

AKIRA

Mode-Tracking of Molecular Vibrations

PVM and MPI Parallelized Mode-Tracking Calculations using raw
data produced by ADF, DALTON, GAUSSIAN or TURBOMOLE

Johannes Neugebauer, Carmen Herrmann,
Stephan Schenk and Markus Reiher

Laboratorium für Physikalische Chemie¹, ETH Zürich,
Wolfgang-Pauli-Str. 10, CH-8093 Zürich, Switzerland

Version 3.4.0
June, 2009

¹contact email: markus.reiher@phys.chem.ethz.ch

Copyright (c) 2009 J. Neugebauer, C. Herrmann, St. Schenk and M. Reiher.

Contents

1	Introduction	7
2	Quickstart	11
3	Installation and technical issues	13
3.1	Getting started	13
3.1.1	General: the commands in the bin and scripts directory	13
3.1.2	The DCINPUT file	14
3.1.3	Preparations for the PVM version	16
3.1.4	Preparations for the MPI versions	16
3.2	Parallelization standards	17
3.2.1	PVM	17
3.2.2	MPI	18
3.3	Memory management and pre-processing	18
3.4	Installation of AKIRADefine and AKIRA	18
3.5	Monitoring AKIRA calculations	18
3.6	Further programs and scripts needed by AKIRA	19
4	Methodology	21
4.1	Subspace iteration techniques	21
4.1.1	Davidson algorithm	21
4.1.2	Preconditioning: The Davidson algorithm	23
4.1.3	The Jacobi–Davidson algorithm	25
4.1.4	The Lanczos algorithm	27
4.1.5	Details of the implementation	28
4.2	Generation of displaced structures	28
4.2.1	Displacements in units of length	28
4.2.2	Calculation of second derivatives	29
5	Running the calculation	31
5.1	Preparations	31
5.1.1	Structure optimization	31
5.1.2	Electronically excited states	31
5.1.3	Molecular symmetry	32
5.2	AKIRADefine: setting up the calculation	32

5.2.1	Program-specific input	33
5.2.2	Program selection	36
5.2.3	Convergence criteria menu	38
5.2.4	Output menu	39
5.2.5	Hessian guess and rigid modes	40
5.2.6	Root homing and preconditioning	41
5.2.7	Additional keywords	42
5.2.8	Data collector options	43
5.3	AKIRADefine: Setting up the initial guess	43
5.3.1	Internal coordinates	44
5.3.2	Symmetry coordinates	44
5.3.3	Cartesian coordinates	45
5.3.4	Normal coordinates from file	45
5.3.5	Subsystem menu	47
5.3.6	Other options	49
5.3.7	Restricting the Subspace	50
5.4	Subspace iteration with AKIRA	50
5.5	Restart facilities	53
5.6	Double-parallel runs	57
A	First aid	59
B	Parameter analysis	61

Advice for Program Usage

The AKIRA program implements the MODE-TRACKING technique, which allows you to target normal modes directly and circumvents the computer-time demanding calculation of *all* normal modes. However, though we have put much effort into setting up a computer program which is almost a *black box* code, it does not replace thinking about the particular problem to be solved and requires a basic understanding of vibrational analyses in general (in order to gain this, the reader is referred to the literature below and references cited therein). Therefore, when you have installed the program, our advice is to start with some small toy examples in order to learn about how the algorithm works (some test examples come with the package). In general, MODE-TRACKING is well suited in cases of

1. very large molecules for which a complete calculation of the spectrum is not feasible,
2. standard systems if only limited computer time is available so that a complete calculation cannot be carried out, and
3. smaller systems in combination with highly accurate *ab initio* calculations.

References:

- Mode-Tracking Reference:
M. Reiher, J. Neugebauer, *A mode-selective quantum-chemical method for tracking molecular vibrations applied to functionalized carbon nanotubes*, J. Chem. Phys. **118** **2003**, 1634–1641.
- Convergence Analysis of Mode-Tracking Algorithm:
M. Reiher, J. Neugebauer, *Convergence Characteristics and Efficiency of Mode-Tracking Calculations on Pre-Selected Molecular Vibrations*, Phys. Chem. Chem. Phys. **6** **2004**, 4621–4629.
- Mode-Tracking for a Molecular Subsystem:
J. Neugebauer, M. Reiher, *Modetracking of Pre-selected Vibrations of One-Dimensional Molecular Wires*, J. Phys. Chem. A **108** **2004**, 2053–2061.
J. Neugebauer, M. Reiher, *Vibrational Center–Ligand Couplings in Transition Metal Complexes*, J. Comput. Chem. **25** **2004**, 587–597.

- Reference for Semi-Numerical Harmonic Frequency Calculations:
J. Neugebauer, M. Reiher, C. Kind, B. A. Hess, *Quantum Chemical Calculation of Vibrational Spectra of Large Molecules — Raman and IR spectra for Buckminsterfullerene*, J. Comput. Chem. *23* **2002**, 895–910.
- Mode-Tracking for Molecules Adsorbed at Surfaces:
C. Herrmann, M. Reiher, *Direct targeting of adsorbate vibrations with mode-tracking*, Surf. Sci. *600* **2006**, 1891–1900.
- QM/MM-Mode-Tracking:
C. Herrmann, J. Neugebauer, M. Reiher, *QM/MM Vibrational Mode Tracking*, J. Comput. Chem. *29* **2008**, 2460–2470.

For a review on Mode-Tracking see:

- C. Herrmann, J. Neugebauer, M. Reiher, *Finding a needle in a haystack: direct determination of vibrational signatures in complex systems*, New J. Chem. *31* **2007**, 818–831.

1. Introduction

The development of the program AKIRA¹ started in the Theoretical Chemistry department at the University of Erlangen–Nuremberg for an efficient calculation of vibrational frequencies and normal modes. The Mode-Tracking protocol developed for and used in AKIRA allows the specific calculation of characteristic normal modes as introduced in Ref. [1] (see also [2]), while normal frequency analyses always determine the full set of normal modes and frequencies of a molecule. This is usually much more information than required, since in many cases only a small subset of characteristic vibrations is desired and necessary for comparison to or prediction of experimental data.

AKIRA originated in parts from the vibrational spectroscopy program package SNF² of the Theoretical Chemistry Group at the University of Erlangen–Nuremberg. The original SNF package aims at the parallelized calculation of complete vibrational spectra within the double harmonic approximation (SNF [3]).

