 10 can be used to change between various plot tasks. What is possible depends on the system: for a non-adiabatic system the choices include not only plotting the reduced density, but plotting a cut through the adiabatic or diabatic wavefunction. These plots should be used with caution as the values are often extremely low in a cut. If a pes file is present (see Sec. 11.10), cuts through the PES may also be plotted. If there is more than one electronic state, one may chose between a diabatic or adiabatic representation of the potential.

Option 20 allows a different cut to be chosen. Enter either $x$, $y$, or a number for each DOF. Information about the mode boundaries is given by option 40. The program then chooses the grid point nearest to the selected coordinate for the plotted cut. Note that when densities are plotted, the values of the numbers given are irrelevant. They merely serve as space-holders. The density is the integral of $|\Psi|^2$ over all those DOFs, which are labelled with a number.

Option 400 allows to generate overlay plots, i.e. plotting a density on top of the contour lines of the potential. Before using this menu point, the potential plot data has to be stored to some file by using menu point 5. (See the tutorial Sec. (2.1) and Fig. 2.2 for an example).

### 11.10 Plotting cuts through the potential energy surfaces

The program showsys can be used to display one- and two-dimensional cuts through the potential energy surfaces of the system. Before this can be done, the MCTDH program must be used to generate a pes file from the Hamiltonian information in the operator file (see Chapter 6). The pes file is an operator file from which all terms are removed which contain derivative operators or CAPs.

To generate a pes file, set up an input file specifying the system and operator with a PRIMITIVE-BASIS-SECTION, A SPF-BASIS-SECTION, and an OPERATOR-SECTION. While no information is required about the single-particle function basis, the SPF-BASIS-SECTION is required as it also defines which degrees of freedom defined in the PRIMITIVE-BASIS-SECTION are included. In the RUN-SECTION, the keyword genpes is then required, along with the name of a directory in which to store the new file. Now the MCTDH program is run, and the file name/pes generated. The op.log.pes file contains information on the function that has been set up, and the log is now called log.pes. This makes it possible to use the same name-directory as the propagation run.

Rather than editing the input file it is often more convenient to use options. E.g.

```
mctdh85 -pes noc11
```

will generate a pes file of the NOCl S1-surface and stores it in the name directory.

Now change to the directory where the pes file has been stored and start the showsys program. The program is also able to generate two-dimensional plots of the system density from the psi file. If only pes plotting is required then start the program using the -pes option:

```
shows85 -pes
```

If both pes cuts and density plots are wanted, this option should not be used. A menu appears which allows the interactive generation of plots. This is shown in Example 11.6.

When the potential used is given by a natpot file, generated by the potfit85 program (see next Section), then it is more convenient to use the showpot85 program to visualise the potential energy surface. showpot85 is menu driven, similar to showsys85, and it allows to plot 1D and 2D cuts of the original surface, of the natural potential fit, and of differences
between them. When the parameter natpot-cut (menu point 500) is larger than zero, then all natural potential terms, the supremums norm of which is smaller than natpot-cut, are removed. A proper use of natpot-cut may reduce the number of potential terms (by a factor 1/2, or so) with only marginally reducing the accuracy of the fit. The parameter natpot-cut is also available in mctdh85. The number of omitted terms is protocolled in the showpot.log file.
Chapter 12

Using the Potfit program

12.1 Transforming a potential to product form

For optimal performance, the MCTDH algorithm requires the Hamiltonian to be given as a sum of products of single-particle operators (MCTDH product form). The kinetic energy operator usually is in MCTDH product form, but the potential is often given as a multidimensional function. The program potfit is able to transform a given potential energy surface to MCTDH product form. For small systems (e.g., 3D) it does this job fast and reliable. However, in contrast to the MCTDH program, which avoids using the primitive product grid, potfit has to employ the full primitive product grid. The computational resources used by potfit thus increase much more strongly with the size of the system than the ones required by MCTDH.

The numerical effort of potfit can be reduced, if the potential surface is partly given in product form. For example, if a 6D surface reads

\[ V(u, v, w, x, y, z) = V_1(u, v, w)V_2(x, y, z) + V_3(u, v, w)V_4(x, y, z) \] (12.1)

one should, of course, apply potfit to \( V_1, ..., V_4 \) individually, rather than applying it directly to \( V \).

Similar to mctdh85, potfit85 is started with giving the path of an input file as argument.

potfit85 <inputfile>

The input file is structured similar to the MCTDH input file. To understand it, one must be familiar with the basics of the potfit algorithm. The potfit algorithm is described in Ref. [1] and in the original papers [29, 30]. In short, the fit to product form is performed in two steps. First, the potential density matrices are diagonalised to obtain the natural potentials. The thus generated product representation of the potential minimises (to a very good approximation) the overall \( L^2 \) error. As there are regions of greater and lesser physical importance, a better representation can be achieved by introducing weights which emphasise the regions of physical importance. Separable weights, i.e., weights that act on one degree of freedom only, can be incorporated into the first step. More powerful, however, are correlated weights, i.e., weights which cannot be written in product form. In a second step one thus may iteratively improve the representation by employing correlated weights. The correlated weights are implemented as relevant regions. A relevant region may be defined, e.g., as those areas where the potential energy is below some threshold, \( V(R) < V_{max} \). The program then iteratively improves the fit in the relevant region (while making it worse in the non-relevant region).
Example 12.1: A potfit input file for the NOCl S1 surface.

The output files of potfit are structured similarly to the MCTDH ones. There are the files output, input, log and timing. The prodwei file lists the separable weights and the file iteration compiles various error measures for each iteration step. The script plpweight reads prodwei and plots the separable weights, and the script plpit reads iteration and plots the error measures versus the number of iterations. The file natpot finally contains the natural potential fit which may be read by the MCTDH program.

A potfit input file, noclpot.inp, is shown in Example 12.1. The RUN-SECTION defines the number of iterations to be performed, the name directory, and the files to be opened. The potential is first evaluated on the full product grid and written to the file vpot. Note, vpot is needed by showpot when plotting the exact potential.

The OPERATOR-SECTION specifies the potential energy surface to be used and defines cuts which remove large (positive and/or negative) potential values which, if kept, would slow down the integrator.
The NATPOT-BASIS-SECTION defines how many natural potentials are used for the fit. The more natural potentials one includes, the more accurate is the fit but the slower is the MCTDH calculation. One of the degrees of freedom should have the argument contr. This is the mode, over which a contraction is performed (see Refs. [1, 29, 30]). Contract over that degrees of freedom which converges most slowly. Inspect the output to see which one it is. One should avoid to contract over a mode if it is defined on a much larger grid than the other modes. To decide how may natural single particle potentials should be included in the potfit, one should inspect the output file which displays the natural populations as well as the sums of neglected natural populations. This will be discussed below.

MODE COMBINATION is also possible. In the NATPOT-BASIS-SECTION, the degrees of freedom can be combined into a single mode in the same way, as it is done in the spf-basis-section for an mctdh run. If the resulting natpot file is used in a mctdh run, the degrees of freedom combined in potfit must also be combined in the spf-basis-section (note that the order of the degrees of freedom in the mode must be identical in the natpot-basis-section and in the spf-basis-section). If the degrees of freedom are not combined in the natpot-basis-section, they still can be combined in a mctdh run. There are, however, restrictions. For each combined mctdh-mode, potfit must use precisely the same combination, or may treat all degrees of freedom of this mode uncombined. Combining degrees of freedom can reduce the number of natural potentials needed for convergence. This will be important for large systems.

The PRIMITIVE-BASIS-SECTION must be identical to the PRIMITIVE-BASIS-SECTION of the following MCTDH calculation, except for the ordering of the degrees of freedom. This ordering, however, must be consistent with the ordering of the arguments of the pes to be fitted. If one is insecure about the latter ordering, inspect the file source/opfuncs/funsrf.f and search for the name of the particular pes under discussion. See also the HTML documentation. NB: One may use the order keyword (OPERATOR-SECTION of the potfit input file) to define a new order in which the arguments are passed to the surface routine. However, this does not work for the readsrf surface. See the HTML documentation for details.

Note that mctdh uses the modelabels to associate the natpot terms with the DOFs. If this is not wanted one may give the keyword ignore as a parameter to the natpot keyword in the LABELS-SECTION. In this case one must use a numbered input (e. g. 16&2&3 V ) in the HAMILTONIAN tableau to indicate on which DOFs and in which order the natpot shall operate. See the HTML documentation for details.

The SEPARABLE-WEIGHT-SECTION and the CORRELATED-WEIGHT-SECTION finally define the separable and correlated weights, respectively. See the HTML documentation for details.

The OUTPUT FILE contains important information on the natural populations and on error measures. Shown in Example 12.2 is an excerpt of an potfit output-file, which is generated by running the input Example 12.1. The block named Trace - Sum of all preceding Natural Weights [eV**2] displays the sum of "neglected" reduced natural weights, e. g. the second entry, $0.5371E−03$, is the sum of eigenvalues 3 to 36. This sum is directly related to the fit error. As in this example we took 5 and 4 single particle potentials into account, the estimated error is given by $\sqrt{0.2817E−04 + 0.2474E−05} = 0.5535E − 02$ in eV, i. e. 5.5 meV. This estimate is printed in the output below the line Global (weighted) L2 error estimated from neglected ..., and it
12.1 Transforming a potential to product form

************* Mode: 1 rd *************
Trace of reduced density matrix : 0.8297 au, red. trace 0.8399 eV**2
Number of eigenvalues considered: 5 / 36
Reduced Eigenvalues (Natural Weights) [eV**2]:
1  0.8363E+00  0.3063E-02  0.3058E-03  0.1667E-03  0.3651E-04  0.1198E-04
7  0.5736E-05  0.4436E-05  0.2863E-05  0.1240E-05  0.1052E-05  0.4074E-06
13 0.1324E-06  0.1037E-06  0.8599E-07  0.4732E-07  0.3625E-07  0.1433E-07
19 0.7491E-08  0.7248E-08  0.5348E-08  0.3652E-08  0.1783E-08  0.7816E-09
25 0.4284E-09  0.3753E-09  0.2785E-09  0.2162E-09  0.1371E-09  0.6923E-10
Trace - Sum of all preceding Natural Weights [eV**2]:
1  0.3600E-02  0.5371E-03  0.2313E-03  0.6467E-04  0.2817E-04  0.1618E-04
7  0.1045E-04  0.6010E-05  0.3147E-05  0.1907E-05  0.8552E-06  0.4478E-06
13 0.3154E-06  0.2118E-06  0.1258E-06  0.7847E-07  0.4222E-07  0.2789E-07
19 0.2040E-07  0.1315E-07  0.7802E-08  0.4150E-08  0.1585E-08  0.8030E-09
25 0.1157E-08  0.7814E-09  0.5028E-09  0.2866E-09  0.1495E-09  0.8030E-10
Abs(Trace - Sum of relevant Natural Weights) [eV**2]: 2.81665E-05
Sqrt(Abs(Trace - Sum of rel. Nat. Weights)) [meV]: 5.3072

************* Mode: 2 rv *************
Trace of reduced density matrix : 0.8297 au, red. trace 0.8399 eV**2
Number of eigenvalues considered: 4 / 24
Reduced Eigenvalues (Natural Weights) [eV**2]:
1  0.8370E+00  0.2502E-02  0.3699E-03  0.2115E-04  0.1505E-05  0.5984E-06
7  0.1717E-06  0.9290E-07  0.4180E-07  0.3013E-07  0.1483E-07  0.7315E-08
13 0.4230E-08  0.3149E-08  0.1693E-08  0.1479E-08  0.9545E-09  0.3832E-09
19 0.1571E-09  0.2029E-10  0.6050E-12  0.1463E-18  0.2231E-36 -0.2449E-18
Trace - Sum of all preceding Natural Weights [eV**2]:
1  0.2896E-02  0.3935E-03  0.2313E-03  0.6467E-04  0.2817E-04  0.1618E-04
7  0.1990E-06  0.1061E-06  0.6434E-07  0.3421E-07  0.1938E-07  0.8030E-09
13 0.7838E-08  0.4688E-08  0.2995E-08  0.1516E-08  0.5612E-09  0.1780E-09
Abs(Trace - Sum of relevant Natural Weights) [eV**2]: 2.47378E-06
Sqrt(Abs(Trace - Sum of rel. Nat. Weights)) [meV]: 1.5728

************* Mode: 3 theta *************
Contracted mode. Dimension = 60 / 60
Global (weighted) L^2 error estimated from neglected natural weights:
5.5354 meV 44.646 cm^-1 2.0342E-04 a.u.
Weighted rms-error on rel. grid points [meV]: 3.9734 1.4602E-04 au
Weighted rms-error on all grid points [meV]: 5.4941 2.0190E-04 au
Unweighted rms-error on rel. grid points [meV]: 31.2338 1.1518E-03 au
Unweighted rms-error on all grid points [meV]: 133.0489 4.8895E-03 au
Max. absolute error on rel. grid points [eV]: 2.2438 8.2457E-02 au
Max. absolute error on all grid points [eV]: 2.2438 8.2457E-02 au

Example 12.2: An excerpt of a potfit output file for the NOCl S1 surface.
Using the Potfit program

compares well with the numerically evaluated error (weighted rms-error on all..) which is 5.4941 meV. By the way, the choice 5/4 for the number of natural potentials is a bit unbalanced, as the sums of neglected weights of the two DOFs are quite different. A more balanced choice would be 5/3 or 9/4, which would lead to (estimated) errors of 7.2 meV or 2.4 meV, respectively.

The following 10 iterations reduce the fit error to 2.202 meV on the relevant region, whereas the global error on all grid points increases to 9.067 meV. Note that weighted always refers to separable weights and relevant refers to correlated weights. For large systems the numerical evaluation of the rms-error may become costly. In such a situation it may be useful to switch this evaluation off or perform it only every \( n \)-th iteration step. See the HTML documentation for details.

12.2 Using ab initio data

An interesting feature of the MCTDH package is the possibility to define the potential energy operator (or a part of it, or in general any local operator in configurational space) directly from ab initio data, in a way that can later be used by a MCTDH calculation. The multidimensional grid, on which the ab initio data is collected, however, must be a product grid. In general, there are no further restrictions, e.g., an equidistant distribution of the grid points is not required. The potfit program can be used in order to transform the supplied data into a product form, practically a natpot file, that can later be used in the MCTDH simulations. This is similar to the transformation of general multidimensional functions into product form that has been covered in chapter 12.1. Also, the primitive grid where the ab initio data is collected happens to be usually rather sparse, due to the cost of evaluating the desired property on each point. The MCTDH package provides the chnpot utility, which allows to interpolate between natural potentials defined in different primitive grids. Therefore, the user may collect the ab initio data in a rather sparse primitive grid that can be later be interpolated into a more suitable one for the dynamical simulation phase. These operations will be discussed in detail in this chapter.

The practical implementation of the MCTDH algorithm uses DVR’s for the primitive basis, whose points define the primitive grid (see Chapter 4). It is assumed in our discussion that the value of some property, e.g., the potential energy, has been collected on the points defined by the primitive grid using some external program. One has to differentiate between the actual points of the primitive grid in each coordinate and the associated value of a certain property. As will be immediately seen, both pieces of information are given separately to the programs that have to use them.

There are mainly three ways to use ab initio data in a MCTDH calculation:

12.2.1 Using ab initio data directly with the mctdh program

This is the least flexible of the possibilities being discussed, but the concepts that will be introduced apply equally to the other procedures. It corresponds to the direct path to the usage of MCTDH as depicted in Fig. 12.1. First, let’s assume that we have the ab initio data values at the points defined by some primitive grid. The primitive grid being used should correspond, for each coordinate, with some of the DVR’s defined in the MCTDH program (What to do when this is not the case, will be covered in the following sections). As usual, the primitive grid is defined in the PBASIS-SECTION of the input file. For example one could have something like:
12.2 Using \textit{ab initio} data

\textbf{PRIMITIVE-BASIS-SECTION}
\begin{align*}
\text{x} & \sin 17 \quad -2.4 \quad 2.4 \\
\text{y} & \sin 21 \quad -2.5 \quad 3.5
\end{align*}
\textit{end-primitive-basis-section}

for coordinates \textit{x} and \textit{y}. The information concerning where the actual data values are found is given in the \textbf{LABELS-SECTION} of the operator file. One has to define a new label making use of the \texttt{readsrf} keyword:

\textbf{LABELS-SECTION}
\begin{align*}
\text{vdat} & = \texttt{readsrf(path-to-file S)}
\end{align*}
\textit{end-labels-section}

\textit{S} can be either \texttt{ascii} or \texttt{binary} depending on the file to be read. \texttt{path-to-file} is the absolute or relative path to the file containing the data. The newly defined \textit{vdat} label can then be used in the \textbf{HAMILTONIAN-SECTION} of the operator, for example:

\textbf{HAMILTONIAN-SECTION}
\begin{align*}
\text{---------------------} \\
\text{modes} & \quad | \text{x} | \text{y} \\
\text{-------------} \\
1.0 & |1\&2 \text{ vdat} \\
\text{other operator lines} \\
\text{end-hamiltonian-section}
\end{align*}

The file containing the data values consists of a single column of numeric entries, written so that the first index runs fastest. The order of the indexes is defined by the \texttt{|i\&j\&k\&...} construct in the \textbf{HAMILTONIAN-SECTION} as shown above. In our example the file has to be created so that it could be read by the following pseudo-code, where the \textit{i} index runs on the \textit{y} coordinate:

\begin{verbatim}
iterate \textit{i} in range 1 to 21:
   iterate \textit{j} in range 1 to 17:
      read \textit{v}(j,i)
   end iterate
end iterate
\end{verbatim}

One should note the following: the described procedure implies that both the \textit{x} and \textit{y} coordinates belong to the same combined mode and there are no other coordinates are present in this \textit{mctdh}-particle. Otherwise the program would treat the new potential as a \texttt{muld-potential}, i.e. a multi-dimensional potential. This slows down the performance of MCTDH. Therefore, the procedure outlined above is most useful when the potential operates on one combined mode (MCTDH particle) exclusively.

\textbf{12.2.2 Using the \textit{potfit} program}

A second alternative is to use the \texttt{potfit} program to convert the \textit{ab initio} data to product form, and then use \texttt{mctdh}, as it is shown in Fig. 12.1. This possibility circumvents the inconveniences described at the end of the previous section, since the different degrees of freedom can then be used in different combined modes, or be simply uncombined. The usage of \texttt{potfit} has been covered in chapter 12.1 of this guide, so here it will only be covered how to make it read an \textit{ab initio} surface defined on a primitive grid. The \texttt{readsrf} keyword (see previous section for details) comes now into play in the \textbf{OPERATOR-SECTION} of the input file of \texttt{potfit}:
120 12 Using the Potfit program

As in the previous case, the file containing the data values has to be written with the index of the first specified degree of freedom running fastest (the most internal loop), the second degree of freedom in the second most internal loop and so on. The order of the degrees of freedom is given by the PRIMITIVE-BASIS-SECTION in the potfit input file. The program potfit will generate a natpot file to be used directly by mctdh. Remember that mctdh uses the modelabels to associate the natpot terms with the DOFs.