AKIRA follows a different strategy for the determination of vibrational normal modes involving predefined (collective) motions of the atoms in a molecule. These motions (=cartesian distortions) have to be selected by the user, and the algorithm applied in AKIRA will determine all normal modes of vibrations which involve these characteristic motions. Since the creation of the initial (guess) vibration is the most delicate step for the user, we have created a setup tool for this purpose:

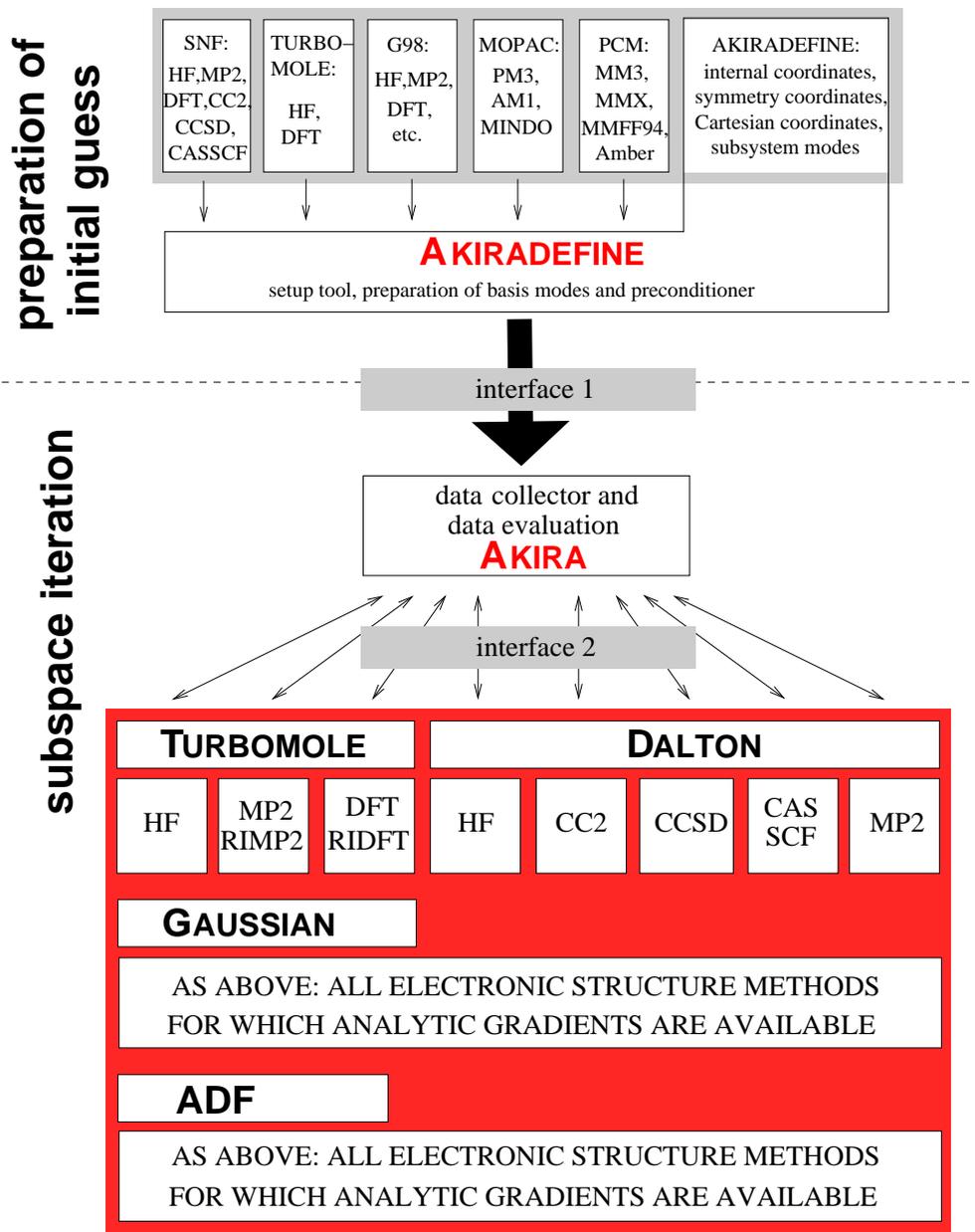
The setup tool AKIRADefine features several modules to create initial motions, either as Cartesian or internal coordinates, or from lower-level approximations like semiempirical models or force field calculations. Furthermore it is possible to use normal modes of former SNF or TURBOMOLE calculations as an initial guess for the desired normal mode, so that calculations with smaller basis sets can be applied as initial guesses for the mode-tracking with a larger basis set. Another possibility is to use normal modes for a sub-system of the molecule as a guess for the normal mode of the complete molecule.

AKIRA uses subspace iteration techniques in order to find only those

¹download page: http://www.thch.uni-bonn.de/tc/groups/reiher/download_akira.html

²see http://www.thch.uni-bonn.de/tc/groups/reiher/download_snf.html for download information of the SNF package

Figure 1.1: Hierarchical structure of the mode-tracking program AKIRA.



roots of the Hessian Matrix whose eigenvectors show the largest overlap with the initial guess vectors. Davidson- [4], Jacobi-Davidson- [5], and Lanczos-type [6] diagonalization schemes can be selected for this purpose.

This manual is organized as follows: Sect. 3 contains help on how to get AKIRA binaries running on your system. In case you have been given the source code, a description of the installation of AKIRA is given and the requirements for the parallelization are explained. In Sect. 4, the theoretical background of the subspace iteration methods employed in AKIRA is illustrated and the numerical derivative methods are explained in detail. Sect. 5 deals with how to prepare and to run modetracking calculations using the commands AKIRADefine and AKIRA, respectively. If an AKIRA calculation crashes, you will find some information on first aid in the appendix. An analysis of the influence of the step size used in numerical differentiation on the calculation of vibrational frequencies is also given in the appendix.

Any use of this program that results in published material should cite the following:

- M. Reiher, J. Neugebauer, *J. Chem. Phys.* **118** (2003), 1634 – 1641.

This article introduces the MODE-TRACKING method and describes the implementation of the algorithm; comparisons of some initial guesses and preconditioners are also given. We should, however, emphasize that we cannot warrant that the method is indeed suitable for your particular purpose (see also page 5).

Further details of the program capabilities and examples for mode-tracking calculations are given in the following articles (see also page 5):

- *extraction of couplings between different parts of a molecule, applications to large gold clusters:*
J. Neugebauer, M. Reiher *J. Comput. Chem.* **25** (2004), 587 – 597.
- *local vibrations in large molecules, applications to molecular-wires type transition metal complexes:*
J. Neugebauer, M. Reiher *J. Phys. Chem. A* **108** (2004), 2053 – 2061.
- *systematic comparison of initial guesses, preconditioners, convergence characteristics, preconditioning schemes, near-degeneracies:*
M. Reiher, J. Neugebauer *Phys. Chem. Chem. Phys.* **6** (2004), 4621 – 4629.
- *applications to molecules adsorbed at surfaces:*
C. Herrmann, M. Reiher *Surf. Sci.* **600** (2006), 1891 – 1900.

- *QM/MM calculations:*
C. Herrmann, J. Neugebauer, M. Reiher *J. Comput. Chem.* **29** (2008), 2460 – 2470.

For further information on the semi-numerical implementation in our vibrational spectroscopy programs, see the following articles, which do also give references to important work by other groups:

- general review of theoretical vibrational spectroscopy for large molecules:
 - C. Herrmann, M. Reiher *Top. Curr. Chem.*, **168** (2007), **85 – 132**.
- parallelized, semi-numerical calculation of vibrational frequencies and intensities (in particular, Raman intensities [2nd paper]):
 - J. Neugebauer, M. Reiher, C. Kind, B. A. Hess. *J. Comput. Chem.*, **23** (2002), 895 – 910.
 - J. Neugebauer, M. Reiher, B. A. Hess. *J. Chem. Phys.*, **117** (2002), 8623 – 8633.
- anharmonicity effects in the density functional framework (1st paper) and error cancellation of harmonic BP86/TZVP frequencies in comparison with fundamental frequencies (both papers below):
 - J. Neugebauer, B. A. Hess. *J. Chem. Phys.*, **118** (2003), 7215 – 7225.
 - M. Reiher, G. Brehm, S. Schneider. *J. Phys. Chem. A*, **108** (2004), 734 – 742
- mode-wise calculation of intensities for vibrational spectra:
 - M. Reiher, J. Neugebauer, B. A. Hess. *Z. Phys. Chem.*, **217** (2003), 91 – 103.
 - J. Neugebauer, M. Reiher, B. A. Hess, in: S. Wagner, W. Hanke, A. Bode, F. Durst (Eds.), *High-Performance Computing in Science and Engineering 2000-2002*, Springer-Verlag, Berlin **2002**, pp. 157 – 169.

2. Quickstart

The program package AKIRA has been developed to implement the Mode-Tracking idea. Selected vibrations (normal modes and wavenumbers) can be obtained using the harmonic approximation. The vibrational frequencies are determined using numerical differentiation of analytic gradients of the total electronic energy with respect to collective Cartesian nuclear coordinates. AKIRA requires single-point calculations with either DALTON or TURBOMOLE or ADF or GAUSSIAN, which can be performed using coarse-grained parallelization (PVM and MPI) with automatic load-balancing. Note that you need to possess an official licence for any of these quantum chemistry packages!! AKIRA does not intermingle with any of these programs but only scans the output of them in order to extract all relevant raw data for the Mode-Tracking protocol. AKIRA will automatically start DALTON, TURBOMOLE, ADF, or GAUSSIAN single-point jobs on slave nodes (if no PC cluster is available it is possible to run AKIRA in a single-processor mode). For the easy set up and handling of the calculations you may start the set-up tool AKIRADefine. Normal modes may be tracked for any electronic structure method implemented in DALTON, TURBOMOLE, ADF, or GAUSSIAN for which analytic energy gradients are available.

To install Akira, you need the following steps:

- unpack: `tar -xvjf akira-3.4.0.tar.bz2`
- change to subdirectory: `cd akira-3.4.0`
- read help: `less INSTALL`
- configure package: `./configure --with-pvm --with-scp`
- compile package: `make -j2`

If you encounter any problems with the installation: you might install the SNF package first in order to make sure that the PVM or MPI libraries, resp., work properly! If SNF is running on your system, AKIRA also will!

3. Installation and technical issues

3.1 Getting started

3.1.1 General: the commands in the bin and scripts directory

In the bin directory of the akira directory, you will find two binaries, `akiradefine` and `akira`. You will call `akiradefine` in order to set up the modetracking calculation, and `akira` in order to perform the calculation.

In the scripts directory, the following commands are available:

script	use of script	needs ...	produces ...
<code>mkrdcinput</code>	specifies paths and directories for <code>akira</code>	—	DCINPUT
<code>mkparainput</code>	specifies paths and directories for <code>akira</code> if the parallelized version of TURBOMOLE is to be used	—	DCINPUT
<code>choose_nodes</code>	checks cluster for free nodes and writes them to file <code>USE_NODES</code> (only necessary in pvm version)	ELIGIBLE, runtime	USE_NODES
<code>clear_pvm</code>	kills the user's pvm daemons and removes temporary files created by AKIRA on the slave nodes (only necessary in pvm version)	—	—
<code>dcmore</code>	monitors AKIRA calculation by displaying <code>TMPdcstat</code> file, updating it by default every 5 seconds. The update time can be set via <code>dcmore <val></code> , <code><val></code> = update time in seconds.	—	—

In order to assure that `akiradefine` finds these scripts, you should add the `akira/scripts` directory to the `$PATH` in your `.bashrc` or copy the scripts into the bin directory in your home.

`mkrdcinput` and `choose_nodes` are called automatically by `akiradefine`. For further information on how to monitor a calculation with `dcmore`, see section 3.5.

The `clear_pvm` script contains a list of all machines on which the user's pvm daemons still running due to a previous run of a parallelized program

using PVM shall be killed and temporary files created in a previous AKIRA run shall be removed:

```
for m in computer1 computer2 computer3 computer4\
; do
echo ----- $m -----
ssh $m rm -r 'ssh $m ls -d '/tmp/TMP*' '
'ssh $m ls -d 'TMPDIR/<username>/TMP*' '
'ssh $m ls -d 'TMPDIR/TMP*'
' TMPDIR/pvmd.<uid> TMPDIR/pvml.<uid>
TMPDIR/file* TMPDIR/file* TMPDIR/pvmtmp*
ssh $m killpvmd3
done
```

The machines list must be modified by the user and should contain all machines listed in the ELIGIBLE file. Furthermore, <uid> must be replaced by the user ID, and TMPDIR by the scratch directory which is defined in the DCINPUT file. Of course, if the user has chosen for the temporary directories on the slaves a prefix which does not start with the default TMP, TMP* has to be replaced, too.

The ELIGIBLE file is searched by default in your \$CIPROC path. Examples for all in- and output files mentioned here can be found in the examples/files subdirectory in the akira directory. Furthermore, a detailed explanation of the DCINPUT file and the choose_nodes command is given in the subsections 3.1.2 to 3.1.4.

As far as akira is concerned, all paths necessary for PVM and MPI, respectively, and for the quantum chemical package you want to use, that is TURBOMOLE, DALTON, ADF or GAUSSIAN, must be set correctly. In the examples/files subdirectory, you will find a bashrc file which contains an example of all paths that need to be set for AKIRA.

In the following three subsections, more detailed information will be given on the DCINPUT file and the preparations necessary for PVM and MPI, respectively.

3.1.2 The DCINPUT file

This file has the following structure:

```
DCPATH /usr/bin/akira
MYDIR /home/<usrname>/calculations/example
PREFIX TMP<usrname>%
TMPDIR /tmp
LOGDIR /home/<usrname>/calculations/example/log
```

with the variables

- DCPATH: path to the AKIRA executable file (optional)
- MYDIR: your working directory (mandatory)
- PREFIX: prefix for logfiles and temporary files (optional)
- TMPDIR: directory for temporary files (including all temporary files of the single point calculations); must be available on every machine (optional)
- LOGDIR: directory for logfiles (optional)

If the optional variables are not specified in DCINPUT, AKIRA will try to use \$MYDIR/tmp as TMPDIR and \$MYDIR/log as LOGDIR. PREFIX will be set to "TMP".

The variables DCPATH, TMPDIR, and LOGDIR may be specified individually for some (or all) of the slave nodes. This is achieved by the entry

```
DCPATH(<nodename>) <pathname>
```

etc. Additional optional variables are

- PROGDIR: path to the TURBOMOLE, DALTON or GAUSSIAN executable files (note that ADF commands will always be taken from the path which is set in your shell).
- ARPATH: path for archiving of mos files etc. This option has been disabled.
- CHOOSENOPT <opt>: option for choose_nodes call; in particular, the following options are possible for the latest version of choose_nodes:
 - -e <eligfile>: specify alternative path/name of ELIGIBLE file (default: \$CIPROC/ELIGIBLE)
 - -local: use only one process on local host in parallel machinery
 - -nlocal <n>: use <n> processes on local host in parallel machinery
 - -ignore: leave USE_NODES file unchanged

If you want to perform calculations with values for the variables cstep or scfconv, which are out of the confidence interval (AKIRADEFINE will inform you if this happens), the entry

```
CRAP_OK yes
```

must be added to DCINPUT.