12.3 Extra flexibility, combining potfit and chnpot

The maximum flexibility in the usage of ab initio data is accomplished by combining potfit with the chnpot utility. There are two main reasons why the initial primitive grid in which the ab initio data points are given should be transformed into a more suitable one. First, the given points may be too sparse, and a more dense grid is desired for the dynamical simulation phase. Second, the primitive grid where the points are supplied does not correspond to a DVR defined in the MCTDH code.

12.3.1 Dealing with an arbitrary primitive grid

In case that the ab initio data values are given in a grid that does not correspond to a DVR known to MCTDH, the external keyword can be used in the PRIMITIVE-BASIS-SECTION of the potfit input file:

```
PRIMITIVE-BASIS-SECTION
x  external 16 path_to_x_grid <unit>
y  external 16 path_to_y_grid <unit>
end-primitive-basis-section
```

path_to_x_grid is the absolute or relative path to a file containing, in one column, the values $x_i$ of the $x$ coordinate, in this case, a total of 16 entries. One needs to create a file with this information for every external entry. Optionally, a unit may be given to convert e. g. Angstroem to au or degree to radian. As before, the information concerning where the ab initio data points are found is given in the OPERATOR-SECTION of the same input file. After execution, the program yields a natpot file and a dvr file. One should note that these natpot and dvr files cannot be used directly in a simulation, since only the grid points of the DVR are found on the dvr file (and not the matrices representing the derivative operators). However, any natpot whose primitive grid is known (through the corresponding dvr file) can be interpolated to a desired DVR by the chnpot utility, yielding new natpot and dvr files to be used in the simulation phase. This is covered in the next subsection. (As a remark, the external keyword is more powerful and the non-local part of an arbitrary DVR can also be supplied, see the HTML documentation for this advanced feature).

12.3.2 Transforming between two natural potentials with chnpot

The chnpot utility interpolates a given natpot file into a new primitive grid corresponding to the desired DVR. It needs to know where to find the initial natpot and dvr files. An example input file for chnpot reads:
12.3 Extra flexibility, combining \textit{potfit} and \textit{chnpot}

![Diagram](image)

Figure 12.1: Main concepts involved in the usage of \textit{ab initio} data with the MCTDH package

```
RUN-SECTION
name = <S1>
dvrdir = <S2>
natpotdir = <S3>
end-run-section

PRIMITIVE-BASIS-SECTION
  x  sin 45  -2.4 2.4
  y  sin 45  -2.5 3.5
end-primitive-basis-section

Fit-Section
  x  spline
  y  spline
end-fit-section
end-input
```

The RUN-SECTION contains keywords that control the program execution, e.g., where the (old) initial files are found, where to store the (new) results, etc. See the HTML documentation for a detailed description of every keyword. The PRIMITIVE-BASIS-SECTION specifies the new DVR, and hence the new primitive grid. The Fit-Section describes how each degree of freedom has to be interpolated. A detailed description of the \textit{chnpot} command line and input file options, is maintained in the HTML documentation. Here some hints are given in how to use this utility: after \textit{chnpot} has been executed, the user should check that the \textit{natpot} obtained has been correctly interpolated. This can be done conveniently with the \textit{showpot} utility. Original potentials with discontinuities or regions where the function slope changes abruptly can lead to oscillatory behavior of the interpolated function. Depending on the starting data points, the user may have to find the most convenient interpolation scheme for each degree of freedom. The program can perform spline and essentially non oscillatory interpolation (ENO) for degrees of freedom without special boundary conditions and fourier, sine and cosine interpolation for angular or other degrees of freedom. The program can handle combined modes (combined in \textit{potfit}) when the combination is up to 2D type, as it is capable of 1D and 2D interpolations. If the original \textit{natpot} has been obtained from a sparse primitive grid, it may be a good idea to \textit{potfit} it using as many natural potentials as grid points in each degree of freedom, in case it is computationally feasible. In this way, the resulting
natpot is exact on the original grid points. What follows is the already described interpolation to a suitable grid using chnpot.

12.4 Advanced topic: Manipulating potentials with the projection program

While it is fairly straightforward to include program code for new potential energy surfaces into the MCTDH program package, it is sometimes desirable to use an existing PES routine to generate a related PES. Examples for this would be:

- Models with reduced dimensionality, where you need an effective potential which is derived from the full dimensional potential by averaging over some degree(s) of freedom.
- Series expansion of a PES, e.g. multipole expansions or Fourier transforms of angular degrees of freedom.

These tasks can be accomplished with the projection program. Basically, this program takes the potential \( V(q_1, \ldots, q_f) \) from an existing PES routine and “projects out” some degrees of freedom (say, \( q_1, \ldots, q_p \)) by integrating over them with projection functions \( \chi_\kappa(q_\kappa) \) to yield an \( (f-p) \)-dimensional projected potential:

\[
V_{proj}(q_{p+1}, \ldots, q_f) = \int dq_1 \cdots dq_p \chi_1(q_1) \cdots \chi_p(q_p) V(q_1, \ldots, q_f)
\]  

(12.2)

Numerically this is done by evaluating the PES and the projection functions on the DVR grid points \( x_{\alpha_\kappa}^{(\kappa)} \) and employing the DVR weights \( w_{\alpha_\kappa}^{(\kappa)} \):

\[
V_{proj}(x_{\alpha_{p+1}}^{(p+1)}, \ldots, x_{\alpha_f}^{(f)}) = \sum_{\alpha_1} \cdots \sum_{\alpha_p} \sum_{\alpha_1} w_{\alpha_1}^{(1)} \chi_1(x_{\alpha_1}^{(1)}) \cdots w_{\alpha_p}^{(p)} \chi_p(x_{\alpha_p}^{(p)}) V(x_{\alpha_1}^{(1)}, \ldots, x_{\alpha_f}^{(f)})
\]  

(12.3)

projection will store the resulting \( V_{proj} \) as a vpot file which must then be processed in a subsequent potfit run.

During the projection run, the first \( p \) degrees of freedom are removed, and so one is absolutely free in choosing the DVR basis (i.e. basis type as well as basis parameters and basis size) for these DOFs. For the \( f-p \) remaining DOFs however, the DVR basis must match the one for the subsequent potfit run.

As one is free to choose the DVR basis for the DOFs on which one projects, one can enhance the accuracy of the integration scheme (12.3) by choosing it according to the projection functions. E.g. if the projection function is a Legendre polynomial, one should use a Legendre DVR, as this will turn the integration into a Gauss quadrature. For some of the available projection functions, the HTML documentation gives hints on which DVR to choose.

12.4.1 Input and output files

As both projection and potfit take a PES on the full grid as primary input, their input files are quite similar. An example for a projection input file is given in example 12.3. (You can find a full version with more extensive comments in $MCTDH_DIR/pinputs/bmkpe_proj.inp$.)
12.4 Manipulating potentials with the projection program

**Example 12.3:** A projection input file for the BMKP surface for \((H_2)_2\) (excerpt).

The RUN-SECTION only specifies the name directory and which output files to be opened. The OPERATOR-SECTION is the same as in potfit; it specifies the PES to be used and defines cuts to remove large potential values.

The PRIMITIVE-BASIS-SECTION has the same structure as in mctdh and potfit. It defines the grid points of the product grid on which (12.3) is evaluated. As mentioned above, for the DOFs on which one projects one is free to use any basis definition. For the remaining DOFs, one must use the same basis definitions as for the subsequent potfit run (which in turn must be the same as for the following mctdh run, unless one uses chnpot in between).

The PROJECTION-SECTION contains the definitions of the projector functions \(\chi_\kappa(q)\). During a single projection run, it is possible to use several different sets of projection functions. One such set of functions is defined in one PROJECTOR section. After the
PROJECTOR keyword one must give an arbitrary string which is used as a label for the projected potential (in example 12.3 this would be the labels “000”, “200”, and “222”). The definition of the individual projector functions is done by first stating the modelabel of the DOF on which to project (here e.g. “AL”), and then giving the type and parameters of the projection function (here e.g. “leg 2 0”, which stands for the Legendre polynomial $P_2$). Please see the HTML documentation for which projector function types are available and what parameters they take.

For each defined projector, projection produces a vpot file which contains the projected potential on the product grid of the remaining DOFs. The name of this file depends on the given projector label, e.g. in case of the “200” projector the corresponding file will be called vpot.200. To use this file in the subsequent potfit run, one must read it with the readvpot keyword (in the RUN-SECTION) and also specify pes=none (in the OPERATOR-SECTION). Example input files for potfit runs which use projection output can be found in $MCTDH_DIR/pinputs/bmkpe_fit*.inp$.

In the case where the projected potentials $V_{proj}^{(i)}$ (where $i$ numbers the different sets of projector functions) constitute a series expansion of $V$, it is worthwhile to re-assemble the projected potentials into another full-dimensional potential $\tilde{V}$ which can then be compared to the original potential $V$. Thus one can check whether enough terms of the series expansion have been taken into account to faithfully represent the original potential. To this end, let $\chi^{(i)}_\kappa$ be the projector function for the $\kappa$-th DOF in the $i$-th set, and let $\tilde{\chi}^{(i)}_\kappa$ be the corresponding complementary projector function such that

$$\int dq_\kappa \chi^{(i)}_\kappa(q_\kappa) \tilde{\chi}^{(i)}_\kappa(q_\kappa) = 1$$

(12.4)

If we assume that the projector sets $i$ and $j$ are orthonormal, i.e.

$$\prod_{\kappa=1}^P dq_\kappa \chi^{(i)}_\kappa(q_\kappa) \tilde{\chi}^{(j)}_\kappa(q_\kappa) = \delta_{ij}$$

(12.5)

then we can define the re-assembled potential

$$\tilde{V}(q_1, \ldots, q_f) = \sum_i \chi^{(i)}_1(q_1) \cdots \chi^{(i)}_p(q_p) V_{proj}^{(i)}(q_{p+1}, \ldots, q_f)$$

(12.6)

An error measure that depends on the remaining coordinates and which has the unit of energy is:

$$\Delta V(q_{p+1}, \ldots, q_f) = \left( \frac{\int dq_1 \cdots dq_p |V(q_1, \ldots, q_f) - \tilde{V}(q_1, \ldots, q_f)|^2}{\int dq_1 \cdots dq_p} \right)^{1/2}$$

(12.7)

If you give the error keyword in the PROJECTION-SECTION, this error measure will be calculated and stored in another vpot file, named projerr.vpot. This can then be inspected with the showpot program.

Finally, the log file lists some brief information about the projected potentials, and (in case that error was requested) the mean, minimum and maximum values of $\Delta V$. 
12.4.2 Generating a Fourier-transformed potential

In the case of a four-atomic system which is described in Jacobi coordinates (see Fig. 12.2), it is of technical advantage for the MCTDH package to replace the relative torsional angle \( \varphi \) by two torsional angles \( \varphi_1 \) and \( \varphi_2 \) such that \( \varphi = \varphi_1 - \varphi_2 \) and then describe the system in terms of their conjugate momenta \( k_1 \) and \( k_2 \). (For more details see [31] or [32].) In this case the transition from \( \varphi_{1,2} \) to \( k_{1,2} \) is simply done by a Fourier transform (in the following we will abbreviate the set of coordinates \( R, r_1, r_2, \theta_1, \theta_2 \) simply by \( Q \)):

\[
\tilde{\Psi}(Q, k_1, k_2) = \frac{1}{(2\pi)^2} \int_0^{2\pi} d\varphi_1 e^{-i k_1 \varphi_1} \int_0^{2\pi} d\varphi_2 e^{-i k_2 \varphi_2} \Psi(Q, \varphi_1 - \varphi_2) \tag{12.8}
\]

Likewise, we Fourier-transform the potential \( V(Q, \varphi) \):

\[
\tilde{V}_{\Omega}(Q) = \frac{1}{2\pi} \int_0^{2\pi} d\varphi e^{-i \Omega \varphi} V(Q, \varphi) \tag{12.9}
\]

or vice versa:

\[
V(Q, \varphi) = \sum_{\Omega=-\infty}^{+\infty} e^{i\Omega \varphi} \tilde{V}_{\Omega}(Q) \tag{12.10}
\]

Then it is straightforward to see that the action of the potential operator \( \hat{V} \) on the wavefunction in terms of \( k_{1,2} \) is given by

\[
(\hat{V}\Psi)(Q, k_1, k_2) = \sum_{\Omega=-\infty}^{+\infty} \tilde{V}_{\Omega}(Q) \tilde{\Psi}(Q, k_1 - \Omega, k_2 + \Omega) \tag{12.11}
\]

Since the potential \( V \) is real and often symmetric, \( V(Q, \varphi) = V(Q, -\varphi) \), the Fourier-transformed potential is also symmetric, \( \tilde{V}_{\Omega}(Q) = \tilde{V}_{-\Omega}(Q) \), and (12.9) simplifies to

\[
\tilde{V}_{\Omega}(Q) = \frac{1}{2\pi} \int_0^{2\pi} d\varphi \cos(\Omega \varphi) V(Q, \varphi) \tag{12.12}
\]

The latter form can easily be evaluated with the projection program. A PROJECTION-SECTION of the corresponding input file would look like this:

![Figure 12.2: Jacobi coordinates for a 4-atomic system](image-url)
126 12 Using the Potfit program

PROJECTION-SECTION
PROJECTOR K0
PHI cos 0
end-projector
PROJECTOR K1
PHI cos 1
end-projector
PROJECTOR K2
PHI cos 2
end-projector
# ...
error
end-projection-section

where PHI is the modelabel of the torsional coordinate in question. Take note that the definition of the cos projector function already includes the factor 1/2π! The usage of the error keyword is recommended here, as it will calculate the error measure described in the last section, so one can check whether one has calculated enough Fourier components to faithfully represent the original potential (look at the log file).

Once this projection run is finished, you will find the files vpot_K0, vpot_K1, etc., which contain the individual Fourier components \( \tilde{V}_\Omega \). As these are defined on the full grid of the remaining coordinates \( Q \), one must use potfit to bring them into the product form required by MCTDH. It will be necessary to write separate input files to fit the separate Fourier components, but these input files will look very much alike. When writing the input files, note the following:

- in the RUN-SECTION, use readvpot to read the vpot_Kx files created by projection
- in the OPERATOR-SECTION, use pes=none
- in the PRIMITIVE-BASIS-SECTION, use exactly the basis definitions from the projection input file, but only for the remaining coordinates \( Q \); omit all the coordinates that were projected out.

After the potfit runs are complete, you can optionally change the primitive basis by using chnpot.

12.4.3 Using a Fourier-transformed potential in MCTDH

The last section described how to generate the Fourier components \( \tilde{V}_\Omega \) in a form usable by MCTDH; however this is only one ingredient of (12.11). The other is the shifting of the \( k \) coordinates, \( \tilde{\Psi}(Q, k_1, k_2) \to \tilde{\Psi}(Q, k_1 - \Omega, k_2 + \Omega) \). In MCTDH this can be accomplished with the simple shift operator, which we will denote here formally as \( \hat{S}_q^\Delta \) (where \( q \) is the DOF on which it operates and \( \Delta \) is the amount by which the DOF is shifted):

\[
(\hat{S}_q^\Delta \tilde{\Psi})(..., q, ...) = \tilde{\Psi}(..., q - \Delta, ...)
\] (12.13)

In this notation, (12.11) becomes (recall that \( \tilde{V}_\Omega = \tilde{V} - \Omega \)):

\[
(\hat{V}\tilde{\Psi})(Q, k_1, k_2) = \tilde{V}_0(Q) \tilde{\Psi}(Q, k_1, k_2) + \sum_{\Omega=1}^\infty \tilde{V}_\Omega(Q) \left[ (\hat{S}_{k_1}^{\Omega} \hat{S}_{k_2}^{-\Omega} \tilde{\Psi})(Q, k_1, k_2) + (\hat{S}_{k_2}^{-\Omega} \hat{S}_{k_1}^{\Omega} \tilde{\Psi})(Q, k_1, k_2) \right]
\] (12.14)
12.4 Manipulating potentials with the projection program

HAMILTONIAN-SECTION

<table>
<thead>
<tr>
<th>modes</th>
<th>R</th>
<th>r1</th>
<th>r2</th>
<th>th1</th>
<th>k1</th>
<th>th2</th>
<th>k2</th>
</tr>
</thead>
<tbody>
<tr>
<td># ... (kinetic energy omitted)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1&amp;2&amp;3&amp;4&amp;6</td>
<td>V0</td>
<td>5 1</td>
<td>7 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1&amp;2&amp;3&amp;4&amp;6</td>
<td>V1</td>
<td>5 kp1</td>
<td>7 km1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1&amp;2&amp;3&amp;4&amp;6</td>
<td>V1</td>
<td>5 km1</td>
<td>7 kp1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1&amp;2&amp;3&amp;4&amp;6</td>
<td>V2</td>
<td>5 kp2</td>
<td>7 km2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1&amp;2&amp;3&amp;4&amp;6</td>
<td>V2</td>
<td>5 km2</td>
<td>7 kp2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

end-hamiltonian-section

LABELS-SECTION

V0 = natpot{fitK0}
V1 = natpot{fitK1}
V2 = natpot{fitK2}
kp1 = shift[1]
km1 = shift[-1]
kp2 = shift[2]
km2 = shift[-2]

end-section

Example 12.4: An operator file showing the use a Fourier-transformed potential (excerpt). Only Fourier components up to $\Omega=2$ are included.

or even more formally on the operator level:

$$\hat{V} = \hat{V}_0 + \sum_{\Omega=1}^{\infty} \left( \hat{V}_{\Omega}\hat{S}_{k_1}\hat{S}_{k_2} - \hat{V}_{\Omega}\hat{S}_{k_2}\hat{S}_{k_1} \right)$$  (12.15)

This form can be easily translated into an MCTDH operator file. (Of course one will truncate the series at a certain $\Omega_{\text{max}}$.) Parts of the corresponding HAMILTONIAN- and LABELS-SECTIONs are shown in example 12.4. A few explanations are in order:

- During the following MCTDH propagation, we will be using the two-dimensional KLeg DVR for the combined coordinates $(\theta_1, k_1)$ and $(\theta_2, k_2)$. (The use of this DVR is part of the technical reason why we did the Fourier transform in the first place.) This makes it necessary to order the coordinates differently than in the above theoretical considerations. Namely, $k_2$ can not directly follow $k_1$ but the $k_j$ must be paired with the $\theta_j$.

- This different coordinate ordering makes it necessary to use the slightly cumbersome $123&4&6$ syntax for the potfitted Fourier components $V0, V1, V2$. This tells the program to apply them to the set of DOFs $(1, 2, 3, 4, 6)$, which corresponds to $(R, r_1, r_2, \theta_1, \theta_2)$. (Also see section 6.13.)

- Here we assume that the natpot files of the potfitted Fourier components are located in the directories fitK0 etc. That is, fitK0 should contain the potfit of vpotK0, fitK1 that of vpotK1, etc.

- The shift operators must be put into the LABELS-SECTION and then referenced by their labels (kp1, km1, etc.) because of a syntactical restriction of MCTDH: it is not allowed to use operators which take an argument directly in the HAMILTONIAN-SECTION.
This completes the setup of the operator file for a Fourier-transformed potential.