To create a standard inputfile, you can use the script mkrdcinput, which is automatically executed by AKIRADEFINE.

This file should be executed in the working directory. The settings for `DCPATH` and `TMPDIR` in this script must be adopted to the local settings before using it.

3.1.3 Preparations for the PVM version

To run the PVM version of AKIRA, it is necessary to execute the script `choose_nodes`, which selects the nodes for the parallelized calculation from the file `SCIPROC/ELIGIBLE`. It contains the names of the computers in the “parallel virtual machine” and the numbers of processors available on each computer, e.g.,

```
computer1 1
computer2 1
computer3 2
computer4 2
#computer5 2
```

Note that this default setting can be modified by passing options to `choose_nodes`, see Section 3.1.2. By using the script `choose_nodes`, all processors which are idle will be added to the list in `USE_NODES`. The latter file is of the form

```
# 4 nodes with 6 processors are eligible.
# parallel environment is pvm
# Thu Oct  4 12:24:01 MEST 2001
node computer1 1 1
node computer2 2 2
node computer3 1 2
# chosen 3 machines with 4 processors.
```

Both `choose_nodes` and `mkrdcinput` are executed by `akiradefine`, such that no further preparations are necessary if the file `mkrdcinput` is in line with the local settings (path containing the `akira` executable, `TMPDIR`). After these preparations, AKIRA can be started using the command `akira`.

3.1.4 Preparations for the MPI versions

If one of the MPI versions shall be used, this can be done by the command

```
mpirun -machinefile MACHINES -np <number of CPU's> akira
```

The file `MACHINES` must contain the names of all computers available for the parallel calculation. You may create this file by running

```
choose_nodes -mpi
```

and then renaming the file `USE_NODES`, which is created by `choose_nodes`, to `MACHINES`. This is done automatically by `AKIRADEFINE`, which also creates the runscript `mpi.sub`. The runscript contains the `mpirun` command mentioned above; the number of CPU's is extracted from the files `USE_NODES`. Hence, after running `AKIRADEFINE` you may immediately run the MPI version of `AKIRA` using the command `mpi.sub`. Note that it might be necessary to modify the `mpi.sub` runscript, since different MPI installations may use different options.

Note that the exact syntax of the `MACHINES` file and the `mpirun` command depends on the MPI version. The syntax above corresponds to that of `MPICH`. `OPENMPI`, for instance, follows a slightly different syntax.

3.2 Parallelization standards

`AKIRA` uses raw data from several structures displaced from a reference structure along a set of basis modes, which start with a (or a couple of) user-defined basis vector(s). Each basis vector represents a collective distortion of the cartesian coordinates of the molecular (minimum) structure. The program executes two single-point calculations of electronic energies and their gradients for each basis vector employed in each iteration. Although parallelization is not as vital as in normal frequency analyses [3], the total wall clock time can be considerably reduced if these calculations are executed simultaneously in a coarse-grained parallel manner. In order to use the parallelized version of the data collection in `AKIRA`, it is necessary to provide the software for one of the following parallelization standards. Besides these, note that a *serial version of the program is also available*.

3.2.1 PVM

To run the PVM version of `AKIRA`, it is necessary to install PVM. It can be obtained from

http://www.epm.ornl.gov/pvm/pvm_home.html

It is necessary to set the following environment variables for PVM:

`PVM_ROOT`: PVM installation directory

`PVM_ARCH`: architecture of the computer (`LINUX{,64}` or `ALPHA`)

`AKIRA` will look for the file `fpvm3.h` in `$PVM_ROOT/include` and for PVM libraries in `$PVM_ROOT/lib/$PVM_ARCH`

3.2.2 MPI

Different implementations of the MPI standard are available. OPENMPI, MPI, MPICH, and LAM/MPI, resp., may be obtained from

```
http://www.open-mpi.org,  
http://WWW.ERC.MsState.Edu/misc/mpi/  
http://www-unix.mcs.anl.gov/mpi/mpich, or  
http://www.lam-mpi.org/.
```

We use OPENMPI for development and, therefore, recommend its usage. AKIRA will look for a MPI Fortran compiler (like `mpif90`) and will use it for compilation.

3.3 Memory management and pre-processing

In the latest AKIRA version, program-specific setup tools and memory management constructs were reduced to a minimum. In particular, dynamic memory allocation is now handled by default FORTRAN90 `allocate/deallocate` calls.

Earlier versions of AKIRA used a memory management that relied on the MEMMGR module by B. A. Hess [7], which is also used in SNF [3]. The MEMMGR allows dynamical allocation of memory within FORTRAN routines which are written entirely in FORTRAN77. These versions also employed the preprocessor DELREM by B. A. Hess [8].

3.4 Installation of AKIRADEFINE and AKIRA

AKIRA comes as a source code tar-ball. To install it, you first need to unpack the archive using

```
tar -xvjf akira-3.4.0.tar.bz2
```

In the `akira-3.4.0` directory you will find a file `INSTALL` which tells you how to compile and install the package. You will need at least these steps:

```
./configure  
make
```

To enable parallel calculations you need to give certain options to `configure` which are described in `INSTALL`. It may also be helpful to look at the output of `./configure --help`.

3.5 Monitoring AKIRA calculations

Most important to observe the progress of the calculation is the file `TMPdcstat` which contains information about the status of each slave process. The file

3.6. FURTHER PROGRAMS AND SCRIPTS NEEDED BY AKIRA 19

can be watched during the calculation by executing the script `dcmore` which can be found in the scripts directory. It consists of 11 columns:

```
2  1 computer1  vbsy      0  90005  1 rdy      1  1 results sent
3  2 computer2                0  90005  1 rdy      2  2 results sent
4  3 computer3                0   103  0 run      3  3 968:15 dscf
5  4 computer4                0  90005  1 rdy      4  4 results sent
```

The rows contain the following entries:

- 1 task ID
- 2 continous numbering
- 3 node name
- 4 entry `vbsy` indicates a “very busy” machine
- 5 process ID (0 for finished processes)
- 6 cpu-load times 100, or
90005 for finished calculations, or
70005 for trouble in TURBOMOLE programs, or
80005 for wrong results.
- 7 number of finished single point calculations on this node
- 8 status:
`rdy` = ready for next step,
`run` = running calculation,
`wait` = node waiting,
`pvme` = PVM error
- 9 step number
- 10 step number (redundant)
- 11 cpu time and name of running program or message

As already indicated by the name of the data file, AKIRA has got a restart facility, such that any partly completed `restart.akira` file can be supplied in a AKIRA run, if the program has been aborted.

3.6 Further programs and scripts needed by AKIRA

As a meta-program, AKIRA starts standard quantum chemical packages on slave nodes in order to produce the raw data. The output produced on the slave nodes is then scanned in order to pick up all necessary information needed by AKIRA. Thus, AKIRA does not intermingle with any of these programs and you do need a license to run these programs, which you have to acquire separately according to the conditions of the particular vendor or theoretical chemistry groups, respectively!

For the performance of the single point calculations, the programs of the ADF [9], DALTON [10], GAUSSIAN [11] or TURBOMOLE [12] package, respectively, must be available on every node in `$PATH`. AKIRADefine allows to perform semiempirical MOPAC [13] calculations automatically as an initial guess for normal modes and Hessians. If this feature shall be used,

MOPAC must be installed on your system. AKIRADefine furthermore requires MOLDEN [14] if the initial guesses selected for the mode-tracking calculations shall be visualized.

Furthermore, the scripts from the `scripts` subdirectory of the AKIRA installation directory must be available in `$PATH`. The script `choose_nodes` also requires the file `ELIGIBLE` in the directory `$CIPROC`, which contains the names of computers available and their numbers of processors (see Chapt. 5). An example file is provided with the AKIRA package. The variable `$CIPROC` must be set by the user. `choose_nodes` also requires the shell-command `ruptime` which comes with the `rwho` package. *Therefore, you need to install this package first as it is essential for the load balancing with PVM.* This is only necessary for the parallel version of AKIRA.

4. Methodology

The first section of this chapter explains the theoretical background of the subspace iteration methods employed in AKIRA. The next section deals with details of the numerical derivative methods.

4.1 Subspace iteration techniques

Two different subspace iteration techniques are available in AKIRA, namely the Davidson and Lanczos algorithms, which will be explained in the next subsections (compare also the reference to the original work given above).

4.1.1 Davidson algorithm

In order to calculate the vibrational frequencies, we have to solve the eigenvalue equation

$$\mathbf{H}^{(m)}\mathbf{Q}_k = \lambda_k\mathbf{Q}_k, \quad (4.1)$$

where $\mathbf{H}^{(m)}$ is the mass-weighted Cartesian Hessian, which contains the (mass-weighted) second derivatives of the total electronic energy with respect to nuclear Cartesian coordinates, and $\{\lambda_k, \mathbf{Q}_k\}$ is the eigensystem to be determined (with $\lambda_k \sim \omega_k^2$ and ω_k being the k th vibrational frequency; see [15,16]).

The conventional procedure is to calculate *all* elements of the matrix $\mathbf{H}^{(m)}$ (either analytically or numerically) and to diagonalize this matrix to obtain all $3N$ eigenvalues and eigenvectors for a molecule containing N atoms. If only selected vibrations are of interest, one can apply subspace iteration methods like those by Lanczos [6] or by Davidson [4]. This has the major advantage that the full Hessian need not be calculated, which is the time limiting step in the standard procedure.

Our Davidson-type method starts with a collective displacement \mathbf{b} of all atoms

$$\mathbf{b} = \sum_{j=1}^{3N} b_j \mathbf{e}_j^{(m)}, \quad (4.2)$$

where $\mathbf{e}_j^{(m)}$ are the $3N$ (mass-weighted) nuclear Cartesian basis vectors, and b_j are the components of the displacement. The k th elemof the vector

$\boldsymbol{\sigma} = \mathbf{H}^{(m)} \cdot \mathbf{b}$, which is the first approximation to the left-hand side of Eq. (4.1), is then given as

$$\sigma_k = \{\mathbf{H}^{(m)} \cdot \mathbf{b}\}_k = \sum_l \frac{\partial^2 E}{\partial R_l^{(m)} \partial R_k^{(m)}} b_l = \frac{\partial^2 E}{\partial R_k^{(m)} \partial \mathbf{b}}. \quad (4.3)$$

$\partial^2 E / [\partial R_l^{(m)} \partial R_k^{(m)}]$ is the second derivative of the total electronic energy with respect to (mass-weighted) nuclear Cartesian coordinates. This relation allows us to calculate the vector $\boldsymbol{\sigma}$ as a numerical directional derivative of the gradient of the total electronic energy E with respect to the collective displacement \mathbf{b} ,

$$\boldsymbol{\sigma} = \mathbf{H}^{(m)} \cdot \mathbf{b} = \begin{pmatrix} \sum_l \frac{\partial^2 E}{\partial R_1^{(m)} \partial R_l^{(m)}} b_l \\ \sum_l \frac{\partial^2 E}{\partial R_2^{(m)} \partial R_l^{(m)}} b_l \\ \vdots \\ \sum_l \frac{\partial^2 E}{\partial R_{3N}^{(m)} \partial R_l^{(m)}} b_l \end{pmatrix} = \begin{pmatrix} \frac{\partial^2 E}{\partial R_1^{(m)} \partial \mathbf{b}} \\ \frac{\partial^2 E}{\partial R_2^{(m)} \partial \mathbf{b}} \\ \vdots \\ \frac{\partial^2 E}{\partial R_{3N}^{(m)} \partial \mathbf{b}} \end{pmatrix}. \quad (4.4)$$

The vector $\boldsymbol{\sigma}$ can thus be calculated as the numerical derivative of the analytic gradients of the total energy. For this numerical differentiation it is necessary to carry out single point calculations for the along- \mathbf{b} distorted structures such that n -point central difference formulae [17] for the numerical finite-difference approximation of the second derivative can be applied. For the generation of these distorted structures, we use displacements which result in a preselected norm of the corresponding (non-mass-weighted) Cartesian displacement vector; in general, a step size of 0.01 bohr proved to yield reliable and numerically stable derivatives [2, 3].

In the i th subspace iteration we build the Davidson matrix $\tilde{\mathbf{H}}^{(m),i}$ as

$$\tilde{\mathbf{H}}^{(m),i} = \mathbf{B}^{iT} \mathbf{H}^{(m)} \mathbf{B}^i = \mathbf{B}^{iT} \boldsymbol{\Sigma}^i \quad (4.5)$$

where all vectors \mathbf{b}^l and $\boldsymbol{\sigma}^l$ (with $l = 1, \dots, i$ and i being the actual iteration step) are collected in the matrices \mathbf{B}^i and $\boldsymbol{\Sigma}^i$, respectively. We then solve the eigenvalue problem for the small Davidson matrix,

$$\tilde{\mathbf{H}}^{(m),i} \mathbf{c}_\mu^{(i)} = \lambda_\mu^{(i)} \mathbf{c}_\mu^{(i)}, \quad (4.6)$$

where $\lambda_\mu^{(i)}$ is the i th approximation for eigenvalue λ_μ , from which we can calculate approximate wavenumber in every iteration step. The desired eigenvector \mathbf{c}_μ^i is selected from the set of vectors obtained from Eq. (4.6) and the residuum vector reads

$$\mathbf{r}_\mu^{(i)} = \sum_{l=1}^i c_{\mu,l}^{(i)} [\boldsymbol{\sigma}^l - \lambda_\mu^{(i)} \mathbf{b}^l], \quad (4.7)$$

(note that i always denotes the actual i th iteration and μ marks the selected vector). The sum is over all basis vectors \mathbf{b}^l , and the number of basis vectors is increased in each iteration. In the standard Davidson method, the number of basis vectors is equal to the number of iterations, since in each iteration one new basis vector is introduced. For each new basis vector \mathbf{b}^{i+1} , we obtain a new vector $\boldsymbol{\sigma}^{i+1}$ as the numerical derivative of the gradient with respect to the collective displacement \mathbf{b}^{i+1} . The i th approximation \mathbf{v}_s^i to the exact eigenvector \mathbf{q}_s in Eq. (4.1) is obtained as

$$\mathbf{Q}_\mu^{(i)} = \sum_{j=1}^i c_{\mu,j}^{(i)} \mathbf{b}^j, \quad (4.8)$$