12.5 Downsizing previous *potfits*: the cutnpot and rdnpot functions

Under certain circumstances, it may be necessary to make use of previous *potfitted* potentials, *i.e.* natpot files. Possibly, it may be convenient to reduce the number of single particle potentials and the corresponding contracted coefficients. For instance, this may be useful to investigate the quality of the potential expansion (*e.g.* the fitting error) upon reduction of the expansion coefficients or, simply, in order to decrease the cost of the energy evaluation. For such a purpose, the cutnpot and rdnpot functions have been developed. It should be highlighted that they play a major role in the mgpf program (under development).

The process can be summarized in two steps: (i) elimination of unwanted single particle potentials and the corresponding contracted coefficients from a previously existing natpot file; and (ii) reading of the newly generated (reduced) natpot file for the evaluation of error measurement. The first step is accomplished by means of the cutnpot function. This is invoked by including the cutnpot keyword in the RUN-SECTION of the potfit input file. The NATPOT-BASIS-SECTION should contain the desired (smaller) values of the expansion coefficients while preserving the definitions of the contracted mode and the particles. The natpot file to be reduced (cut)\(^1\) should either be placed in the working directory as natpot1 or under a different name (or location) in which case the name (with absolute path) should be provided as: cutnpot=path. Then potfit will produce a new (shrinked) natpot file within the same directory. Finally, the subsequent measurement of the fitting error (for sufficiently small primitive grids) is obtained by using the rdnpot function which is invoked by adding the RUN-SECTION keyword rdnpot and running again potfit.

As previously indicated, for the sake of consistency, the DVR definitions, the contracted mode, and the mode combination scheme, if any, must remain unaltered during the whole process. Hence, if a new particle scheme is needed, then a new potfit or a new mgpf calculation has to be performed.

The cutnpot and rdnpot keywords appear in the RUN-SECTION, otherwise the input is the standard one.

\(^1\)Note that cutnpot yields the same results as a normal potfit with the reduced number of single particle potentials, provided no correlated weights are used.
Chapter 13

Using the Monte-Carlo Potfit program

13.1 Monte-Carlo Potfit

The potfit program introduced in the previous chapter is a powerful tool to convert a given potential into sum-of-product form. For low dimensional problems, i.e., up to 6D (with not too many grid points), it is a very stable and well behaved algorithm. For larger dimensions, however, potfit runs into a serious bottleneck: potfit requires multiple integrations over the complete primitive grid. This not only means multiple runs over a large number of grid points, but also that the potential must be computed and stored on all these grid points.

Monte-Carlo Potfit (or mcpotfit) seeks to circumvent the complete integrals by just using a (random) subset of the complete set of primitive points and to extract all necessary information needed to produce a natpot file from this limited set. In this respect mcpotfit has the advantage that much larger grids can be treated as the number of grid points that enter the calculation can be drastically reduced, often by multiple orders of magnitude. Also correlated weights can be easily incorporated by means of the distribution of sampling points such that relevant regions of the potential can be treated with increased accuracy. For a detailed discussion of the theory behind mcpotfit and a discussion of importance sampling see Ref. [33].

mcpotfit relies on the existence of a potential energy routine that can be compiled into the operator library. If this is for some reason not possible or wanted one can also implement the routine in the file $MCTDH_DIR/source/mcpotfit/userpot.F90. It already contains a skeleton implementation that defines the routine interface.

The mcpotfit program is organized in tasks that lead to generation of the natpot file. They are consecutively executed. The tasks are specified by means of an input file which is organized in sections, analogue to the traditional potfit. The tasks are

1. Create or read the sampling points.
2. Create or read the reduced densities and calculate the SPP
3. Calculate the coefficients.
4. Estimate the fit error.
The organization in tasks allows re-using earlier calculated parts in later calculations, for instance re-using reduced densities or SPP with a different set of sampling points for calculating the coefficients, or to use a different number of SPP. Which tasks are skipped is mainly controlled in the \texttt{SAMPLING-SECTION}. See below.

### 13.2 Compiling

During installation a basic version of \texttt{mcpotfit} is readily compiled. It lacks, however, any parallelization. Other than \texttt{mctdh} and other programs of the package, \texttt{mcpotfit} is written in Fortran 90 and uses OpenMP instead of Posix threads as its shared memory parallelization method. To compile with shared memory parallelization the \texttt{-P} option has to be used:

```
compile -P mcpotfit
```

This produces the executable \texttt{mcpotfit85P}. As for \texttt{mctdh} MPI support is added with the \texttt{-m} option, which can, of course, be combined with the \texttt{-P} option.

In \texttt{mcpotfit} a number of solvers are implemented to solve for the expansion coefficients, some of which use standard linear algebra tools, i.e., LAPACK or ScALAPACK routines. For large numbers of SPP using ScALAPACK instead of LAPACK is almost always unavoidable. If ScALAPACK is to be used, \texttt{mcpotfit} needs to be compiled with ScALAPACK support. Since the MCTDH package is not shipped with any ScALAPACK libraries, the ScALAPACK libraries must already be available on the system - either self compiled or installed from the package manager. If ScALAPACK is installed through the systems package manager it is usually sufficient to use the \texttt{-S} compile option. This may however depend on the compiler that is used for compilation.

In case of a self compiled ScALAPACK or if the library is in some non-standard location one usually also has to specify the linker search path. This can either be done by modifying system environment variables or, more easily, by specifying the search path directly in the compile.cnf file: Open the file \texttt{${MCTDH\_DIR/install/compile.cnf}} with your favorite editor and search for the compiler label you intent to use for compilation. Note, that this must be a compiler with MPI support. Add or edit the variables controlling ScALAPACK support, for instance

```
MCTDH\_SCALAPACK\_FLAGS="-DSCALAPACK"
MCTDH\_SCALAPACK\_LIBS="-L/path/to/library -lscalapack"
```

The \texttt{-DSCALAPACK} flag here includes the ScALAPACK function calls in the source code. If it is not present, regular LAPACK is called instead. Now compile with the \texttt{-S} option.

Note, that sometimes ScALAPACK comes in two separate libraries \texttt{libscalapack.a} and \texttt{libblacs.a}, separating numerics and communication. In this case one needs to include the BLACS library as well, of course, for instance with

```
MCTDH\_SCALAPACK\_FLAGS="-DSCALAPACK"
MCTDH\_SCALAPACK\_LIBS="-L/path/to/libraries -lscalapack -lblacs"
```

### 13.3 Selecting the sampling method

At present \texttt{mcpotfit} supports generation of uniformly distributed sampling points as well as Boltzmann distributed sampling points. Furthermore samplings can be read in from file.
13.3 Selecting the sampling method

Example 13.1: A SAMPLING-SECTION in MC-Potfit by which different sources of sampling points are assigned to a task: 1) 50000 points from a Metropolis trajectory for calculating the SPP, 2) another Metropolis sampling with 50000 points plus additional 5000 points of uniform distribution for calculating the coefficients, and 3) samplings read from file for testing the fit. The numbers after \texttt{metropolis, 50000} mean: equilibrate for 1000 steps before recording sampling points, use only every 10th Metropolis step as a sampling point, and assume the temperature $k_B T$ to be 3000.0 cm$^{-1}$

13.3.1 Sampling methods

The samplings are specified in a SAMPLING-SECTION. At present there is only a distinction between the sets of sampling points used for generating the reduced densities and hence SPP, for the coefficients, and for the error estimation. This means in particular, that the SPP of the single modes are all calculated with the same set of sampling points, i.e., also the same number of sampling points. This can probably be changed in the future as for different modes a different number of sampling points might be sufficient.

Multiple sources of sampling points can be chosen per task, for instance multiple Metropolis sets with different temperature or additional special points stored in a file. To use multiple sets for one task one simply specifies multiple sets in the SAMPLING-SECTION. The sets are internally merged into one large set that is used to perform the calculations.

The format of the trajectory files which can be read in is rather simple: each sampling point is represented as an integer array in ASCII format, one sampling point per line. The $n$-th integer entry in the $n$-th line, $i_{\kappa}^{(n)}$, corresponds to the $i$-th DVR point of the $\kappa$-th DOF in the primitive basis and belongs to the $n$-th sampling point. One can therefore simply create multiple sampling files and pipe them into a single file for later use or specify multiple files to read in in the SAMPLING-SECTION. When the keyword \texttt{sampling-only} is set in the RUN-SECTION, \texttt{mcpotfit} will only create and write the samplings specified in the SAMPLING-SECTION and exit thereafter.

Example 13.1 illustrates the structure of the SAMPLING-SECTION: The sampling methods are assigned to a task with the label \texttt{sampling-<task>}, where \texttt{<task>} is one of \texttt{spp}, \texttt{coeff}, or \texttt{test}. The task can be omitted in which case the same sampling method is assigned to all tasks. In this case only the method is the same, but still different sets of sampling points are created (unless they are read from file, of course).

Sometimes one may want to use really the same set of points, not only the same methods, for the coefficients and the SPP. In this case one can specify the sources for one of the tasks (for instance the SPP) and then assign this sampling to the other task. Example 13.2 illustrates this.

As shown in Example 13.1, three different sources of sampling points are currently implemented: a simple random number generator that produces uniform sampling, and a Metropolis algorithm that produces a Boltzmann distributed set of sampling points, i.e., distributed according to the weight $\exp\left(-V(q)/k_B T\right)$. Additionally sampling points can be read from
Example 13.2: A SAMPLING-SECTION in MC-Potfit by which different sources of sampling points are assigned to a task: 1) Metropolis sampling with 50000 points plus additional 50000 points from a Metropolis trajectory with a different temperature for calculating the SPP, 2) The same points as used for the SPP are also used for the coefficients, and 3) samplings read from file for testing the fit. The numbers after `metropolis, 50000` mean: equilibrate for 1000 steps before recording sampling points, use only every 10th Metropolis step as a sampling point, and assume the temperature $k_B T$ to be 1000.0 cm$^{-1}$ and 1000.0 cm$^{-1}$, respectively.

file. Also the complete grid can be used but in this case one might better switch to the traditional potfit as it is implemented here with very limited features and more for testing purposes. When using complete sampling, only the generation of the vpot file, i.e., the initial sampling of the potential, is MPI-parallel. All other operations are only shared memory parallel. Table 13.1 illustrates the parameters that are available for the various sampling methods.

13.3.2 Remark on the Metropolis algorithm

While reading points from index files and creating a uniform distribution is computationally relatively cheap, this is not the case for the Metropolis algorithm. The algorithm is modified to only select neighboring DVR points as candidates for new positions. This can sometimes lead to a trapping of the random walker (`mcpfotfit` will detect this and exit with the suggestion to increase the temperature) but it may also need a number of tries to calculate the next step. This involves multiple, depending on the parameters sometimes even many, evaluations of the PES routine before a sampling point is calculated. If started in MPI mode, `mcpfotfit` will therefore start one random walker in each MPI rank (shared memory parallelization is not yet supported within the Metropolis algorithm). A single walker will therefore only create a fraction of the total trajectory. However, all walkers will do a `burn-in`, i.e., propagate a number of steps to equilibrate (cf. $I_2$ parameter of the metropolis algorithm in Table 13.1) before sampling points are recorded. The `burn-in` is necessary because the initial position of the random walkers is chosen at random on the complete grid (and of course differently in each MPI rank). The Parameter $I_3$ of the metropolis algorithm can be used to reduce the correlation length of the trajectory of a single walker.

13.3.3 Choosing a sampling method

The method that should be chosen for generating the sampling depends on the fitting goal. For a globally optimal fit one should, of course, use uniform sampling as it resembles the traditional potfit without any weighting. Very often a globally optimal fit is not intended but increased accuracy in the low energy regions of the potential is required, i.e., in the regions where the wavefunction resides. This can be achieved with Metropolis sampling.
Using the Metropolis algorithm above, however, requires choosing appropriate parameters. One critical decision is the temperature which defines the width of the Boltzmann distribution. It should be chosen such that its width is larger than the region where the wavefunction will reside. But it should also not be too high to keep the area of increased accuracy as small as possible. A temperature with an energy equivalent \( E_{\text{max}} + \frac{1}{2}E_0 \) seems to be a reasonable choice to start with for many systems. Here \( E_{\text{max}} \) is the energy of the largest involved excitation energy of the system Hamiltonian, \( E_0 \) is the zero point energy, and \( f \) is the number of DOF. The remaining parameters concern the number of sampling points, burn-in and step selection. Checks for the number of sampling points is discussed in more detail in Subsection 13.8 while \( I_3 \) (given a reasonably large number of points and/or walkers) is of less importance.

If Metropolis sampling has been used for calculating the reduced densities it proved advantageous to use a metropolis trajectory with the same temperature for the coefficients as well. In this case, the SPP are optimal for the same region of the potential for which also the coefficients are calculated.

Often, when Metropolis sampling is used with a low temperature, some high-energy regions of the potential remain unsampled. This can lead to a situation where some parts of the potential are fitted with high accuracy while others (in regions without sampling points) deviate substantially from the original potential. This behavior can be cured by adding a small fraction of uniform sampling to the trajectories. Also a small fraction of Metropolis sampling with a high temperature may serve this purpose.

Similarly, some points might be of increased importance for an accurate fit but may not be part of the set of random sampling points. This can be, for instance, points in the vicinity of a transition state where the potential varies rapidly. These points need to be added manually to a trajectory file at present.
13.3.4 Re-using and pre-sampling trajectories

Especially Metropolis trajectories are numerically expensive to create. One might therefore wish to re-use formerly created trajectories or create a number of trajectories before one starts to perform any calculations to create a PES fit.

In a regular run, any newly created trajectories are stored in the name directory with file-names starting with dvrindex-. Any dvrindex- can be read-in with the readidx{\text{/path/to/dvrindex-file}} command in the SAMPLING-SECTION. With this one easily re-use these trajectories.

One can also start a number of runs of the program just to create dvrindex- files. To this end one set the keyword sampling-only in the RUN-SECTION. If sampling-only is set, then mcpotfit will only create the dvrindex- files and exit thereafter.

13.3.5 Re-using densities or SPP

Once the reduced densities and SPP are generated they can be re-used in subsequent calculations without the need to re-calculate them. This is done by removing or commenting-out the sampling method for generating the SPP. The program will then assume that the respective SPP files are present in the name directory and that they can be read in. The program will first look for the files evecs\_mode\_i where \( i \) is the mode number. If these files are not present (one needs to set save-evecs in the RUN-SECTION to store the SPP, by default only densities are stored) the program will attempt to read the files density\_mode\_i. If neither are present mcpotfit will produce an error message. Reading SPP instead of densities also works if the numbers of SPP are changed between two runs as also unused SPP are stored in the files.

Note also, that the format for storing the densities and SPP files is ASCII by default. It can be set to binary in the RUN-SECTION with the keyword density=binary. The ASCII format allows inspecting the densities with various plotting tools while binary is of advantage if one plans to re-use the densities in later calculations.

13.4 Solving for the coefficients

Within mcpotfit a number of solvers for the coefficients are available. In particular these are LAPACK and ScaLAPACK based solvers and a conjugate gradients algorithm. Since solving for the coefficients is usually the most demanding part, an efficient parallelization is essential. While the conjugate gradients algorithm is implemented as a module within the mcpotfit source and can use natively MPI and OpenMP parallelization, this is not the case for ScaLAPACK based solver. To use ScaLAPACK support one additionally needs to compile with ScaLAPACK support, otherwise the program will use standard LAPACK routines as a fall-back (cf. Section 13.2).

The method that is used for solving for the coefficients is specified with the invert-method keyword in the RUN-SECTION. The available solvers are outlined in Table 13.2.

While invert-method = direct and invert-method = eigen require explicit building the SPP-overlap matrix \( \Omega^T \Omega \), which is numerically very costly, this is not the case for invert-method = pseudo. The latter, however, operates directly on \( \Omega \), which might be very large in which case the method will become very slow. Within the former
13.4 Solving for the coefficients

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Parameters</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct</td>
<td>–</td>
<td>(Sca)LAPACK solver (P)DPOSV using Cholesky decomposition</td>
</tr>
<tr>
<td>eigen</td>
<td>regularization</td>
<td>(Sca)LAPACK solver (P)DSYEV using eigenvalue decomposition. Optional regularization.</td>
</tr>
<tr>
<td>pseudo</td>
<td>–</td>
<td>(Sca)LAPACK solver (P)DGELS calculates pseudo-inverse of $\Omega$ using QR-decomposition.</td>
</tr>
<tr>
<td>conjgrad</td>
<td>–</td>
<td>(Block-) conjugate gradients.</td>
</tr>
</tbody>
</table>

Table 13.2: Solvers for the coefficients.

Two $\Omega$ can be calculated on the fly as needed to reduce memory usage. See Section 13.5 for benefits and downsides of these variants.

Note that while invert-method = direct and invert-method = conjgrad both rely on a truly positive definite SPP-overlap matrix $\Omega^T \Omega$, this condition is relaxed for the remaining two solvers which can handle singular overlap matrices. (Note that the pseudo solver, however, is usually slow as the $\Omega$ matrix is usually very large.)

In practice, the direct solver showed the best performance in the vast majority of cases. For uniform sampling, sometimes the conjgrad solver can outperform the direct solver if many sampling points are used. A very nice feature of the eigen solver is that it one obtains the eigenvalues of the SPP overlap matrix which are stored in the file eigenvalues in the name directory. In most cases the pseudo solver showed the poorest performance and should be considered experimental.

13.4.1 Using conjugate gradients to solve for the coefficients

Some special rules apply for the conjgrad solver. The conjugate gradients method is implemented in two variants, one for the case where no contracted mode is present, the other for the opposite case. If mode contraction is omitted, the set of equations to solve has a single right hand side and hence a single solution vector. If a contacted mode is present, the situation changes somewhat: in this case there exist multiple right hand sides and a block variant of conjugate gradients is used. The benefit of multiple right hand sites is that the SPP-overlap matrix $\Omega^T \Omega$ is much smaller and hence less costly to invert.

Conjugate gradients usually converges rapidly when uniform sampling is used as then the SPP-overlap matrix is similar to a unit matrix. The situation changes substantially when other sampling distributions are used. In this case, conjugate gradients converges slowly. Within mcpotfit a number of preconditioners are available that can speed up the convergence. They can be added with the keyword cg-precon in the RUN-SECTION. The preconditioners are outlined in Table 13.3.

In general, the Jacobi preconditioner, diag, is the fastest to build. It only uses the diagonal of the SPP-overlap, however, it does usually not yield much speed-up of the convergence. The band preconditioner can yield quite some speed-up, however this is somewhat unpredictable as it is not necessarily positive definite. It can therefore also lead to slower convergence than without any preconditioner. Usually the sparse and block preconditioners yields the most speed-up but are also the most expensive to build and to apply. The sparse preconditioner builds a small SPP-overlap matrix from a reduced set of SPP and
13.5 Reducing memory consumption

By default `mcpotfit` will build the so-called $\Omega$-matrix (cf. Ref. [33]) for calculating the coefficients. The $\Omega$-matrix contains all configurations evaluated at all sampling points. It can therefore become very large. For MPI calculations it is distributed among the various MPI-ranks in equal proportions. Building the $\Omega$-matrix once is very cheap in terms of CPU time and if it exists it can be used to efficiently calculate the SPP overlap matrix or to operate on the residual vector within the conjugate gradients algorithm.

However, the $\Omega$-matrix may be so large that it exceeds the memory capacity of the compute nodes. To this end one can set the flag `no-omega` in the `RUN-SECTION`. If set, the matrix elements are (repeatedly) calculated on the fly when needed. This will of course have quite some speed impact but will also considerably reduce the memory costs.

Within conjugate gradients, the $\Omega$-matrix is even allocated twice by default: once in the original order and once transposed to reduce cache misses. Of course, this speeds up calculation at the cost of a lot of memory. In this case one can set `no-omega-t` in the `RUN-SECTION` such that the $\Omega$-matrix is only allocated once. This will have a moderate speed impact but still reduce memory quite a bit. If additionally `no-omega` is set, also within conjugate gradients, the matrix elements are (repeatedly) calculated on the fly. This will have a considerable speed impact.
13.6 Restoring molecular symmetries

In the traditional potfit, molecular symmetries are usually preserved automatically. This is because one usually defines the primitive grids symmetrically and integrals over (sub-)spaces do not lead to any symmetry breaking. In mcpotfit this is no longer the case due to the random choice of the selection of sampling points. mcpotfit will therefore in general not produce symmetric potential fits.

One can, however, artificially re-symmetrize the the potential fits. This is done in two steps within the program: first, symmetry adapted SPP are obtained. Second, coefficients are calculated and subsequently symmetrized. The crucial point to be able to do these two steps is the knowledge about the molecular symmetries. They can be provided to the program within an optional SYMMETRY-SECTION.

The SYMMETRY-SECTION contains a table of symmetry operations on the primitive grid. Cf. Example 13.3 to see the structure of the symmetry table. It consists of a number of columns, each of which with a label in the first row, followed by signed DOF labels, expressions in curly brackets or the usersym keyword (see below).

The first column of the symmetry table is special: it contains the E symmetry operation, i.e., the identity. In this column all coordinate labels from the PBASIS-SECTION must be given. They can be in a different order then within the PBASIS-SECTION which allows re-arranging the DOF labels, for instance according to mode combinations for clarity. The first column serves as a reference for all further symmetries in the remaining columns in which the actual symmetry operations are implemented.

### 13.6.1 Simple symmetries

Simple symmetries are grid based operations, i.e., here one operates directly on the DVR-Grid points of the single DOF. Any coordinate values of the grid-points are ignored. Allowed operations are DOF-permutations (with respect to E) and reversing the primitive grid.
DOF permutations are interpreted as an interchange of the primitive grids. For instance, consider a two dimensional system with potential \(V(x, y)\) which is expressed on the product grid as \(V_{i_1,i_2} = V(x_{i_1}, y_{i_2})\). Then the input

```
SYMMETRY-SECTION
E SXY1
--------------
x y
y x
end-symmetry-section
```

will lead to a fit which obeys the symmetry \(V_{PF_{i_1,i_2}} = V_{PF_{i_2,i_1}}\). Note, that this is really a grid-based symmetrization as only grid point indices are considered and the values of the coordinates \(x\) and \(y\) are ignored. Additionally the DOF labels may bear a sign which means that the reverse primitive grid index is used instead. Therefore, the input

```
SYMMETRY-SECTION
E SXY2
--------------
x y
y -x
end-symmetry-section
```

will lead to a fit which obeys the symmetry \(V_{PF_{i_1,i_2}} = V_{PF_{i_2,N_1-i_1+1}}\).

Defining grid-based symmetries within \texttt{mcpotfit} is subject to a number of restrictions:

1. Interchanging DOFs must have the same number of grid points and the same DVRs
2. DOF permutations must either stay within one combined mode or complete modes must be swapped (with arbitrary mode-internal permutations)
3. Permuting (combined) modes must have the same number of grid points and the same DVRs

Note, that at present, \texttt{mcpotfit} does not check if the DVRs of interchanging modes are compatible. It just checks if the numbers of grid points of the two grids are identical.

### 13.6.2 Coordinate-based expressions

Sometimes it is more convenient to use arithmetic expressions (instead of counting grid points) to implement the symmetry operations. This can be done using curly brackets. Curly brackets must contain an expression of the form \(\{\pm S \pm R\}\), where \(S\) is a DOF label and \(R\) is an optional real number or an arithmetic expression (using \(+,-,\times,\div\) of real numbers. The arithmetic expression of real numbers may also contain the symbols \(\pi\) and \(2\pi\) with obvious meanings. (Note: at present no blanks in curly brackets are supported.)

When using coordinate-based expressions, any the DVR index points are converted to coordinate values which are transformed according to the expression in curly brackets and then converted back to grid indices. In the background, coordinate-based expressions are hence still grid based operations but with a more intuitive notation. One must therefore take care that coordinate expressions again lead to primitive grid points (unless one uses extended grids, see below).

In Example 13.3 the coordinate based expressions for the DOF labels \(la\) and \(lb\) could also have been done with grid based operations that ignore the coordinate values. The real
13.6 Restoring molecular symmetries

Example 13.4: Symmetry operations with one DOF (\(\phi\)) with periodic boundary conditions.

usefulness of coordinate-based expressions actually becomes apparent when periodic boundary conditions are utilized and when extended grids are used. See below.

13.6.3 Periodic boundary conditions

When a coordinate has periodic boundary conditions there is usually not a natural point around which an inversion should take place such that grid-based inversions are unhelpful. Also, when using coordinate expressions any results outside the original periodic interval must be mapped back into it. One can therefore assign a 'periodic' flag to a list of DOFs with the periodic keyword. This is shown in Example 13.4.

13.6.4 The usersym keyword

Sometimes, more complicated symmetry operations are needed. For instance, when the transformation of one coordinate depends on another. In Example 13.3, the two coordinates \(x\) and \(y\) describe the position of the central proton of the Zundel cation. The orientation of the \(x - y\)-plane, however, is bound to the orientation of one of the water molecules. If a symmetry operation now interchanges the two waters, the \(x - y\)-plane is rotated according to the relative angle between the two water molecules \(\alpha\). The equivalent \(x\) and \(y\) values therefore depend on \(\alpha\) and need to be calculated in an external routine. In such a case the more complicated symmetry operations are labeled with the usersym keyword within the symmetry table. If usersym is present, the routine GetUserSym is called in which the symmetry operation must be implemented. It is sufficient here to only implement the transformation for the coordinates that are involved in the more complicated transformation, that is for which a special implementation is needed and for which the usersym keyword is set. All other transformations can be given using coordinate labels. If usersym is used, it must, however, then be given and implemented for all symmetry operations (except \(E\)) and for all coordinates of an involved mode. This is to make sure that possibly extended grids (see below) can be mapped for all symmetry operations.

A template of the GetUserSym routine with additional comments is located in the source code file $MCTDH_DIR/source/mcpotfit/usersym.F90. Also the routine used for the example 13.3 in Ref. [33] is given in the in file $MCTDH_DIR/source/mcpotfit/usersym.F90.zundel.

There is an important difference for these user implemented symmetry operations compared to the simple symmetries described in Subsection 13.6.1: user implemented symmetry operations are coordinate based, not grid based: the GetUserSym routine receives a vector
of coordinate values and is expected to return the correct equivalent coordinate values. However, the restrictions that apply to simple transformation also apply for user implemented symmetry operations.

13.6.5 Using intermediate extended grids

In Example 13.3, the usersym keyword is used to calculate the equivalent points of the \(x\), \(y\), and \(\alpha\) grids upon the various symmetry operations using the GetUserSym routine. While the operations \(E, C_2, \text{SigmaA}, \text{and SigmaB}\), stay on the original primitive grid and could have been implemented using coordinate labels as shown above this is not true for the operations \(S_4, S_4^{-3}, C_{2A}, \text{and C}_{2B}\). The latter four interchange the two water molecules to which the orientation of the \(x-y\)-plane is bound. In addition to coordinate permutations, the \(x-y\)-plane must therefore be rotated which leads to coordinate values outside the original primitive grid. mcpotfit can therefore calculate extended grids for these cases.

Extended grids are detected by scanning the original primitive grid mode-wise and detecting any points that lie outside the original grid. Symmetry operations that can lead to points outside the original grid are usersym operations as mentioned before, but also coordinate-based expressions. For instance in Example 13.4, it the DVR of \(\phi\) is of type EXP it will have an odd number of grid points such that none of the symmetry operations (except \(E\), of course) will lead again to a grid point.

If new points are detected, they are added as additional grid points to the respective mode. To reduce the possibility of programming errors the flag auto-extgrid has to be set in the RUN-SECTION to enable auto-generation of grid-points. Otherwise an error will be produced as soon as points outside the original grid are detected (after all, this may be unintended). This technique of auto-generating extended grids has also been used in Ref. [33].

When generating extended grids one must ensure that all symmetry operations stay on the extended grid once it has been created. This means, none of the symmetry operations operated on an extended grid again must lead to additional grid points. If this is the case (and if the symmetries are correctly implemented) then there exists another symmetry operation which is a combination of two of the already implemented ones. This symmetry must also be implemented in the SYMMETRY-SECTION. In short, one always has to define all symmetry operations of a symmetry group.

13.6.6 Symmetry checking

When a SYMMETRY-SECTION is present in the input file, two additional ASCII files are generated in the name directory: sigma and sym.log. The first file, sigma, contains the coefficient-transformation matrices (cf. Ref. [33]) that are used to symmetrize the expansion coefficients. These matrices should be diagonal, with entries +1 or -1 only (except when degeneracies are present). The file sym.log contains information about the internal representation of the symmetry operations. It can be used to debug the symmetry table. Furthermore, all symmetry operations are checked against the PES routine if they really lead to the same energies and, after the final natpot file has been symmetrized, it is tested on a number of random points if they transform correctly. If external grids have been used not all operations may stay on the primitive grid any more as the additional points have been removed again at this stage already. If a symmetry operation does not stay on the primitive grid, the nearest grid point is used instead and the "on-grid" flag is set to false in the log file.
13.7 Implementing a surface outside the MCTDH operator library

As mentioned before, \texttt{mcpotfit} relies on a potential energy surface which should usually be implemented in the MCTDH operator library. If for some reason implementation of a surface routine within the MCTDH operator library is not wanted or not possible it can be implemented as an (almost) stand-alone routine within the \texttt{mcpotfit} source code. There is a template routine in the source tree: $\texttt{MCTDH}\_\texttt{DIR/source/mcpotfit/userpot.F90}$ which can be used to this end. The function called by \texttt{mcpotfit} is \texttt{UserPot} which receives a coordinate vector with values in the same order as they are defined in the \texttt{PBASIS-SECTION} and the number of coordinates and should return the potential energy at the given point.

If shared memory parallelization is used, the function must be implemented thread-safe as it is called in parallel. If this is not possible one has to set the keyword \texttt{no-OMPpotential} in the \texttt{RUN-SECTION} in which case the calls to the potential routine are executed sequentially. Since the function \texttt{UserPot} is essentially stand-alone it must initialize itself upon the first call if necessary, for instance read-in parameters, allocate memory etc. This must also be thread-safe.

To switch to the user-implemented potential routine one needs to set the \texttt{pes = usersurf} in the \texttt{OPERATOR-SECTION}. Note that the \texttt{usersurf} keyword does not support any arguments.

13.8 Checking convergence and fit quality

In addition to the usual \texttt{potfit} error which results from the basis truncation, \texttt{mcpotfit} introduces an additional source of error which results from the Monte-Carlo integrations used throughout the program. Both aspects are interdependent, cf. Ref. [33], and need to be observed. In the following the main focus lies on convergence with respect to the Monte-Carlo integration as basis truncation errors are already discussed in the previous chapter on \texttt{potfit}.

MC-Potfit critically depends on converged SPP as well as coefficients, where the convergence depends on the number of sampling points. Unfortunately there is no obvious and simple way to estimate the needed number of sampling points beforehand. As a role of thumb one can start with about 10 times as many sampling points as primitive grid points on largest mode for the SPP and 10-100 times as many sampling points as there are configurations for the coefficients. After the first run one then needs to carefully check the convergence.

13.8.1 Convergence of the SPP

The SPP are the basis functions in which the original potential is to be expressed as good as possible. Of course, one therefore needs as good as possible basis functions, which in turn means that the SPP should be converged with respect to the number of sampling points used for their generation. Luckily, the most important SPP converge fastest, but it is essential to converge all SPP.

Checking the convergence of the SPP is done by calculating two reduced densities per mode, each one calculated from half of the total number of sampling points. Subsequently, SPP are calculated from each density separately. The SPP are then compared in two ways:
1. **By overlap of the \( n \)th SPP of the first set with the \( n \)th SPP of the second set.** The direct comparison allows to evaluate the difference between the individual SPP created from two different sets of sampling points. This however may be misleading when degenerate or close to degenerate SPP are present. In this case the individual SPP may be at different positions within the two sets and a one-to-one correspondence does not exist.

2. **By comparison of the spans of the SPP.** The spans are compared by calculating
\[
\frac{1}{n_{\kappa}} \text{tr} \left\{ P_{\kappa}^{(1)} P_{\kappa}^{(2)} \right\},
\]
where \( P_{\kappa}^{(i)} = \sum_j \left| \langle j^{(i)} | j^{(i)} \rangle \right| \) is a projector on the space spanned by the SPP of mode \( \kappa \) of set \( i \). The trace over the two projectors therefore would give unity if the spaces are identical and a result less then unity indicates that the spans are different.

Details of the above are recorded in the log file.

**13.8.2 Convergence of the Coefficients**

The convergence of the coefficients is hard to estimate directly. **mcpotfit** resorts to evaluate the \( \text{natpot} \) after the fit has been created. It is tested with use of the sampling-test trajectory in the SAMPLING-SECTION as well as the sampling-coeff trajectory. When the coefficients are calculated, they are optimized for the sampling points in the sampling-coeff trajectory and are optimal just for these points in a least squares sense while the SPP interpolate between these points. If the test trajectory is an independent trajectory it tests the quality of the interpolation between the points used to optimize the coefficients and hence tests how good coefficients are represent the complete potential. This assumes that the SPP are converged and represent the optimal interpolating functions.

Detailed statistics about the two trajectories are written to the output file. Example 13.5 shows an excerpt from an output file for the Zundel cation as used in Ref. [33]. The first part of the output file is similar as for the traditional **potfit** (cf. Example 12.2) and has been skipped in Example 13.5. Only the different error statistics are shown. The statistics that are evaluated are the mean error, i.e., the mean difference between the original potential and the fit, the RMS error (the root-mean-square difference between the original potential and the fit,
\[
\text{RMS} = \sqrt{\frac{1}{N} \sum_I (V_I - V_I^{PF})^2},
\]
where \( I \) runs over the sampling points of the respective trajectory.). Furthermore, the minimum and maximum of the difference along the trajectory are given.

If a SYMMETRY-SECTION is present in the input file the statistics are done twice before and after the symmetrization of the coefficients (the SPP are always symmetry adapted in case a SYMMETRY-SECTION is present). If the symmetries are implemented correctly one should observe an increase of the errors along the coeff trajectory while the errors should decrease along the independent test trajectory after symmetrization. If the sampling methods for the coefficient- and the test-trajectories are the same (in particular the same temperature in case of Metropolis sampling) the difference between the results obtained with the coeff and test samplings is therefore a good measure for the convergence of the coefficients in terms of sampling points.
13.8 Checking convergence and fit quality

Global (weighted) L² error estimated from neglected natural weights:

- 73.5279 meV
- 593.043 cm⁻¹
- 2.7021E-03 a.u.

ERRORS OBTAINED WITH COEFF-SAMPLING BEFORE COEFF-SYMMETRIZATION:

Statistics of the difference (V(exact) - V(natpot)) using 1000000 sampling points.

Mean = -4.3031E-06 au, -1.1709E-04 eV, -9.4443E-01 cm⁻¹
RMS = 8.4528E-04 au, 2.3001E-02 eV, 1.8552E-02 cm⁻¹
Minval = -7.0715E-02 au, -1.9243E+00 eV, -1.5520E+04 cm⁻¹
Maxval = 3.3613E-02 au, 9.1465E-01 eV, 7.3773E+03 cm⁻¹

ERRORS OBTAINED WITH TEST-SAMPLING BEFORE COEFF-SYMMETRIZATION:

Statistics of the difference (V(exact) - V(natpot)) using 1000000 sampling points.

Mean = -1.3461E-05 au, -3.6628E-04 eV, -2.9543E+00 cm⁻¹
RMS = 1.9485E-03 au, 5.3021E-02 eV, 4.2764E+02 cm⁻¹
Minval = -9.7127E-01 au, -2.6430E+01 eV, -2.1317E+05 cm⁻¹
Maxval = 7.3143E-01 au, 1.9903E+01 eV, 1.6053E+05 cm⁻¹

ERRORS OBTAINED WITH COEFF-SAMPLING:

Statistics of the difference (V(exact) - V(natpot)) using 1000000 sampling points.

Mean = -1.2842E-05 au, -3.4946E-04 eV, -2.8186E+00 cm⁻¹
RMS = 1.5235E-03 au, 4.1457E-02 eV, 3.3438E+02 cm⁻¹
Minval = -3.7359E-01 au, -1.0166E+01 eV, -8.1994E+04 cm⁻¹
Maxval = 3.4751E-01 au, 9.4562E+00 eV, 7.6269E+04 cm⁻¹

ERRORS OBTAINED WITH TEST-SAMPLING:

Statistics of the difference (V(exact) - V(natpot)) using 1000000 sampling points.

Mean = -1.5470E-05 au, -4.2097E-04 eV, -3.3954E+00 cm⁻¹
RMS = 1.6876E-03 au, 4.5922E-02 eV, 3.7039E+02 cm⁻¹
Minval = -4.1859E-01 au, -1.1390E+01 eV, -9.1870E+04 cm⁻¹
Maxval = 1.1291E-01 au, 3.0726E+00 eV, 2.4782E+04 cm⁻¹

Example 13.5: An excerpt of an output file for the Zundel cation (D₂d): After calculating the natpot file the fit errors are estimated with two sets of sampling points: the one that has been used for calculating the coefficients and an independent test set. If a SYMMETRY-SECTION is present this is done twice: before and after the symmetrization of the coefficients. The 15D Zundel cation is a very large system for mcpotfit. Hence, here one is happy with these rather large fit-errors. For smaller systems one usually aims for fit-errors which are smaller by one or two orders of magnitude.

13.8.3 Testing the fit with other trajectories

Often, a single test trajectory does not yield all information needed about the fit. For instance one may require information about the fit quality on certain regions of the potential or may want to test the fit with Metropolis trajectories obtained with different temperatures. To this end one can use the program npotminmax. With npotminmax one can test the potential fit along arbitrary trajectories.

npotminmax requires a small input file as displayed in Example 13.6. The input consists of a RUN-SECTION and a NATPOT-SECTION. The RUN-SECTION defines a name directory to which the program writes output files while the NATPOT-SECTION defines the directory in which the natpot file that is to be tested is located. The name directory should be different than the name directory of a potfit or mcpotfit run since several statistics files are written. npotminmax can in general test the sum of several natpot files (see HTML docu-
Example 13.6: An input file for npotminmax: the natpot file in directory h5o2p_fit_small_1250 is tested on the DVR points in file h5o2p_metr_2500/dvrindex-test. The exact corresponding energies of the potential are in the file h5o2p_metr_2500/energies-test.