4.1.2 Preconditioning: The Davidson algorithm

The new basis vectors are generated from the residuum vectors,

$$\mathbf{b}^{i+1} = \mathbf{D}^{-1,(i)} \mathbf{r}_\mu^{(i)}, \quad (4.9)$$

where $\mathbf{X}^i = \mathbf{D}^{-1,(i)}$ is a preconditioner, which should ideally be as close as possible to $[\mathbf{H}^{(m)} - \lambda_\mu^{(i)} \mathbf{1}]^{-1}$. The simplest approximation for the inverse matrix of $[\mathbf{H}^{(m)} - \rho_s^i \mathbf{1}]$ is to use a diagonal preconditioner with diagonal elements $X_{jj} = 1/(H_{jj}^{(m)} - \rho_s^i)$. However, this is only a good approximation for diagonally dominant matrices, a condition which is fulfilled for configuration interaction matrices, but not for the Hessian matrices investigated here. This procedure is repeated until the convergence criterion drops below a predefined threshold. Convergence criteria are: i) the maximum element of the residuum vector, ii) the norm of the residuum vector, iii) the contribution $u_{s,i}^i$ of the latest basis vector i in Eq. (4.8) to the selected eigenvector and iv) the change in the wavenumber.

The convergence characteristics of this algorithm strongly depend on the reliability of i) the initial guess of the first basis vector \mathbf{b}^1 , which is the first approximation to the desired exact eigenvector \mathbf{q}_s and of ii) the preconditioner. The latter problem is delicate since we do not have any information about the matrix $\mathbf{H}^{(m)}$; only matrix–vector products $\boldsymbol{\sigma}^l = \mathbf{H}^{(m)} \mathbf{b}^l$ are known.

The Hessian may be approximated using the inverse transformation of Eq. (4.5)

$$\mathbf{H}^{(m)} = \mathbf{B} \tilde{\mathbf{H}}^{(m)} \mathbf{B}^T, \quad (4.10)$$

with $\mathbf{B} := \mathbf{B}^{3N}$. This transformation would thus only be exact if we used a complete set of $3N$ basis vectors. If the basis set is not complete, we may use the approximation

$$H_{nj,\text{appr.}}^{(m)} = \sum_{kl} \tilde{H}_{kl}^{(m),i} B_{kn}^i B_{lj}^i \quad (4.11)$$

for the default preconditioner, where the sum is over all basis vectors \mathbf{b}^l ($l = 1, \dots, i$) stored in the matrix \mathbf{B}^i . But this is usually a poor approximation and yields only as many approximate diagonal elements as basis functions are used in the current iteration (for the other diagonal elements, one could use either unit entries or the last diagonal element determined in this way for all other diagonal entries). However, the more iterations are needed, the better becomes the preconditioner in this default preconditioning scheme. Furthermore, $1/(H_{jj}^{(m)} - \lambda_\mu^{(i)})$ is a poor approximation to the inverse of a matrix $(\mathbf{H}^{(m)} - \lambda_\mu^{(i)}\mathbf{1})$ if $\mathbf{H}^{(m)}$ is not diagonally dominant. Consequently, this approach is in most cases not better than using a unit matrix as a preconditioner at the very beginning of the procedure, when only very few basis vectors are available.

Both problems mentioned above in connection with the convergence criteria can be overcome by using a semi-empirical calculation as an initial approximation: We calculate an estimate for the Hessian and approximate normal modes using the PM3 model (of course, other semi-empirical models can also be utilized). An initial guess for the eigenvector can be chosen from the set of semi-empirical normal modes, while the semi-empirical Hessian can be used for the preconditioning procedure. Since the Hessian matrices under investigation are of dimensions of about a few hundred rows and columns, it is — in contrast with configuration interaction matrices — possible to explicitly calculate the inverse preconditioner matrix

$$\mathbf{X}^i = \left[\mathbf{H}_{\text{PM3}}^{(m)} - \rho_s^i \right]^{-1} \quad (4.12)$$

in each iteration. It should be emphasized that the bottleneck of the calculation is not this matrix inversion, which takes only a couple of seconds, but the single point calculations of electronic energies and gradients for the displaced structures. Using a 3-point central differences formula [17] for the numerical differentiation, we need two single-point calculations for structures distorted along each basis vector, which are performed in a coarse-grained parallelized way using standard parallelization techniques as provided by PVM and MPI. Unfortunately, it is not possible to perform *all* single-point calculations at once as the basis vectors of iteration i depend on the results of all $(i-1)$ former iterations. Therefore, the little computational effort for the generation of more accurate preconditioners is easily compensated by the resulting reduction of the number of iterations.

In course of the calculation of $\mathbf{H}_{\text{PM3}}^{(m)}$, we also obtain the PM3 normal modes, which we use as the first approximation \mathbf{b}^1 . Note that this ‘guessing of normal modes’ is different from the standard projection operator technique, which always requires a certain point group in order to set up the projector from the irreducible representations of this point group. Instead, we project out a selected mode and do not rely on any group theoretical

tools. Consequently, our approach is applicable also in C_1 -symmetric cases. Nevertheless, these projection operator techniques can be used to determine an initial guess for the desired normal modes.

4.1.3 The Jacobi–Davidson algorithm

Let us take another look at the preconditioning problem: In a mode-tracking calculation however, we iteratively solve the equation

$$(\mathbf{H} - \lambda_\mu^{(i)})\mathbf{Q}_\mu^{(i)} = \mathbf{r}_\mu^{(i)} \quad (4.13)$$

With the residuum vector, we want to construct the correction $\Delta\mathbf{Q}$ to the current eigenvector approximation with

$$(\mathbf{H} - \lambda_\mu)(\mathbf{Q}_\mu^{(i)} - \Delta\mathbf{Q}) = \mathbf{r}_\mu^{(i)} - (\mathbf{H} - \lambda_\mu)\Delta\mathbf{Q} = \mathbf{0}. \quad (4.14)$$

This suggest that

$$\Delta\mathbf{Q} = (\mathbf{H} - \lambda_\mu)^{-1}\mathbf{r}_\mu^{(i)} = \mathbf{D}^{-1}\mathbf{r}_\mu^{(i)}, \quad (4.15)$$

where we used the definition $\mathbf{D} = (\mathbf{H} - \lambda_\mu)$. Since the purpose of applying subspace iteration techniques is to avoid the calculation and/or storage of the full Hessian matrix, the matrix \mathbf{D} is usually not known. The knowledge of this correction vector would allow us to use $\Delta\mathbf{Q}$ as the next basis vector, which should immediatly reduce the residual vector to zero.

The original Davidson procedure is mainly used in CI-type problems, where electronic energies are identified as eigenvalues of the CI-matrix. These matrices are strongly diagonally dominant, which means that an appropriate guess for them can be constructed by using the diagonal elements of the CI-matrix. Such guesses for \mathbf{D} are very successful for preconditioning and usually lead to rapid convergence. A guess for the eigenvalue λ_μ is readily available from the last iteration.