13.9 Output files

Table 13.4 outlines a brief overview of the various output files of mcpotfit. In cases where files can be both, ASCII or binary, there is an option to specify the desired format in the RUN-SECTION.

<table>
<thead>
<tr>
<th>File</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>natpot</td>
<td>binary</td>
<td>The potential fit to be read in by mctdh.</td>
</tr>
<tr>
<td>dvr</td>
<td>binary</td>
<td>The DVR points as in mctdh.</td>
</tr>
<tr>
<td>output</td>
<td>ASCII</td>
<td>Natural population statistics for each mode and final error estimates and CPU time.</td>
</tr>
<tr>
<td>input</td>
<td>ASCII</td>
<td>A copy of the input file and command line options.</td>
</tr>
<tr>
<td>log</td>
<td>ASCII</td>
<td>General information and programs progress, such as files opened, Type of calculation,</td>
</tr>
</tbody>
</table>

continued...
### Continued

<table>
<thead>
<tr>
<th>File</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>conjgrad.log</td>
<td>ASCII</td>
<td>surface used, memory usage, converge of the SPP, error and warn messages, if any. Iterations and residual vector norm if conjugate gradients is used to solve for the coefficients</td>
</tr>
<tr>
<td>sym.log</td>
<td>ASCII</td>
<td>DOF and mode operations for symmetries and symmetry tests. Only present if symmetries are specified. If more than 1 test-trajectories are specified in the SAMPLING-SECTION this file contains statistical details of the fit error for each of the trajectories.</td>
</tr>
<tr>
<td>natpot-statistics</td>
<td></td>
<td>Matrices that are used to transform the coefficients for symmetrizing the fit. Only present if symmetries are specified.</td>
</tr>
<tr>
<td>sigma</td>
<td>ASCII</td>
<td>The extended intermediate grid of mode i. Each line contains the coordinates of all DOF of that mode at one DVR or extended point. Extended points are at the end of the file. Only present if symmetries are specified, and an extended grid is detected.</td>
</tr>
<tr>
<td>timing</td>
<td>ASCII</td>
<td>CPU time and number of calls to various subroutines.</td>
</tr>
<tr>
<td>dvrindex-spp</td>
<td>binary or ASCII</td>
<td>Integer DVR indices of the sampling points to generate the SPP. Binary or ASCII can be specified in the RUN-SECTION (default: ASCII). The index is given in the same order as the DOF in the PRIMITIVE-BASIS-SECTION. Each line (or record if binary) contains one sampling point.</td>
</tr>
<tr>
<td>dvrindex-coeff</td>
<td>binary or ASCII</td>
<td>Same as for dvrindex-spp but contains the sampling used for calculating the coefficients.</td>
</tr>
<tr>
<td>dvrindex-test</td>
<td>binary or ASCII</td>
<td>Same as for dvrindex-spp but contains the sampling used for the final test.</td>
</tr>
<tr>
<td>energies-spp</td>
<td>binary or ASCII</td>
<td>Energies that correspond to the DVR index points in file dvrindex-spp. If dvrindex-spp is binary then also energies-spp is binary. Each line (or record if binary) contains one single energy.</td>
</tr>
<tr>
<td>energies-coeff</td>
<td>binary or ASCII</td>
<td>Same as for energies-spp but contains the energies for the coefficient sampling.</td>
</tr>
<tr>
<td>energies-test</td>
<td>binary or ASCII</td>
<td>Same as for energies-spp but contains the energies for the test sampling.</td>
</tr>
<tr>
<td>energies</td>
<td>binary or ASCII</td>
<td>Same as for energies-spp, energies-coeff and energies-test, but in case the keyword same-sets is set and all samplings are the same.</td>
</tr>
<tr>
<td>eigenvalues</td>
<td>ASCII</td>
<td>If invert-method=eigen is set in the RUN-SECTION, then the eigenvalues</td>
</tr>
</tbody>
</table>

continued...
Table 13.4: Output files of mcPotfit and a brief description of their content.
Appendix A

The concept of the input file

With the exception of a continuation run, where a previous wavepacket propagation is carried on to longer times, all calculations require an input file, name.inp. This file is a text file, with the required options input as keywords. As all lines beginning with a # are treated as comments, title and other text can be usefully added to make the file easier to understand.

Example input A.1 shows the input file required for a simple wavepacket propagation calculation, using a modified Henon-Heiles Hamiltonian (see Ref. [2] for more details of this calculation). As the example shows, it is possible to include the information of the operator file in the input file. This is particularly useful for systems having a simple Hamiltonian.

The keywords in the input file are grouped together into sections, each with a specific set of information. The sections start with a line containing the keyword XXX-SECTION, and end with END-XXX-SECTION, where XXX is the name of the section. The possible sections are compiled in Tab. A.1.

Table A.1: The possible sections in the input file. Also displayed is whether a section is required for a certain calculation type.

<table>
<thead>
<tr>
<th>Section</th>
<th>gendvr</th>
<th>genoper</th>
<th>geninwf</th>
<th>propagation</th>
<th>diagonalisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PRIMITIVE-BASIS</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PARAMETER(^a)</td>
<td>yes(^b)</td>
<td>yes(^c)</td>
<td>yes(^c)</td>
<td>yes(^c)</td>
<td>yes(^c)</td>
</tr>
<tr>
<td>SPF-BASIS</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OPERATOR(^d)</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OP_DEFINE(^a)</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>HAMILTONIAN(^a)</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>LABELS(^a)</td>
<td>no</td>
<td>yes(^e)</td>
<td>yes(^e)</td>
<td>yes(^e)</td>
<td>yes(^e)</td>
</tr>
<tr>
<td>INITWF</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>INTEGRATOR</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

\(^a\) May be in the operator file
\(^b\) Only if parameters are used in the DVR specifications
\(^c\) Only if parameters are used in the Hamiltonian specification
\(^d\) Only if the Hamiltonian is in an operator file
\(^e\) Only if label definitions are required in the Hamiltonian specification
Table A.2: Description of the calculation types. The table shows the RUN-SECTION keyword required for a certain calculation type and the level associated with this type. Also given are the files that are created and needed by the different calculation types.

<table>
<thead>
<tr>
<th>Level</th>
<th>Keyword</th>
<th>Description</th>
<th>Created files</th>
<th>Required files</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gendvr</td>
<td>Sets up primitive bases</td>
<td>dvr</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>genpes</td>
<td>Sets up a pes file for analysis</td>
<td>pes, dvr</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>genoper</td>
<td>Sets up an operator for use</td>
<td>oper, dvr</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>geninwf</td>
<td>Sets up an initial wavefunction</td>
<td>restart, dvr, oper</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>propagation</td>
<td>Propagates a wavepacket</td>
<td>User defined, dvr, oper, restart</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>relaxation</td>
<td>Relaxes a wavepacket</td>
<td>User defined, dvr, oper, restart</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>diagonalisation</td>
<td>Diagonalises a Hamiltonian</td>
<td>User defined, dvr, oper, restart</td>
<td></td>
</tr>
</tbody>
</table>

Which sections are required depends on the type of calculation to be made. Table A.1 lists the various sections, and indicates which are required for the various types. A possible calculation type is, for instance, a propagation, a relaxation, or a diagonalisation, symbolised in the RUN-SECTION by the corresponding keyword `propagation`, `relaxation`, or `diagonalisation`. Additionally hereto, one may use the MCTDH program to solely set up a primitive basis, a Hamiltonian operator, or an initial wavepacket. This is done with the keywords `gendvr`, `genoper`, or `geninwf` in the RUN-SECTION. (The RUN-SECTION is hence required for all calculation types.) The generated information can then be read from file in following calculations be using the keywords `readdvr`, `readoper`, or `readinwf` in the RUN-SECTION.

Each calculation type has a level associated with it, which reflects the stages for a calculation. These levels are listed in Tab. A.2. Each level keyword automatically contains the lower levels, thus the keyword `propagation` implies `gendvr`, `genoper`, `geninwf`, `propagation`, and a wavepacket propagation will be performed after first setting up a DVR, operator, and initial wavepacket. The listed files are files which contain the information from the lower level calculations.

In the input file there may appear keywords which have a UNIX filename as argument (e.g., `oppath = · · ·`). These filenames are interpreted relative to the location of the input file.
Example A.1: An input file for a wavepacket propagation using the Henon-Heiles Hamiltonian.
Appendix B

The Structure of the Programs

Figure B.1 displays a flowchart of the MCTDH program package. The MCTDH program first reads the input file via the eingabe routines and computes the memory requirements. Depending on the input settings, it then starts some or all of the calculation types. The routines callx allocate the memory, the routines runx perform the calculations. Communication between these parts of the MCTDH program, as well as between the MCTDH and the Potfit and Analysis programs, is done employing the files indicated by ovals in the diagram.

Figure B.1: The structure of the MCTDH programs. See text for details.
Appendix C

The built-in symbolic expressions

The following tables describe the symbols and related operators that can be used to set up a Hamiltonian operator.

General Remarks

With the aid of the caret ^ one may apply a power to operators. The power may be integer or real and may carry a sign. This, however, works only for potential like operators. Inspect the Tables below to learn, which operators can be exponentiated. Note, that symbols like \( dq^2 \) or \( j^2 \) are operator labels of their own right, they do not denote that the second power of the operators \( dx \) or \( j \) is taken literally. (Compare with Appendix B (Discrete Variable Representation) of the MCTDH review (Phys.Rep. 324 (2000) 1-105), to learn how \( dq \) and \( dq^2 \) are defined).

One may multiply operators, e.g. a construct like \( dq \cdot \cos \cdot dq \) is allowed. However, multiplication is allowed only among potential like operators and operators with a simple matrix representation. This excludes all \( KLeg \) and \( PLeg \) operators from multiplication. (See Table C.2 and notes to this table). Moreover, \( natpots \) cannot be multiplied with other operators. (See Section 6.9)
Table C.1: Simple one-dimensional operators. The expression $x$ is the coordinate, $r$ can be replaced by any real number, positive or negative. If $r = 1$ it is not required, e.g. $q$ and $q^1$ are synonymous. The expression $n$ can be replaced by any non-negative integer.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$1$</td>
<td>Unit operator</td>
</tr>
<tr>
<td>$i^*1$</td>
<td>Multiply by $i$ times unit operator</td>
<td></td>
</tr>
<tr>
<td>$x^r$</td>
<td>Multiply by $r$th power of $x$</td>
<td></td>
</tr>
<tr>
<td>$(1 - x^2)^{r/2}$</td>
<td>Multiply by $r$th power of $\sqrt{1 - x^2}$</td>
<td></td>
</tr>
<tr>
<td>$\sin^r(x)$</td>
<td>$r$th power of sine of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\cos^r(x)$</td>
<td>$r$th power of cosine of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\tan^r(x)$</td>
<td>$r$th power of tangent of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\sinh^r(x)$</td>
<td>$r$th power of hyperbolic-sine of $x$</td>
<td></td>
</tr>
<tr>
<td>$\cosh^r(x)$</td>
<td>$r$th power of hyperbolic-cosine of $x$</td>
<td></td>
</tr>
<tr>
<td>$\arccos^r(x)$</td>
<td>$r$th power of arc-cosine of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\arcsin^r(x)$</td>
<td>$r$th power of arc-sine of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\arctan^r(x)$</td>
<td>$r$th power of arc-tangent of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\exp^r(x)$</td>
<td>Exponential of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\exp^r(\arccos(x))$</td>
<td>Exponential arc-cosine of coordinate</td>
<td></td>
</tr>
<tr>
<td>$\exp^r(\arccos(x)^2)$</td>
<td>Gaussian of arc-cosine of coordinate</td>
<td></td>
</tr>
<tr>
<td>$P_n(x)$</td>
<td>$n$th order Legendre polynomial of cosine of $x$</td>
<td></td>
</tr>
<tr>
<td>$P_{n}^m(x)$</td>
<td>Associated Legendre polynomial of $x$ (see function plgndr in source/lib/utilities/legendre.f)</td>
<td></td>
</tr>
<tr>
<td>$P_{n}^m(\cos(x))$</td>
<td>Associated Legendre polynomial of cosine of $x$</td>
<td></td>
</tr>
<tr>
<td>$\sqrt{J(J+1) - K(K+1)}$</td>
<td>$^{C_{JK}^+}$ symbol appearing with the $j_+$ operator (see Table C.2). $J$ is fixed.</td>
<td></td>
</tr>
<tr>
<td>$\sqrt{J(J+1) - K(K-1)}$</td>
<td>$^{C_{JK}^-}$ symbol appearing with the $j_-$ operator (see Table C.2). $J$ is fixed.</td>
<td></td>
</tr>
<tr>
<td>user supplied routine. See Note 8 of Table C.3.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table C.2: Operator symbols which require no arguments. The expression $n$ can be replaced by any positive integer. Finally, $m$ is the mass of the relevant degree of freedom. In the MCTDH input, this mass is given by the reserved parameter mass modelabel. If mass modelabel is not explicitly set, it is 1 by default.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>dq</td>
<td>$\partial_x$</td>
<td>first derivative. Cannot be used for rHO, Leg, KLeg, PLeg, Wigner or sphFBR.</td>
</tr>
<tr>
<td>dq$^2$</td>
<td>$\partial_x^2$</td>
<td>second derivative. Cannot be used for Leg, KLeg, PLeg, Wigner or sphFBR.</td>
</tr>
<tr>
<td>p</td>
<td>$-i\partial_x$</td>
<td>momentum (deprecated, use dq)</td>
</tr>
<tr>
<td>KE</td>
<td>$-\frac{1}{2m}\partial_x^2$</td>
<td>Kinetic energy term. Cannot be used for modes with a Legendre DVR or sphFBR.</td>
</tr>
<tr>
<td>$j^2$</td>
<td>$-\sin^{-1}(\theta) \partial_\theta \sin(\theta) \partial_\theta$ $- \sin^{-2}(\theta) \partial_\phi^2$</td>
<td>Angular momentum squared. In this form used for sphFBR and PLeg. For Leg and KLeg $\partial_\phi^2$ is replaced by $-m^2$ (or $-K^2$).</td>
</tr>
<tr>
<td>$j_+p$</td>
<td>$e^{i\theta}(\partial_\theta + i \cot(\theta) \partial_\phi)$</td>
<td>Angular momentum raising operator $j_+$. Only for KLeg and PLeg. For Wigner see below.</td>
</tr>
<tr>
<td>$j_-m$</td>
<td>$e^{-i\theta}(-\partial_\theta + i \cot(\theta) \partial_\phi)$</td>
<td>Angular momentum lowering operator $j_-$. Only for KLeg and PLeg. For Wigner see below.</td>
</tr>
<tr>
<td>jpm</td>
<td>$C_{jk}^+ j_+ + C_{jk}^- j_- $</td>
<td>Combined angular momentum operator. $C_{jk}^{\pm}$ are defined in Table C.1. Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>cjpm</td>
<td>$C_{jk}^+(j_{1,+} + j_{2,+}) + C_{jk}^-(j_{1,-} + j_{2,-})$</td>
<td>Combined operator for two angular momenta. Here $K = k_1 + k_2$, so this is different from jpm for the two individual angular momenta. Only for two successive KLegs, which furthermore have to be combined in one mode.</td>
</tr>
<tr>
<td>$j_z$</td>
<td>$j_z = -i\partial_\phi$</td>
<td>Angular momentum operator. Only for sphFBR and KLeg. For PLeg use dq or p on the $\phi$ DOF.</td>
</tr>
<tr>
<td>$j_z^2$</td>
<td>$(j_z)^2$</td>
<td>second power of angular momentum operator $j_z$. Only for sphFBR and KLeg. For PLeg use dq$^2$ on the $\phi$ DOF.</td>
</tr>
<tr>
<td>jp$^2$</td>
<td>$(j_+)^2$</td>
<td>Square of angular momentum raising operator. Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>jm$^2$</td>
<td>$(j_-)^2$</td>
<td>Square of angular momentum lowering operator. Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>jpm</td>
<td>$(j_+)*(j_-)$</td>
<td>Product of angular momentum raising and lowering operators. Only for KLeg and PLeg.</td>
</tr>
</tbody>
</table>
Table 2, continued.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmjp</td>
<td>(j_) * (j_)</td>
<td>Product of angular momentum lowering and raising operators. Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>sJp</td>
<td>sin(θ) * J_</td>
<td>J is total angular momentum. Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>sJm</td>
<td>sin(θ) * J_</td>
<td>J is total angular momentum. Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>sJpk</td>
<td>(sin(θ) * J_ * k + k * sin(θ) * J_) / 2</td>
<td>Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>sJmk</td>
<td>(sin(θ) * J_ * k + k * sin(θ) * J_) / 2</td>
<td>Only for KLeg and PLeg.</td>
</tr>
<tr>
<td>Jp</td>
<td>J_</td>
<td>multiplication with $C_{JK}^-$ and shift $k \to k - 1$</td>
</tr>
<tr>
<td>Jm</td>
<td>J_</td>
<td>multiplication with $C_{JK}^+$ and shift $k \to k + 1$</td>
</tr>
<tr>
<td>Jx</td>
<td>J_</td>
<td>$J_x = (J_+ + J_-)/2$</td>
</tr>
<tr>
<td>Jy</td>
<td>iJ_</td>
<td>$iJ_y = (J_+ - J_-)/2$</td>
</tr>
<tr>
<td>dth1</td>
<td>$\partial_\theta \sin \theta$</td>
<td>&quot;first derivative&quot;, only for Leg-KLeg- and PLeg-DVR (no symmetry).</td>
</tr>
<tr>
<td>dth2</td>
<td>$\frac{1}{2} \left( \cos \theta \partial_\theta \sin \theta + \partial_\theta \sin \theta \cos \theta \right)$</td>
<td>&quot;first derivative&quot;, only for Leg-KLeg- and PLeg-DVR (symmetry).</td>
</tr>
<tr>
<td>qdq</td>
<td>$\frac{1}{2} (x \partial_x + \partial_x x)$</td>
<td>for rHO-DVR this replaces the &quot;first derivative&quot;.</td>
</tr>
<tr>
<td>sdq</td>
<td>$\frac{1}{2} (\sin(x) \partial_x + \partial_x \sin(x))$</td>
<td>for cos-DVR this replaces the &quot;first derivative&quot;.</td>
</tr>
<tr>
<td>sdq2</td>
<td>$\frac{1}{2} (\sin^2(x) \partial_x + \partial_x \sin^2(x))$</td>
<td></td>
</tr>
<tr>
<td>cdq</td>
<td>$\frac{1}{2} (\cos(x) \partial_x + \partial_x \cos(x))$</td>
<td></td>
</tr>
<tr>
<td>cdq2</td>
<td>$\frac{1}{2} (\cos^2(x) \partial_x + \partial_x \cos^2(x))$</td>
<td></td>
</tr>
<tr>
<td>csdq</td>
<td>$\frac{1}{2} (\sin(x) \cos(x) \partial_x + \partial_x \sin(x) \cos(x))$</td>
<td></td>
</tr>
<tr>
<td>udq</td>
<td>$\frac{1}{2} (\sqrt{1-x^2} \partial_x + \partial_x \sqrt{1-x^2})$</td>
<td></td>
</tr>
<tr>
<td>uqdq</td>
<td>$\frac{1}{2} (\sqrt{1-x^2} x \partial_x + \partial_x \sqrt{1-x^2} x)$</td>
<td></td>
</tr>
<tr>
<td>udq2</td>
<td>$\frac{1}{2} ((1-x^2) \partial_x + \partial_x (1-x^2))$</td>
<td></td>
</tr>
</tbody>
</table>

(continued)
Table 2, continued.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(j^2)</td>
<td>(-\partial_\beta^2 - \cot(\beta) \partial_\beta - \sin^{-2}(\beta) {\partial_\alpha^2 + \partial_\gamma^2 - 2 \cos(\beta) \partial_\alpha \partial_\gamma})</td>
<td>Wigner-DVR angular momentum squared operator, with matrix elements: (j^2</td>
</tr>
<tr>
<td>(j_\rho)</td>
<td>((j_\rho)^{BF} = e^{i\gamma} \left( \frac{i}{\sin(\beta)} \partial_\alpha \partial_\beta - i \cot(\beta) \partial_\gamma \right))</td>
<td>Wigner-DVR body-fixed angular momentum lowering operator, which operates as: ((j^-)^{BF}</td>
</tr>
<tr>
<td>(j_m)</td>
<td>((j^-)^{BF} = e^{i\gamma} \left( \frac{i}{\sin(\beta)} \partial_\alpha \partial_\beta - i \cot(\beta) \partial_\gamma \right))</td>
<td>Wigner-DVR body-fixed angular momentum raising operator, which operates as: ((j^+)^{BF}</td>
</tr>
<tr>
<td>(j_p)</td>
<td>((j_\rho)^{SF} = e^{i\alpha} \left( \frac{i}{\cos(\beta)} \partial_\alpha \partial_\beta - \frac{i}{\sin(\beta)} \partial_\gamma \right))</td>
<td>Wigner-DVR space-fixed angular momentum raising operator, which operates as: ((j^+)^{SF}</td>
</tr>
<tr>
<td>(j_m)</td>
<td>((j^-)^{SF} = e^{-i\alpha} \left( \frac{i}{\cos(\beta)} \partial_\alpha \partial_\beta - \frac{i}{\sin(\beta)} \partial_\gamma \right))</td>
<td>Wigner-DVR space-fixed angular momentum lowering operator, which operates as: ((j^-)^{SF}</td>
</tr>
<tr>
<td>(j_{pm})</td>
<td>(C_{JK}^+ (j^-)^{BF} + C_{JK}^- (j^+)^{BF})</td>
<td>Wigner-DVR body-fixed combined angular momentum operator. (C_{JK}^\pm) are defined in Table C.1.</td>
</tr>
<tr>
<td>(j_{pms})</td>
<td>(C_{JM}^+ (j_\rho)^{SF} + C_{JM}^- (j^-)^{SF})</td>
<td>Wigner-DVR space-fixed combined angular momentum operator. (C_{JM}^\pm) are defined in Table C.1, but here M replaces K.</td>
</tr>
<tr>
<td>(j_{p^2})</td>
<td>((j_\rho)^2_{BF})</td>
<td>Wigner-DVR squared body-fixed angular momentum raising operator.</td>
</tr>
<tr>
<td>(j_{p^2s})</td>
<td>((j_\rho)^2_{SF})</td>
<td>Wigner-DVR squared space-fixed angular momentum raising operator.</td>
</tr>
<tr>
<td>(j_{m^2})</td>
<td>((j^-)^2_{BF})</td>
<td>Wigner-DVR squared body-fixed angular momentum lowering operator.</td>
</tr>
<tr>
<td>(j_{m^2s})</td>
<td>((j^-)^2_{SF})</td>
<td>Wigner-DVR squared space-fixed angular momentum lowering operator.</td>
</tr>
</tbody>
</table>
Table 2, continued.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpm</td>
<td>((j_+)<em>\text{BF} \ast (j</em>-)_\text{BF})</td>
<td>Wigner-DVR product of body-fixed angular momentum raising and lowering operators.</td>
</tr>
<tr>
<td>jpmj</td>
<td>((j_+)<em>\text{SF} \ast (j</em>-)_\text{SF})</td>
<td>Wigner-DVR product of space-fixed angular momentum raising and lowering operators.</td>
</tr>
<tr>
<td>jmjp</td>
<td>((j_-)<em>\text{BF} \ast (j</em>+)_\text{BF})</td>
<td>Wigner-DVR product of body-fixed angular momentum lowering and raising operators.</td>
</tr>
<tr>
<td>jmjps</td>
<td>((j_-)<em>\text{SF} \ast (j</em>+)_\text{SF})</td>
<td>Wigner-DVR product of space-fixed angular momentum lowering and raising operators.</td>
</tr>
<tr>
<td>jpjz</td>
<td>((j_+)<em>\text{BF} \ast (j_z)</em>\text{BF})</td>
<td>Wigner-DVR product of body-fixed angular momentum operators, which operates as: ((j_+)<em>\text{BF}(j_z)</em>\text{BF}</td>
</tr>
<tr>
<td>jpjzs</td>
<td>((j_+)<em>\text{SF} \ast (j_z)</em>\text{SF})</td>
<td>Wigner-DVR product of space-fixed angular momentum operators, which operates as: ((j_+)<em>\text{SF}(j_z)</em>\text{SF}</td>
</tr>
<tr>
<td>jzp</td>
<td>((j_z)<em>\text{BF} \ast (j</em>+)_\text{BF})</td>
<td>Wigner-DVR product of body-fixed angular momentum operators, which operates as: ((j_z)<em>\text{BF}(j</em>+)_\text{BF}</td>
</tr>
<tr>
<td>jzp</td>
<td>((j_z)<em>\text{SF} \ast (j</em>+)_\text{SF})</td>
<td>Wigner-DVR product of space-fixed angular momentum operators, which operates as: ((j_z)<em>\text{SF}(j</em>+)_\text{SF}</td>
</tr>
<tr>
<td>jmjz</td>
<td>((j_-)<em>\text{BF} \ast (j_z)</em>\text{BF})</td>
<td>Wigner-DVR product of body-fixed angular momentum operators, which operates as: ((j_-)<em>\text{BF}(j_z)</em>\text{BF}</td>
</tr>
<tr>
<td>jmjzs</td>
<td>((j_-)<em>\text{SF} \ast (j_z)</em>\text{SF})</td>
<td>Wigner-DVR product of space-fixed angular momentum operators, which operates as: ((j_-)<em>\text{SF}(j_z)</em>\text{SF}</td>
</tr>
</tbody>
</table>

(continued)
Table 2, continued.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Notes</th>
</tr>
</thead>
</table>
| \( j_{zj}m \) | \((j_z)_{BF} \ast (j_-)_{BF}\) | Wigner-DVR product of body-fixed angular momentum operators, which operates as: 
\[ (j_{zj}m_{BF}(j_-)_{BF} | J, K, M) = (K + 1)\sqrt{J(J+1) - K(K+1)} | J, K, M + 1, L). \] |
| \( j_{zj}ms \) | \((j_z)_{SF} \ast (j_-)_{SF}\) | Wigner-DVR product of space-fixed angular momentum operators, which operates as: 
\[ (j_{zj}ms_{SF}(j_-)_{SF} | J, K, M) = (M - 1)\sqrt{J(J+1) - M(M-1)} | J, K, M - 1) \] |

Notes to Table C.2

The volume-element assumed for Leg/KLeg/PLeg/Wigner is \( \sin \theta d\theta \), whereas all other DVRs in MCTDH assume the simple volume-element \( dq \). Because of the non-trivial volume-element, \( \partial_\theta \) is not an anti-hermitian operator, only \( \partial_\theta \sin \theta \) (i.e. \( d\theta_1 \)) is. Note that 
\[
\sin \theta \partial_\theta = \partial_\theta \sin \theta - \cos \theta = d\theta_1 - \cos \theta.
\]

The operators \( j_p, j_m, jpm, j^2 \) and \( jm^2 \) are (KLeg or PLeg) 2D mode-operators, i.e. they operate on the combined mode \((\theta,k)\) or \((\theta,\phi)\) for KLeg or PLeg, respectively. Note that \( j^2 \) becomes a 2D mode-operator, when operating on a KLeg or PLeg mode. Similarly, the operators \( sjp, sjm, sjpk, \) and \( sjmk \) are also 2D KLeg/PLeg operators, where \( J \) denotes the total angular momentum. The 2D mode operator \( j_p \) performs a multiplicative and shift operation on \( k \), but additionally performs a derivative and \( k \)-dependent multiplicative operation on the \( \theta \)-dof of the KLeg mode. Similar operations are done by the \( j_m, sjp, sjm, sjpk, \) and \( sjmk \) operators.

The operators \( jz, j^2, Jp, Jm, Jx, Jz \) are 1D operators and operate on the \( k \)-dof of the KLeg mode only. Hence they must appear in the \( k \)-column whereas the 2D KLeg-operators must appear under the \( \theta \) column. (A more vivid way of writing the operator file is to let the 2D operator appear under both columns by using the \( | \& \) construct, see Section 6.13. However, in contrast to potential functions one must not reorder the DOFs of KLeg/PLeg/Wigner operators. For example \( |2 \& 3 \) is fine, but \( |3 \& 2 \) is not.)

When applied to Wigner functions, the operators \( j^2, j_p, j_m, jpm, j^2, jm^2, jpm, jmjp, jzjz, jzjm, \) and \( jzjz \) are 3D mode operators and are represented as 4D tensors in MCTDH, so care must be taken when multiplying these operators with other operators. The Wigner operators \( j_p, j_m, jpm, j^2, jm^2, jpm, jmjp, jzjz, jzjm, \) and \( jzjz \) operate in the BODY-fixed axis system; that is, these operators perform multiplicative operations and shifts depending on the \( k \) (second) degree of freedom in the combined 3D mode. The corresponding SPACE-fixed operators, which perform multiplications and shifts depending on the \( m \) (third) DOF, are denoted \( jps, jms, jpm, j^2s, jm^2s, jmmps, jjzps, jzjps, jmmps, \) and \( jzjms \). Note that \( j_+ \) and \( j_- \) are defined as \( j_+ = j_x + ij_y \) and \( j_- = j_x - ij_y \) for both the SF- and BF-system. Due to the anomalous commutation relation for the BF operators, \( j_p = (j_+)^{BF} \) decreases \( k \) by one, whereas \( j_{ps} = (j_+)^{SF} \) increases \( m \) by one.
The operator cjpm is a 4D mode-operator (two successive KLegs). It only works if the two KLegs are combined into one mode, and for this case it replaces the use of natural potentials of the cpp/cmm surfaces (see Table C.5).

Note that the operators \( j_p, j_m, jpm, cjpm, j_p^2, j_m^2, sjp, sjm, sjpk \) and \( sjmk \) — as well as \( j^2 \) if the latter operates on a KLeg/PLeg combined mode — are 3D tensors in MCTDH and not matrices. Hence care must be taken when multiplying these operators with other operators. To give an example

```
HAMILTONIAN-SECTION
-----------------------------------
| modes | ... | theta | k |
-----------------------------------
| ...   | ... | \( \cos^*j_p \) | c_p |
| ...   | ... | \( j_p^*\cos \) | c_p |
-----------------------------------
end-hamiltonian-section
```

is a valid construct. Note that the 2D operator \( j_p \) may be multiplied from right or left with an operator operating on \( \theta \) only. However, it may be multiplied only from right with a local \( k \)-dependent function (here \( c_p \)).

The operators dth1, dth2, qdq, and sdq replace the first derivative operator for the Leg (and KLeg/PLeg), \( rHO, \cos \) (and \( \sin \) when the keyword \( sdq \) is given) DVR, respectively. In these cases the operator \( dq \) cannot be used.
Table C.3: One-dimensional operators which require arguments. The expression \( x \) stands for the coordinate, \( p \) can be replaced by any parameter from the PARAMETER-SECTION, or any real number. The exponent \( r \) can be any real number. If \( r = 1 \) it is not required, e.g. \( q[p] \) and \( q[p] \) are synonymous.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q[p]^r )</td>
<td>((x - p)^r)</td>
</tr>
<tr>
<td>( qs[p]^r )</td>
<td>((\sqrt{p - x^2})^r)</td>
</tr>
<tr>
<td>( \sin[p1,p2]^r )</td>
<td>(\sin^r(p1(x - p2)))</td>
</tr>
<tr>
<td>( \cos[p1,p2]^r )</td>
<td>(\cos^r(p1(x - p2)))</td>
</tr>
<tr>
<td>( \tan[p1,p2]^r )</td>
<td>(\tan^r(p1(x - p2)))</td>
</tr>
<tr>
<td>( \exp[p1,p2]^r )</td>
<td>(\exp^r(p1(x - p2)))</td>
</tr>
<tr>
<td>( \expq[p1,p2]^r )</td>
<td>(\exp^r(i \cdot p1(x - p2)))</td>
</tr>
<tr>
<td>( \text{texp}[p1,p2]^r )</td>
<td>(\exp^r(p1(\arccos(x) - p2)))</td>
</tr>
<tr>
<td>( \sinh[p1,p2]^r )</td>
<td>(\sinh^r(p1(x - p2)))</td>
</tr>
<tr>
<td>( \cosh[p1,p2]^r )</td>
<td>(\cosh^r(p1(x - p2)))</td>
</tr>
<tr>
<td>( \tanh[p1,p2]^r )</td>
<td>(\tanh^r(p1(x - p2)))</td>
</tr>
<tr>
<td>( \cos[p1,p2]^r )</td>
<td>((\cos(p1 \cdot x) - \cos(p1 \cdot p2))^r)</td>
</tr>
<tr>
<td>( \exp\cos[p1,p2]^r )</td>
<td>((1 - \exp(p1(x - p2)))^r)</td>
</tr>
<tr>
<td>( \exp\cosq[p1,p2]^r )</td>
<td>(\exp(p1(\cos(x)) - \exp(p1 \cos(p2)))^r)</td>
</tr>
<tr>
<td>( \text{qtanh}[p1,p2,p3]^r )</td>
<td>(\tanh^r(p2(\arccos(x) - p1)^{p3}))</td>
</tr>
<tr>
<td>( \text{motanh}[p1,p2,p3,p4]^r )</td>
<td>(\tanh^r(p3[1 - \exp(-p1(x - p2))]^{p4}))</td>
</tr>
<tr>
<td>( \text{asin}[p1,p2,p3]^r )</td>
<td>(\text{arcsin}(p1 \cdot x - p2 - p3)^r)</td>
</tr>
<tr>
<td>( \text{acos}[p1,p2,p3]^r )</td>
<td>(\text{arccos}(p1 \cdot x - p2 - p3)^r)</td>
</tr>
<tr>
<td>( \text{atan}[p1,p2,p3]^r )</td>
<td>(\text{atan}(p1 \cdot x - p2 - p3)^r)</td>
</tr>
<tr>
<td>( \text{coschirp}[p1,p2,p3]^r )</td>
<td>(\text{cos}[x[p2 + (p1 - p2) \exp(-(x/p3)^2)]])</td>
</tr>
<tr>
<td>( \text{tgauss}[p1,p2]^r )</td>
<td>(\exp(-p1(\arccos(x) - p2)^2)^r)</td>
</tr>
<tr>
<td>( \text{gauss}[p1,p2]^r )</td>
<td>(\exp(-p1(x - p2)^2)^r)</td>
</tr>
<tr>
<td>( \text{ngauss}[x,x_0]^r )</td>
<td>((2\pi \sigma^2)^{-1/2} \exp(-x^2/(2\sigma^2))^r)</td>
</tr>
<tr>
<td>( \text{morse}[p1,p2,p3,p4,p5] )</td>
<td>Morse function. See Note 1.</td>
</tr>
<tr>
<td>( \text{morse1}[p1,p2,p3,p4] )</td>
<td>Morse function. See Note 1.</td>
</tr>
<tr>
<td>( \text{CAP}[p1,p2,p3,p4] )</td>
<td>(-iW). See Note 2.</td>
</tr>
<tr>
<td>( \text{ACAP}[p1,p2,p3,p4,p5] )</td>
<td>(-iW). See Note 3.</td>
</tr>
<tr>
<td>( \text{switch1}[p1,p2] )</td>
<td>(0.5 \cdot [1 - \tanh(p1(x - p2))])</td>
</tr>
<tr>
<td>( \text{switch2}[p1,p2] )</td>
<td>(0.5 \cdot [1 + \tanh(p1(x - p2))])</td>
</tr>
<tr>
<td>( \text{step}[p] )</td>
<td>(\Theta(x - p)) Step function. See Note 4.</td>
</tr>
<tr>
<td>( \text{rstep}[p] )</td>
<td>(\Theta(p - x)) Reverse step function. See Note 4.</td>
</tr>
<tr>
<td>( \text{charfun}[p1,p2] )</td>
<td>characteristic function: if ( x \in [p1, p2] ) then charfun=1 else it is zero.</td>
</tr>
<tr>
<td>( \text{regcoul}[p1,p2] )</td>
<td>regularized coulomb function: (1/\sqrt{(x - p1)^2 + p2}).</td>
</tr>
<tr>
<td>( \text{low}[m,\omega,s] )</td>
<td>lowering operator. See Note 5.</td>
</tr>
<tr>
<td>( \text{rai}[m,\omega,s] )</td>
<td>raising operator. See Note 5.</td>
</tr>
<tr>
<td>( \text{num}[m,\omega,s] )</td>
<td>number operator. See Note 5.</td>
</tr>
</tbody>
</table>

(continued)
Table 3, continued.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>ramorse[m,\omega,\Lambda,\alpha,z_0]</td>
<td>raising operator for Morse potential. See Note 6.</td>
</tr>
<tr>
<td>lwmorse[m,\omega,\Lambda,\alpha,z_0]</td>
<td>lowering operator for Morse potential. See Note 6.</td>
</tr>
<tr>
<td>cspot[J,K,csmax,m]</td>
<td>centrifugal potential. See Note 7.</td>
</tr>
<tr>
<td>external1d{file}</td>
<td>external 1D function read from file file. See Note 8.</td>
</tr>
<tr>
<td>my1d{p1,p2,p3,p4,p5]</td>
<td>user supplied routine. See Note 10.</td>
</tr>
<tr>
<td>flux[x_c,power]</td>
<td>Flux operator for Cartesian kinetic energy. $x_c$ = location of dividing surface, $power$ = exponent of smoothing function. See Note 11.</td>
</tr>
<tr>
<td>pgauss[\sigma,x_0]</td>
<td>Projector $</td>
</tr>
<tr>
<td>shift[\Omega]</td>
<td>simple shift on the grid by $\Omega$, $\tilde{\psi}(x_i) = \psi(x_i-\Omega)$ $\tilde{\psi}(x_i) = 0$ if $i-\Omega &lt; 1$ or $i-\Omega &gt; N$ For a periodic shift on a periodic grid (exp-DVR of FFT) use shift[\Omega] and shift[\Omega-N] (with $\Omega &gt; 0$).</td>
</tr>
</tbody>
</table>