As described in earlier work, the Hessian matrix of a system is usually not diagonally dominant, and furthermore, a calculation of all diagonal elements of the Hessian is not much less work than the calculation of the full Hessian. But as mentioned above in many cases it is possible to get a guess for the Hessian of the system from simpler calculations, like semi-empirical, force-field or small basis set calculations. In these cases it is possible to construct the matrix \mathbf{D}^{-1} by direct inversion of the guess Hessian (minus the approximate eigenvalue),

$$\mathbf{D}^{-1} = (\mathbf{H}^{\text{guess}} - \lambda_\mu^{(i)}). \quad (4.16)$$

Note that here and in the following, we use the notation \mathbf{D} also for guesses of the exact definition given above.

As pointed out by Sleijpen and Van der Vorst [5], the Davidson diagonalization scheme has great difficulties if the guess for the Hessian becomes too good. Imagine that we apply the exact Hessian for preconditioning; in that case, the new basis vector would be obtained as

$$\mathbf{b}^{(i+1)} := (\mathbf{H} - \lambda_{\mu}^{(i)})^{-1} \mathbf{r}_{\mu}^{(i)} = \mathbf{Q}_{\mu}^{(i)}, \quad (4.17)$$

where we used Eq. (4.13) for the second equality. This means that the better the approximation for the Hessian is which we employ for preconditioning purposes, the smaller will be the angle between the new basis vector and the old eigenvector approximation. In the limit of the exact Hessian, no improvement at all will be obtained.

It is, however, difficult to decide what will happen in a practical mode-tracking calculation, since the new basis vectors are always orthogonalized to all preceding basis vectors, which should eliminate the problem of linear dependencies. And even for an exact Hessian, the numerical differentiation of the electronic gradient will introduce some numerical noise, so that always a component orthogonal to the current basis vectors will be present. This component might, however, be very small, and therefore we employ a cyclic procedure of Gram–Schmidt-orthogonalizations and orthogonality checks in our program in order to ensure orthogonality even in problematic cases. Whether the preconditioning is still efficient in those cases is a question which is investigated in the next section, since the orthogonal component of our (non-orthogonalized) new basis vector might just consist of numerical noise.

Sleijpen and Van der Vorst [5] proposed a Jacobi–Davidson diagonalization scheme, which automatically restricts the new basis vector to the subspace orthogonal to the current approximation $\mathbf{Q}_{\mu}^{(i)}$. This is achieved by using the orthogonal projection of the matrix to be diagonalized onto this subspace. Again, usually neither the matrix itself nor this orthogonal projection is available. Therefore, they suggest a one-step approximation to find a new basis vector if a guess for the matrix is available,

$$\mathbf{b}^{(i+1)} := \epsilon \mathbf{D}^{-1} \mathbf{Q}_{\mu}^{(i)} + \mathbf{D}^{-1} \mathbf{r}_{\mu}^{(i)}, \quad (4.18)$$

where

$$\epsilon = \frac{\mathbf{Q}_{\mu}^{(i)} \mathbf{D}^{-1} \mathbf{r}_{\mu}^{(i)}}{\mathbf{Q}_{\mu}^{(i)} \mathbf{D}^{-1} \mathbf{Q}_{\mu}^{(i)}} \quad (4.19)$$

The last equation is determined by the requirement that $\mathbf{b}^{(i+1)}$ is orthogonal to $\mathbf{Q}_{\mu}^{(i)}$. This should definitely fix the problem in Eq. (4.17), which might occur for the Davidson algorithm.

4.1.4 The Lanczos algorithm

The implementation of the Lanczos-type algorithm is very similar to the Davidson-diagonalization scheme. The first step is again the calculation of the vector $\boldsymbol{\sigma}^i$, Eq. (4.4), by numerical differentiation of elements of the gradient vector, calculated for structures perturbed along the basis vector \mathbf{b}^i . In the next step, the diagonal elements of the small Hessian matrix for the subspace are calculated,

$$d_i = \boldsymbol{\sigma}^{i,T} \mathbf{b}^i, \quad (4.20)$$

(note that the Lanczos algorithm is essentially a method to create a tridiagonal matrix

$$\mathbf{H}^{(m),\text{tridiag}} = \mathbf{B}^T \mathbf{H}^{(m)} \mathbf{B} = \begin{pmatrix} d_1 & t_1 & 0 & 0 & \cdots \\ t_1 & d_2 & t_2 & 0 & \cdots \\ 0 & t_2 & d_3 & t_3 & \cdots \\ 0 & 0 & t_3 & d_3 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \quad (4.21)$$

from the original matrix $\mathbf{H}^{(m)}$). With these quantities, the vector

$$\mathbf{x}_{i+1} = \boldsymbol{\sigma}_i - d_i \mathbf{b}_i - t_{i-1} \mathbf{b}_{i-1}, \quad (4.22)$$

is calculated, which in turn determines the elements t_i via

$$t_i = |\mathbf{x}_{i+1}|, \quad (4.23)$$

and the new basis vector,

$$\mathbf{b}_{i+1} = \frac{\mathbf{x}_{i+1}}{|\mathbf{x}_{i+1}|}. \quad (4.24)$$

Note that $t_0 = 0$ in the first iteration. Furthermore, the new basis vector \mathbf{b}_{i+1} is usually explicitly orthonormalized to the set of all previous basis vectors to avoid (near-)linear dependencies. The disadvantage of the Lanczos algorithm is that the new basis vector is in no way preconditioned for better convergence of the eigenvector selected for optimization. Therefore, the convergence characteristics is usually better for the Davidson diagonalization, unless only very poor preconditioners are available (see below).

Calculation of approximate force constants, wavenumbers and normal modes in each iteration are carried out in exactly the same manner as for the Davidson diagonalization. This also holds for the residuum vectors, which are only necessary for convergence control in the case of the Lanczos method.

4.1.5 Details of the implementation

We have implemented the above-described subspace iteration with a Davidson, a Jacobi–Davidson, as well as a Lanczos solver in the AKIRA program. A comparison of both diagonalization schemes shows that they perform equally well if the preconditioning is not well chosen (see below). But, in case of a good preconditioning through a PM3 or similarly sophisticated guess, we obtain a significantly better convergence of the Davidson-type algorithm. Our implementation allows one to optimize several eigenvectors simultaneously, which is known as the *Davidson–Liu* or *block-Davidson* method [18, 19]. Root homing is also guaranteed [20]. For root homing, there exist two promising protocols in the case of normal modes as eigenvectors: i) selection of the eigenvector with the largest overlap with the initial guess vector; ii) selection of the eigenvector with the largest overlap with the approximate eigenvector chosen in the last iteration. Both methods are implemented in our program. While the first method can cause convergence problems if only a poor initial guess vector is available, the second method usually shows better convergence characteristics; however, it may converge to a different, non-desired eigenvector due to poor initial vectors in combination with some preconditioners (see [1] for examples) Furthermore, it is possible to select the vector with minimal residuum or to optimize the lowest root.

4.2 Generation of displaced structures

We use displacements along the mass-weighted basis vectors \mathbf{b}^i , for which the energies and gradients have to be calculated. A displacement along the mass-weighted basis vector of $\Delta\mathbf{b}^i$ gives rise to a displacement in Cartesian coordinates of

$$\Delta\mathbf{R}^i = \Delta\mathbf{b}^i\mathbf{M}^{-1/2}, \quad (4.25)$$

where \mathbf{M}^{-1} is a diagonal matrix with elements $M_{ij}^{-1} = \delta_{ij}/m_i$; m_i is the mass of the atom corresponding to the Cartesian coordinate R_j^i .