Table C.4: One-dimensional potential energy curves.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Potential Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>v:NO</td>
<td>NO potential curve</td>
</tr>
<tr>
<td>v:H2</td>
<td>H2 potential (link lsth)</td>
</tr>
<tr>
<td>vbmkp:H2</td>
<td>H2 potential (link h4bmkp)</td>
</tr>
<tr>
<td>v:HO</td>
<td>OH potential, morse function from h2o.f</td>
</tr>
<tr>
<td>v:OH</td>
<td>OH potential (link hoosrf)</td>
</tr>
<tr>
<td>v:CH</td>
<td>CH potential (link c2h)</td>
</tr>
<tr>
<td>v:C2</td>
<td>C2 potential (link c2hasec)</td>
</tr>
<tr>
<td>v:OF</td>
<td>OF potential curve</td>
</tr>
<tr>
<td>vrho:H3</td>
<td>H+H2 potential in hypersphaerical coordinates, theta=π (link lsth)</td>
</tr>
<tr>
<td>vthe:H3</td>
<td>H+H2 potential in hypersphaerical coordinates, rho=2.484773 (link lsth)</td>
</tr>
<tr>
<td>v dj:000</td>
<td>expansion coeffiecent $V_{000}$ for DJ H4 surface (link h4dj)</td>
</tr>
<tr>
<td>v dj:022</td>
<td>expansion coeffiecent $V_{022}$ for DJ H4 surface (link h4dj)</td>
</tr>
<tr>
<td>v dj:224</td>
<td>expansion coeffiecent $V_{224}$ for DJ H4 surface (link h4dj)</td>
</tr>
</tbody>
</table>
### Table C.5: Two-dimensional operators (used for molecule-surface scattering).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cosh \cos [ \theta ] )</td>
<td>( \cosh(p \cos(\theta)) )</td>
</tr>
<tr>
<td>( \sinh \cos [ \theta ] )</td>
<td>( \sinh(p \cos(\theta)) )</td>
</tr>
<tr>
<td>( \cosh \sin \cos \phi )</td>
<td>( \cos(p \sin(\theta) \cos(\phi)) )</td>
</tr>
<tr>
<td>( \cosh \sin \sin \phi )</td>
<td>( \cos(p \sin(\theta) \sin(\phi)) )</td>
</tr>
<tr>
<td>( \sinh \sin \cos \phi )</td>
<td>( \sin(p \sin(\theta) \cos(\phi)) )</td>
</tr>
<tr>
<td>( \sinh \sin \sin \phi )</td>
<td>( \sin(p \sin(\theta) \sin(\phi)) )</td>
</tr>
<tr>
<td>( \text{Re}Y[l,m] )</td>
<td>( \text{Re}(Y^m_l(\theta,\phi)) )</td>
</tr>
<tr>
<td>( \text{Im}Y[l,m] )</td>
<td>( \text{Im}(Y^m_l(\theta,\phi)) )</td>
</tr>
</tbody>
</table>

### Table C.6: Multi-dimensional \( C_+ \), \( C_- \) symbols defined on truncated \( k_1, k_2, \ldots, k_d \) grid.

See also “Hamiltonian Documentation”/“Available Surfaces”

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{cpp} { jtot=J, dim=d } )</td>
<td>( \sqrt{J(J+1) - (\sum_{j=1}^d k_j)(\sum_{j=1}^d k_j+1)} )</td>
</tr>
<tr>
<td>( \text{cmm} { jtot=J, dim=d } )</td>
<td>( \sqrt{J(J+1) - (\sum_{j=1}^d k_j)(\sum_{j=1}^d k_j-1)} )</td>
</tr>
</tbody>
</table>

### Table C.7: Some general multi-dimensional operators. Here parameters are to be given in curly brackets.

E.g: \( \text{coulomb1d}\{a=1.3 b=2.0 c=0.0 d=1.5\} \). See also “Hamiltonian Documentation”/“Available Surfaces”

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
</table>
| \( \text{readsrfs} \{ file F \} \) | Potential values on grid points are read from file file of format \( F \).
|                | \( F=\text{ascii or binary}. \) See HTML Docu. For 1D-potential use read1d). |
| \( \text{gauss1d} \{ \text{width=}=w S \} \) | \( \exp(-0.5((x_1-x_2)/w)^2)/\sqrt{2\pi w} \) |
|                | If the optional string S is set to periodic, then a \( 2\pi \) periodic grid is assumed. |
|                | In this case the DVR lines should read e.g.: |
|                | \( x1 \) FFT 128 2pi |
|                | \( x2 \) FFT 128 2pi |
| \( \text{gauss2d} \{ \text{width=}=w \} \) | \( \exp(-0.5((x_1-x_2)/w)^2 + (y_1-y_2)/w^2)/2\pi w^2 \) |
| \( \text{coulomb1d} \{ \cdots \} \) | \( 1/\sqrt{(ax_1 - bx_2 + c)^2 + d} \) |
Table C.8: One-dimensional operators for treating symmetric double-well potentials by mapping each side on an artificial (single-set) electronic state. Note, the grid must be a sin-DVR, which, when doubled, lies symmetrically to zero but does not contain zero. The differential operators which are truncated are firstly defined on this doubled grid, but then projected to the working grid.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rf</td>
<td>$\varphi(x_i) = \varphi(x_{N+1-i})$</td>
<td>Reflection operator. (Must not be multiplied with other operators).</td>
</tr>
<tr>
<td>Rfm</td>
<td>$-\varphi(x_{N+1-i})$</td>
<td>Reflection operator. (Must not be multiplied with other operators).</td>
</tr>
<tr>
<td>hKEh</td>
<td>step<em>KE</em>step</td>
<td>Truncated kinetic energy</td>
</tr>
<tr>
<td>hFRh</td>
<td>step<em>KE</em>Rf*step</td>
<td>Truncated kinetic energy times reflection</td>
</tr>
<tr>
<td>hdqh</td>
<td>step<em>dq</em>step</td>
<td>Truncated first derivative</td>
</tr>
<tr>
<td>hdqRh</td>
<td>step<em>dq</em>Rf*step</td>
<td>Truncated first derivative times reflection</td>
</tr>
<tr>
<td>dqR</td>
<td>dq*R</td>
<td>First derivative times reflection</td>
</tr>
<tr>
<td>dq2R</td>
<td>dq^2*R</td>
<td>Second derivative times reflection</td>
</tr>
</tbody>
</table>

Notes to Table C.3

All input variables $\ldots$ are numbers, parameters or arithmetic expressions containing numbers and parameters. (See Hamiltonian-Documentation/Parameter-Section for details). The use of units is not allowed here. Note that these symbolic expressions with parameters must not appear in a HAMILTONIAN-SECTION. They rather have to be linked to simple symbols (without parameters) in a LABELS-SECTION. (Compare with Example 6.4).

1. A morse curve can be given by $D(\exp(-\alpha(x-x_0))-1)^2 + E_0$, where $D$ is the dissociation energy (depth parameter), $\alpha$ defines the curvature, $x_0$ the equilibrium position, and $E_0$ is an energy shift parameter. If one uses the symbol morse1 these are precisely the input parameters, i.e. $[D, \alpha, x_0, E_0]$. For the symbol morse the input parameters are $[D, \omega, x_0, ex_0, m]$, where $m$ is the mass, $\omega$ is the frequency of the related harmonic oscillator, and $ex_0$ is the position at which the potential is zero. Note that $\omega$ and $\alpha$ are related by $\alpha^2 = m\omega^2/2D$, while $ex_0$ and $E_0$ are related by $E_0 = -D(\exp(-\alpha(ex_0-x_0))-1)^2$.

2. A CAP (Complex Absorbing Potential) is an imaginary, negative potential, used to absorb a wavepacket as it approaches the end of the grid. It is defined as $-iW$, where $W = \eta \Theta(k(x-x_0))(k(x-x_0))^n$ and where $\Theta$ denotes the Heaviside’s step function. The input parameters are $[x_0, \eta, n, k]$, where $k$ is used to choose to which end of the grid the CAP is placed: $k = -1$ puts the CAP at the left, and $k = 1$ at the right of the grid. $k = 1$ is default and may be left out.

3. ACAP symbolises an automatic CAP. The ACAP is useful, when one wants to place the initial wavepacket at a position, where it overlaps with the CAP. The ACAP remains disabled as long as the wavepacket overlaps with the CAP. The ACAP is enabled only when the wavepacket starts to re-enter the region where the CAP is defined. There is a fifth parameter: timecap. If this optional parameter is set, the ACAP will remain disabled at least as long as time < timecap, where time is the propagation time in fs. The parameter timecap is useful, because the automatic enabling of the CAP may sometimes happen too early. The time, at which the ACAP is switched on, is protocoted in
the log file. Use this information to set the option \(-1\circ\) in \texttt{flux84} appropriately. When \texttt{flux84} is run, it must not evaluate matrix elements of the CAP for times, at which the CAP is switched off.

4. The symbols \texttt{step} and \texttt{rstep} symbolise a Heaviside’s step function and the reverse of it. I.e. \texttt{step}[p] = \Theta(x - p) and \texttt{rstep}[p] = 1 - \Theta(x - p) = \Theta(p - x).

5. The lowering operator corresponding to a harmonic oscillator is given by
   \[ b = \frac{i}{\sqrt{2m\omega}} p + \sqrt{\frac{m\omega}{2}} (q - q_0), \]
   where \( p \) denotes the momentum operator, \( q \) denotes the position operator, \( m \) is the mass, \( \omega \) is the frequency, and \( q_0 \) is the equilibrium position. The input parameters are \([m, \omega, s]\), which means the mass \( m \), the frequency \( \omega \), and the shift \( s = -\frac{\sqrt{m\omega}}{2} q_0 \).
   (Note the minus sign). The corresponding raising operator is given by \( b^\dagger \), and the number operator by \( \hat{n} = b^\dagger b \). The parameters have the same meaning as for lowering operators. NB. The lowering, raising and number operator require the use of a simple DVR with an ordinary first derivative, e.g. \texttt{sin}, \texttt{HO}, or \texttt{exp} but not \texttt{FFT}, \texttt{rHO}, \texttt{Leg}, \texttt{KLeg}, \texttt{PLeg} or \texttt{sphFBR}.

6. The (approximate) raising/lowering operators (R/L) for a Morse Hamiltonian
   \[ H = \frac{p^2}{2m} + D \left( e^{-2\alpha(x-x_0)} - 2e^{-\alpha(x-x_0)} \right) \]
   are defined as
   \[ L = \sqrt{\frac{\hbar}{2m\omega}} \left[ \left( \Lambda - \frac{1}{2} \right) \alpha - \Lambda \alpha e^{-\alpha(x-x_0)} + \frac{i}{\hbar} p \right] \]
   and
   \[ R = L^\dagger \]
   with \( \omega = \alpha\sqrt{2D/m} \) and \( \Lambda = \sqrt{2Dm/\hbar\alpha} \).

7. The centrifugal potential given by:
   \[ V_{\text{cent}}(x) = \min \left( \frac{J(J + 1) - 2K^2}{2mx^2}, csmax \right). \]

8. An arbitrary (real) 1D–function may be defined through a set of points. The points are read from file \texttt{file} and are then interpolated to define a general 1D–function. The data is in free format with one (x,y) data pair per line. Blank lines and lines which start with a \# are ignored. Currently, the x-data (called \texttt{time} in the code) must increase linearly, i.e. must be equally spaced.
   An arbitrary number of these, with different data files, may be used in one operator. However, when using multiple instances of the same external1d function, do this by defining a label and referring to it in the operator rather than declaring external1d\{\texttt{file}\} with the same file \texttt{file} repeatedly, as this wastes buffer memory due to duplication.
9. An arbitrary (real) 1D–function may be defined through a set of points. The points must coincide with the grid points. The potential values on the grip points are read from file \texttt{file}, one value per line. The file may be in ascii or binary format. (binary is default). Give \texttt{ascii} or \texttt{binary} as second argument after \texttt{file}. (For multi-dimensional surfaces use \texttt{readsr}f).

10. A (real) 1D–function may be defined through a user written subroutine. Edit the subroutine \texttt{my1d} on \$MCTDH\_DIR/source/opfuncs/func1d.F .

11. The flux operator $[\Theta, T]$ is set up in a sine or exponential basis and then transformed to DVR representation. This operator might be used with \texttt{eigenf} to produce flux-eigenstates as initial wavefunctions. To regularize the flux operator and to make its eigenfunctions more localized, it is multiplied from right and left with \[\cos(\pi p/2p_{max})\cosh(\pi p/2p_{max})\]^\texttt{power}. The exponent \texttt{power} may be zero or any positive real number. \texttt{power} = 1 is recommended.

**Special operators**

There is a number of operators especially defined for the methyl-iodine (CH3I) system. Their labels all start with MI: . See \texttt{opfuncs/ch3i.f} and \texttt{opfuncs/ch3igrd.f} for further information.

**Non-adiabatic operators**

If the system contains more than one electronic state, the Hamiltonian can be written in matrix form, i.e.

$$
\hat{H} = \begin{pmatrix}
H_{11} & H_{12} & \cdots \\
H_{21} & H_{22} & \cdots \\
\vdots & \vdots & \ddots
\end{pmatrix}
$$  \hspace{1cm} (C.1)

To input such a form, the symbols in Table C.9 can be used. Thus the operator

$$h_1 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + h_2 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + h_3 \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}
$$  \hspace{1cm} (C.2)

can be represented symbolically as

\begin{tabular}{c|c|c}
\texttt{modes} & \texttt{X} & \texttt{el} \\
1.0 & \texttt{h}_1 & 1 \\
1.0 & \texttt{h}_2 & \texttt{S1}\&\texttt{2} \\
1.0 & \texttt{h}_3 & \texttt{Z1}\&\texttt{2} \\
\end{tabular}

See also Sec. 9 for more examples.

**Table C.9:** Matrix operator symbols, used for an electronic degree of freedom.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sf&amp;i</td>
<td>Symmetric matrix element</td>
</tr>
<tr>
<td>Zf&amp;i</td>
<td>Unsymmetric matrix element</td>
</tr>
<tr>
<td>1</td>
<td>Unit matrix</td>
</tr>
</tbody>
</table>
Appendix D

Structure of the WF array

\[ \psi(dgldim) \]

\[ \Psi \]

\[ \psi(adim) \]

\[ A \]

\[ \psi(totphidim) \]

\[ \phi \]

\[ zpsi(1) \]

\[ \text{block(s)} \]

\[ A^{(s)} \]

\[ \psi(block(s)) \]

\[ zpsi(s) \]

\[ \text{subdim}(m), \text{dim}(m,s) \]

\[ \phi^{(m,s)} \]

\[ zetf(m,s) \]

\[ \text{zetf}(m,1) \]

\[ \text{zetf}(m,s) \]

\[ \text{zetf}(1,1) \]

Figure D.1: The structure of the wave function
Appendix E

Installing the MCTDH package

The installation of the MCTDH package is very easy if you install it on a PC with a not too old Linux system. On more fancier machines one may need to edit the compile scripts (to set compiler options appropriately) and/or install some open-source software like gnuplot or a GNU make. It is essential to have a bash shell (version 3.x is recommended, but lower versions may also work) and it is highly recommended that you work under bash, although it is possible to work under C-shell or kshell as well. A bash, however, must exist as there are several bash scripts. Moreover, for some tasks like OCT or Cluster Expansion python scripts are used. Hence, python 2.4 or higher (but NOT python 3) should be installed.

One should begin with creating a directory MCTDH which eventually will contain all the MCTDH stuff, but not – at least I prefer to do so – the output of production runs. Move the MCTDH tar-ball to the MCTDH directory and unzip and untar it. I.e.

```
mkdir /home/muser/MCTDH
cd /home/muser/MCTDH
mv <path>/mctdh85.x.tgz .
tar xzvf mctdh85.x.tgz
```

Here it is assumed that your login name is muser and that you have a GNU tar. If you do not have a GNU tar you first have to gunzip the tar-ball and then untar it (without the option z). The symbol x stands for the revision number of the particular package which was downloaded. When the tar command is finished, there should exist a directory mctdh85.x under /home/muser/MCTDH.

If you are familiar with Subversion (svn), it will be more convenient to download the code form the svn-repository of the Heidelberg MCTDH package. For details see Appendix F.

After the code is downloaded, move to the directory mctdh85.x/install and run

```
check_system:
```

cd mctdh85.x/install
./check_system

This will create an output like
E Installing the MCTDH package

********************************************************************************
************** ------- CHECK SYSTEM ------- **************
********************************************************************************

Mon Dec 9 14:42:09 CET 2013

The path of the MCTDH-directory is: /home/dieter/MCTDH/mctdh85.2

System : Linux
Platform : i686
Operating-System : GNU/Linux
Machine : cauchy
Processor : unknown
Kernel : Linux

This is a 32-bit system.

These are the variables determined by platform.cnf
MCTDHPLATFORM = i686
MCTDH_COMPILER = gfortran
MCTDH_GFORTRAN_VERSION = 4.7.2

These are the default compilers
Fortran compiler: gfortran
C compiler : gcc
make command : make

If you want to use other compilers please edit platform.cnf* and possibly also compile.cnf* (or use the -c option when compiling).

The Fortran compiler gfortran is /apps/gcc-4.7/bin/gfortran
The C compiler gcc is /apps/gcc-4.7/bin/gcc
The make command make is /usr/bin/make
Congratulation, you have a GNU make.

GNUPLOT exist on your system: gnuplot 4.4 patchlevel 0
Python exist on your system: Python 2.6.6

Your bash is: GNU bash, version 4.1.5(1)-release

Distributed memory parallelization with MPI seems to be possible!
For this use Fortran compiler consistent with:
gcc version 4.7.2 (GCC)

Shared memory parallelization with POSIX-Threads seems to be possible!

The use of the NUMA library for the shared memory parallelization with POSIX-Threads seems to be possible!
Compile with "-u" option to enable the use of NUMA. (See "compile -h").

The standard installation should work without problems.
If not already done, you should now run "./install_mctdh".
This is just for fun, but if you read the message "The standard installation should work without problems", then there should be no problems. By the way, please note that the GNU compilers GCC 3.4.0 – 3.4.3 and GCC 4.0.0 – 4.1.0 are buggy, DO NOT USE THEM. We recommend to use GNU-GCC 4.6.0 or higher. For the 8.5-branch, where we start to use some fancy features of FORTRAN95/2003 we recommend to use even higher versions, e.g. 4.9.2 or higher. However, currently (2015) 4.7.1 is still sufficient. Next to GNU, the intel, pgi, and some other compilers are also fine. See install/compile.cnf for possible compilers. (You may edit the compile scripts and add new compilers to it).

Please read the README file of the install directory and start with the installation, i.e. type

```
./install_mctdh
```

while being in the install directory. The script will ask you several questions and in general you should answer yes, i.e. type y. (In fact, one may run yes | ./install_mctdh but this is not recommended for beginners.) Please, try to understand all the questions before giving an answer. If you think you have made a wrong choice, you can always stop the installation process with the ctrl c command and then start anew. The install_mctdh script LaTeX compiles the guide, compiles the code, sets some environment variables, and writes the path of the MCTDH directory to your ./bashrc file (or some other configuration file if you are not running under bash). Note that you have to source your ./bashrc (or other configuration) file to activate the changes. The path to the MCTDH directory is stored in the environment variable MCTDH_DIR, to inspect it type echo $MCTDH_DIR. But it is more convenient to run the script menv, which writes a list of all MCTDH environment variables to screen.

If you are running under bash the install script will also create the ~/.mctdhrc file. This enables the powerful cdm command (try cdm -h) and sets a link ~/.mctdh which points to the currently active MCTDH directory, in the present case to /home/muser/MCTDH/mctdh85.x. (One may install several MCTDH packages, but only one will be active. Use the script mininstall to switch between the different versions.)

If you want to make use of the surfaces library, move the file addsurf.tgz to your MCTDH directory, i.e. in our example to /home/muser/MCTDH, and untar it. Then you should edit the ~/.mctdhrc. In this case simply remove the # in front of export MCTDH_ADDSURF and source .mctdhrc to activate this change. Similarly, one can set environment variables in .mctdhrc which point to the MCTDH backup and elk directories (for users who want to change the code). Use the script mkl links to set a link to the PES requested. However, one should do so only when a PES is needed. After a PES is linked to the $MCTDH_DIR/source/surfaces directory, one has to compile mctdh or potfit with the -i option, e.g. execute the commands

```
mkl links h4bmkp
compile -i h4bmkp potfit
```

before running potfit85 to bring the BMKP surface of the H₄ system to product form.

Load the URL file:///home/muser/mctdh/doc/index.html into your browser to inspect the HTML on-line documentation. Bookmark this page! A simple but quick help is provided through the script mhelp. It briefly explains the keywords of the input
Installing the MCTDH package  

E  Installing the MCTDH package  

file. Try mhelp -h. (All MCTDH scripts and programs know the help option -h). If you want to inspect the code, try the scripts mcb, mcg, mcl, and phelp. 

You may go to the AdvancedUser directory (e.g. type cd m AdvancedUser) and execute make there (read the README file first). This will give you access to additional scripts and routines (mcalc is quite useful).  

There are four compile-configuration files on the install directory: compile.cnf_le, compile.cnf_be, compile.cnf_lenp, and compile.cnf_benp. The letters le and be stand for little endians and big endians, respectively, np denotes no parallelization. The install_mctdh script copies one of these four configuration files to compile.cnf, the default choice is compile.cnf_le. The file compile.cnf is then read by the compile script which is used to compile individual programs (e.g. compile mctdh) or the full package (compile all). If your compiler does not support pthreads, you have to choose compile.cnf_lenp or compile.cnf_benp. We have tried to find reasonable options for the compilers (see in particular MCTDH_FFLAGS_OPT), but we cannot account for any hardware and software installation on which MCTDH may run. Hence, depending on your particular hardware/software installation, the choice of compiler options may not be optimal. (To inspect the compiler option run compile config). Feel free to adjust the compiler options to your particular installation. If you add a new compiler, please send us the updated compile.cnf file.  

If you are working on a system where computers of different kind (32 bit / 64 bit, Linux / other Unix (including Mac OS X)) are interconnected by a common file system, you may store and install the MCTDH package only once, but run compile on each kind. MCTDH is smart and will load automatically the correct executables. Run menv on interconnected computers of different kind and you will see that the paths are set differently. 

In general, each MCTDH user works with his own package. This allows him to change the code according to his demand. However, sometimes it may be wanted that several users have access to the same package. In this case there is a master-user who installs the package and clients who only need to add the line  

```
source $MCTDH_DIR/install/MCTDH_client
```

to their .bashrc. The file MCTDH_client is generated during installation. Of course, $MCTDH_DIR must be replaced by the full path of the MCTDH directory, which for the present example reads /home/muser/MCTDH/mctdh85.x. Alternatively, the clients may simply copy the file MCTDH_client to their .bashrc.  

If the automatic detection of platform and compiler does not work, one has to edit the platform.cnf script. (Note that during installation platform.cnf.def is copied to platform.cnf. Hence one may wish to edit platform.cnf.def as well). Around line 100, platform.cnf reads:  

```bash
#---------------------------------------------
# SET MACHINE-DEPENDENT OPTIONS
#---------------------------------------------

system='uname -s'
## system=MYSYSTEM # Incomment, if automatic dection doesnt work.
case $system in
  MYSYSTEM) # Here you may set the variables by hand.
    MCTDH_PLATFORM=  # Please set! (try 'uname -n' or 'uname -m')
    MCTDH_COMPILER= # Please set! (if not listed in compile.cnf, you
    ;; # have to edit compile.cnf as well.)
```

Change this to e.g.:
# SET MACHINE-DEPENDENT OPTIONS

```bash
system='uname -s'
system=MYSYSTEM  # Incomment, if automatic dection doesnt work.
case $system in
  MYSYSTEM) # Here you may set the variables by hand.
    MCTDH_PLATFORM=cruncher
    MCTDH_COMPILER=pgf77
  ;;
This is, of course, just an example. One can give any name to MCTDH_PLATFORM, a convenient choice is the output of `uname -n` or `uname -m`. The symbol which is given to MCTDH_COMPILER must be listed in the compile.cnf file. If one wants to use a compiler, which is not listed there, one has to edit compile.cnf to add the new compiler. (Note that during installation one of the files compile.cnf_* is copied to compile.cnf, where * stand for le, be, lenp, or benp. Hence one may wish to edit those files as well.)

Finally, let us summarize the commands you now should be familiar with: `menv`, `compile`, `mhelp`, `cdm`, and, if you use a PES from addsurf, `mklinks`. Try the help option `-h` and inspect the HTML on-line documentation “The Analyse Programs / Utility Scripts” to learn more about the utility scripts. If you want to inspect the code, make yourself familiar with `mcb`, `mcg`, `mcl`, and `phelp`. There is also a backup facility and an automatic program test (Elk Test), see the HTML on-line documentation for details. For additional information on the install process see the page “Installation and Compilation” of the HTML on-line documentaion.

When the installation is completed, it is advisable to work through the tutorial (Sec. 2).

A final remark on Apple computers running under Mac OS x (Darwin) should be made. Versions launched 2014 or later install painlessly on a Mac after some additional software is installed. One needs Apple’s Xcode (to obtain `make`), the GNU-compilers `gcc` and `gfortran` with version GCC-4.7.x or higher, and `gnuplot`. Currently the parallelization with pthreads does not work, although the pthread libraries are installed (?). However, after installing the openMPI software, one can use MCTDH with MPI parallelization.
Appendix F

The svn-repository of the Heidelberg MCTDH package

We use Subversion, or short svn, for version control of the MCTDH package. As svn is likely to be available on your computer installation, we open the possibility to download the MCTDH package directly from our svn-repository, rather than from the MCTDH website http://mctdh.uni-hd.de/packages/ . If you are new to svn you may wish to consult the svn-book, which can be downloaded from the URL http://svnbook.red-bean.com/en/1.7/svnbook.pdf .

To access the MCTDH svn-repository, a username and password are needed. These are given in the Letter to the new MCTDH user and are the same as the ones requested to access the MCTDH web-site http://mctdh.uni-hd.de/packages/ .

F.1 Useful svn commands

In order to abbreviate the commands, we suggest to add the following lines to your .bashrc or .alias file.

```
alias svnm="svn --username <user> --password <psswd> --non-interactive"
export SVNM="svn://www.pci.uni-heidelberg.de:/mctdh"
```

where <user> and <psswd> are to be replaced with the username and password given in the Letter to the new MCTDH user.

To get an overview on the available releases of the mctdh 8.4 branch, submit the command:

```
svnm list $SVNM/mctdh84/releases/
```

This will provide an output similar to

```
8.4.10/
8.4.4/
8.4.4.1/
8.4.4.2/
8.4.5/
8.4.6/
8.4.7/
```
where, of course, one may exchange mctdh84 with mctdh83 or mctdh85 to list the contents of those directories.

If you want to download version 8.4.10 (this is an example, please download the most recent version), type

```
svn export $SVNM/mctdh84/releases/8.4.10/ mctdh84.10
```

where the directory mctdh84.10 will be created by svn, it should not previously exist. Of course, one may give any name to the final directory and may give its full path, if it is to be created in a directory different from the current one. The `svn export` command will provide you with exactly the same data as found on the mctdh84.10.tgz file of http://mctdh.uni-hd.de/packages/.

A better alternative is to use

```
svn checkout $SVNM/mctdh84/releases/8.4.10/ mctdh84.10
```

The difference is that with this command additionally a couple of .svn files will be copied to the final directory, which almost doubles the size of the latter. However, the .svn files give you access to most of the svn-commands. E.g. moving (cd) the mctdh-directory (here mctdh84.10) and submitting the command

```
svn status
```

will tell you which files are modified or added with respect to the repository. Or

```
svn diff --old=$SVNM/mctdh84/releases/8.4.10/ --new=. 
```

will display the differences between your code and the one on the repository. (You may pipe this output to less). If you would like to have a line by line comparison of the two versions, you may add the option `--diff-cmd kdiff3` to the command above.

If you have already checked-out a previous version and want to merge with a newer one, type e.g.

```
svn merge $SVNM/mctdh84/releases/8.4.9/ $SVNM/mctdh84/releases/8.4.10/ 
```

This command merges the differences between release 8.4.9 and 8.4.10 to your mctdh-directory, which must be the current directory. Here we are assuming that you are working with release 8.4.9 and are updating to 8.4.10.

Moreover, rather than downloading a release, one may download the current developers code

```
svn checkout $SVNM/mctdh84/trunk/ mctdh84.dev
```

This makes life easier, as one can simply run

```
svn update
```

to merge with the most recent changes. However, this way is recommended more for experienced users, as the current developers code may not be bug-free. To be on the safe side, one
may run the command

svnm cat $SVNM/mctdh84/trunk/changelog | less

and then update to an appropriate revision by setting the option \texttt{-r<number>}. Finally, if one is interested in the branches 8.3 or 8.5 rather than 8.4 one simply replaces the version numbers accordingly.
List of MCTDH references


List of MCTDH references


Index

ABM integrator, 80, 82
Adiabatic correction, see Correction
Adiabatic population, 107
Analysis
  flux analysis, 105
  of accuracy, 99
  of efficiency, 102
  of electronic populations, 107
  of PES, 112
  of primitive basis, 99
  of PSI, 111
  of results, 96
  of single-particle basis, 101
  of system evolution, 102
  of system spectrum, 104
  reaction probabilities, 105
analysis interface, 96
Auto file, see File
Autospec program, see Program
Auxiliary Operators, 66
Basis
  electronic, 87
  primitive, 36
  single-particle, 45, 87
Bosons, 91
BS integrator, 80, 82
Calculations
  continuing, 29
  distributed memory, 32
  parallel, 29-32
  shared memory, 29
  starting, 28
  stopping, 29
CAP, 62
  order, 62
  starting point, 62
  strength, 62
CDVR, see Correlation DVR
Check file, see File
CMF scheme, see Constant mean-field scheme
Colbert-Miller DVR, see DVR, sine
Complex absorbing potential, see DVR
Constant mean-field scheme, 81
Continuing a calculation, see Calculations
Correction
  adiabatic, 78
  diabatic, 78
Correlation DVR, 85
CSIL integrator, see SIL integrator
Diagonalisation, 28
Discrete variable representation, see DVR
Distributed memory, see Calculations
DOF, mode, and muld potentials, 67
DVR, 36
  exponential, 39
  extended Legendre (KLeg), 42
  Hermite, 36
  Legendre, 38
  radial Hermite, 36
  restricted Legendre, 41
  sine, 39
  three-dimensional rotational (Wigner), 43
  two-dimensional Legendre (PLeg), 42
DVR file, see File
Efield program, see Program
Electronic basis, see Basis
Energy cut-off, 66
Energy distribution, 78
Energy weights, 25
Error estimate
  of the SIL integrator, 83
Error message, 28
Error tolerance
  of the ABM integrator, 82
  of the BS integrator, 82
  of the CMF scheme, 82
  of the RK5/8 integrator, 82
  of the SIL integrator, 82
Exponential DVR, see DVR
Extended Legendre, see Legendre
Extended Legendre DVR, see DVR
Fast Fourier transform, see FFT
FBR, 36
Fdcheck program, see Program
Fdmatch program, see Program
FFT, 39
  Temperton, 40
File
  auto, 22, 104
  check, 101, 107
  chk.pl, 7
  dvr, 36
  eigval, 11
  enerd, 78, 105
  flux, 105

185
flux.log, 105
gridpop, 99, 102
gtau, 105
input, 115, 147
iteration, 115
log, 22, 115
natpot, 115
operator, 48
orben, 25
output, 22, 97, 115
pes, 5, 112
prodwei, 115
psi, 22, 111
ptiming, 29, 32
restart, 76
spectrum.pl, 4, 8, 104
stop, 29
surface, 65
timing, 102, 115
vpot, 115
wtt, 105

File number, 57
Filter program, see Program
Finite basis-set representation, see FBR
Flux program, see Program
Fourier-transformed potential, see Potential
Gaussian, 71
harmonic oscillator, 36
Legendre, 38, 72
particle-in-a-box, 39
spherical harmonic, 41, 73
Gaussian function, see Function
Golden rules, 69
Gridpop file, see File
Hamiltonian section, see Section
Harmonic oscillator DVR, see DVR, Hermite
Harmonic oscillator function, see Function
Henon-Heiles, 51
Hermite DVR, see DVR
Improved block-relaxation, see Wavepacket
Improved relaxation, see Wavepacket, 84
Init_wf section, see Section
Initial stepsize
for the CMF scheme, 82
for the ABM integrator, 83
for the BS integrator, 83
for the RK5/8 integrator, 83
Initial wavefunction, see Wavefunction
Input file, see File, see File
Installing package, 166
Integration order, 83
Integration schemes, 80
Integrator section, see Section
Interaction picture orbital, see Orbital
Iteration file, see File
KLeg, 42
KLeg, see DVR
Labels section, see Section
Lanczos algorithm, 28
Lanczos integrator, see SIL integrator
Lanczos-Arnoldi integrator, see SIL integrator
Legendre
DVR, see DVR
extended, 73
function, see Function
Log file, see File, see File
Metropolis sampling, 129
Mode combination, 46
Monte-Carlo, 129
Mul potentials, 67
Multi-packet, 78
Multi-set, 87
Name-directory, 22
Natpot, 61
Natpot file, see File
Natural orbital, see Orbital
Natural population, see Population
Natural potential, see Potential
Non-adiabatic system, see System
Numerically exact calculation, 27
Op_define section, see Section
Operator file, see File
Operator section, see Section
Operator, 1D, user-defined, 54
Optcntrl program, see Program
Orben file, see File
Orbital
energies, 25
interaction picture, 84
natural, 84
Output file, see File, see File
Parallel calculation, see Calculations, see Calculations
Parameter section, see Section
Particle-in-a-box function, see Function
Plall program, see Program
Plane wave, 40
Plane-wave DVR, see DVR, exponential
Plauto program, see Program
Plbrlx program, see Program, see Program
Plicap program, see Program
PLeg, 43
PLeg, see DVR
Plfspec program, see Program
Plit program, see Program
Plpweight program, see Program
Plqdv program, see Program
Plrlx program, see Program, see Program
Plspec program, see Program
Plstate program, see Program

Plupdate program, see Program
Population
  natural, 101
  of grid points, 99
Potential
  ab initio, 118
  Fourier-transform of, 125
  multi-dimensional, 58, 67
  natural, 61, 114
  non-separable, 58
  one-dimensional, 57
  separable, 56
Potfit program, see Program
Primitive basis, see Basis
Primitive-basis section, see Section
Product form, 51, 86
Prodwei file, see File
Program
  adpop, 107
  adproj, 108
  analysis, 96
  autospec, 4, 8, 104, 105
  efield, 19
  fdcheck, 13
  fdmatch, 13
  filter, 12
  flux, 105
  mcpotfit, 129
  opcentrl, 19
  plall, 5
  plauto, 5
  plbrlx, 16, 27
  plcap, 63
  plfspec, 13
  pflux, 9, 105, 106
  plbit, 115
  plpweight, 115
  plqdq, 5
  plrlx, 15, 16, 27
  plspec, 5, 8, 104, 105
  plspeed, 5
  plstate, 7, 107
  plupdate, 5
  plwt, 9, 105
  potfit, 9, 114
  projection, 122
  rdcheck, 7, 101, 107
  rdgpop, 99
  rdrlx, 15, 16, 27
  reflex, 63
  showd1d, 4, 9, 102
  showpot, 112, 115
  showrst, 103
  showspf, 103
  showsys, 5, 111, 112
Program structure, 150
Projection program, see Program
Propagation, see Wavepacket
Psi file, see File
Radial harmonic oscillator DVR, see DVR, radial Hermite
Radial Hermite DVR, see DVR
Rdcheck program, see Program
Rdgpop program, see Program
Rdrlx program, see Program, see Program
readsr, 118
Reflex program, see Program
Relaxation, see Wavepacket
relevant region, 114
Restart file, see File
Restricted Legendre DVR, see DVR
RK5/8 integrator, 80, 82
Rotator DVR, see DVR, Legendre
Run section, see Section
Section
  correlated-weight, 116
  Hamiltonian, 51, 87, 147
  init_wf, 71, 147
  integrator, 80, 147
  labels, 61, 147
  natpot-basis, 115
  op_define, 48, 147
  operator, 48, 115, 147
  parameter, 49, 147
  primitive-basis, 36, 87, 116, 147
  run, 21, 115, 147
  separable-weight, 116
  spf-basis, 45, 88, 147
Shared memory, see Calculations
Showd1d program, see Program
Showpot program, see Program, see Program
Showrst program, see Program
Showspf program, see Program
Showsys program, see Program
SIL integrator, 82
Sine DVR, see DVR
Single-particle basis, see Basis
Single-particle function, 45
  multi-mode, 46
Single-particle operator, 51
Single-particle-basis section, see Section, spf-basis
Single-set, 87
Spectrum, see Analysis
Spf-basis section, see Section
Spherical harmonic function, see Function
Spherical harmonics FBR, 41
Starting a calculation, see Calculations
Stop file, see File
Stopping a calculation, see Calculations
Structure of the WF array, 165
Surface file, see File
svn-repository, Subversion, 171
Symbolic expression
  built-in, 51, 86, 151
  user-defined, 54
System, bosonic, see Bosons
System, non-adiabatic, 86
TDDVR, see Time-dependent DVR
TDH, see Time-dependent Hartree
Temperton FFT, see FFT
Three-Dimensional rotational DVR, see DVR
Time-dependent DVR, 85
Time-dependent Hartree, 45
Time-dependent operators, 70
Timing file, see File, see File
Two-dimensional Legendre DVR, see DVR

Variable mean-field scheme, 80
VMF scheme, see Variable mean-field scheme
Vpot file, see File

Wavefunction
   initial, 71, 90
   structure of, 165

Wavepacket
   improved block-relaxation, 15, 24
   improved relaxation, 14, 24
   propagation, 23
   relaxation, 23

Wigner, see DVR