4.2.1 Displacements in units of length

Test calculations have shown that the step size s_R should be chosen such that the norm of the Cartesian displacement vector is $s_R|\Delta\mathbf{R}^i| \approx 0.01$ bohr. The displacement may be re-written in terms of normalized displacement vectors for both the Cartesian and the mass-weighted basis modes,

$$s_{Q_k}\Delta\mathbf{b}^{i,\text{norm}} \hat{=} s_{Q_k}\Delta\mathbf{R}^i = s_{Q_k}|\Delta\mathbf{R}^i|\Delta\mathbf{R}^{i,\text{norm}} = s_R\Delta\mathbf{R}^{i,\text{norm}} \quad (4.26)$$

which leads to a step size s_{b^i} for the numerical differentiation of

$$s_{b^i} = s_R / |\Delta \mathbf{R}^i| = s_R \left(\sum_{j=1}^{3N} (b_j^{i,\text{norm}})^2 / m_j \right)^{-1/2} \left(\frac{[\text{unit of length}]}{[\text{unit of mass}]^{1/2}} \right). \quad (4.27)$$

The inclusion of the units is necessary in order to keep both coefficients dimensionless (compare also Ref. [21]). Note that for a given value of s_R , the coefficient s_{b^i} may have different values for different normal coordinates b^i .

4.2.2 Calculation of second derivatives

The calculation of the elements of the $\boldsymbol{\sigma}$ vector, Eq. (4.3), can be accomplished by calculating numerical derivatives of the components of the analytic gradient w.r.t. the collective displacements along the basis vectors \mathbf{b}^i ,

$$\frac{\partial^2 E}{\partial R_k^{(m)} \partial \mathbf{b}} = \frac{1}{2s_{b^i}^2 |\Delta \mathbf{b}^{i,\text{norm}}|} \left[g_k^{(m)}(+s_R \Delta \mathbf{R}^{i,\text{norm}}) - g_k^{(m)}(-s_R \Delta \mathbf{R}^{i,\text{norm}}) \right] \quad (4.28)$$

where the $g^{(m)}$ are the mass-weighted components of the gradient vector for the along- \mathbf{b}^i displaced structures.

5. Running the calculation

In this chapter, the steps necessary to carry out a mode-tracking calculation using AKIRA will be described in detail. Starting from the preparations, we will discuss the setup of the calculation using AKIRADEFINE, the mode-tracking calculation itself, and possible restart and re-evaluation runs.

Running a mode-tracking calculation with AKIRA requires two separate steps: (I) choosing the initial guess vector(s) with AKIRADEFINE and (II) running the mode-tracking calculation with AKIRA. The calculation of intensities, which are obtained from numerical differentiation of property tensors along the converged normal modes, is not available in AKIRA 3.0.0 but will be possible soon via a separate module (note that no re-calculation of normal modes will be necessary then because the new tool will create displaced structures for the numerical differentiation from a previous AKIRA run!).

5.1 Preparations

5.1.1 Structure optimization

Since AKIRA tries to determine the vibrational frequencies and normal modes from the eigenvalues and eigenvectors of the Hessian matrix, and since these quantities have, in a strict sense, a well-defined meaning only for structures for which the electronic energy gradient is zero, it is necessary to perform a geometry optimization first. The mode-tracking calculation must then start from the optimized structure. A large number of test calculations with SNF (see Introduction) has shown that the maximum component of the gradient should be smaller than 0.0001 a.u. in order to obtain accurate results for the vibrational frequencies.

5.1.2 Electronically excited states

From AKIRA 2.1.0 on, mode-tracking calculations can be performed for electronically excited states, provided TURBOMOLE (version 5.6 or higher) is used for the single-point calculations. Of course, for the reasons mentioned in Section 5.1.1, the molecular structure has to be optimized in the chosen excited state before starting the AKIRA program. In this case, no further AKIRA-specific settings have to be made, since AKIRA will recognize the

excited-state keywords in the TURBOMOLE control file and use these settings automatically for the single-point calculations of the distorted structures on the slave nodes.

5.1.3 Molecular symmetry

AKIRA uses a semi-numerical algorithm in which second derivatives of the electronic energy are calculated as numerical first derivatives of analytical energy gradients. Therefore, the algorithm will create structures which are displaced from the equilibrium structure along some basis vectors. In almost all cases, these displacements will lower the molecular symmetry, which can cause problems if the original calculation explicitly used the higher symmetry, since the MOs of the original equilibrium structure are in some cases used by AKIRA as an initial guess for the MOs of the displaced structures.

For TURBOMOLE users: If the molecule under investigation is of higher symmetry than the trivial C_1 point group symmetry, the user should run a single-point energy calculation without taking advantage of the symmetry in order to provide C_1 -symmetric MOs as start-MOs for the perturbed structures of the molecule. As an alternative, you may provide any other initial guess for the MOs (which can, e.g., be generated by TURBOMOLE's `define` program).

There are indeed a few special cases in which the calculations for the displaced structures can be performed using a higher symmetry: If the basis vectors given in the setup preserve a higher symmetry than C_1 , and if also all new vectors generated in the mode-tracking calculation are guaranteed to keep this symmetry, then also each single-point calculation can indeed use this higher symmetry. To give the most important example: If you use AKIRA not to track a specific vibration, but to calculate all vibrations in a certain irrep, then it is possible to restrict the calculation to basis vectors of this irrep. If you want to calculate, e.g., all vibrations in the totally symmetric irrep of the molecule, than all displaced structures will also exhibit the original symmetry. In that case, the original MOs can be used as a guess. Another example is the totally symmetric breathing mode of the buckminsterfullerene C_{60} , for which the calculations for the perturbed structures can also be done in I_h symmetry. In this case, only one basis vector is needed to achieve convergence (see [1]).

It is *not necessary* to provide MOs in the case of ADF, DALTON or GAUSSIAN single point calculations.

5.2 AKIRADefine: setting up the calculation

For an easy preparation of the AKIRA calculation, most of the input/output and program options can be controlled and set via the interactive setup

tool AKIRADEFINE. All steps done by this program are explained here in detail. If your PATH variable contains the directory `akira/bin`, you can start AKIRADEFINE simply by typing:

```
akiradefine
```

The information which is collected by AKIRADEFINE is written to the file `akira_control` .

5.2.1 Program-specific input

Depending on whether TURBOMOLE, ADF or GAUSSIAN has been used for the molecular structure optimization, AKIRA can perform the single-point calculations for the displaced structures with different programs. If the geometry has been optimized with TURBOMOLE, it is possible for the user to request either a TURBOMOLE calculation (HF, DFT, RI-DFT, MP2, RI-MP2) or a DALTON calculation (HF, MP2, CC2, CCSD, CASSCF) for the perturbed structures. In case of a GAUSSIAN geometry optimization, GAUSSIAN is always employed in the single-point calculations. and similarly, in case of ADF optimizations, AKIRA will automatically use ADF for the perturbed structures.

TURBOMOLE input files

If TURBOMOLE or DALTON are to be used for the single-points subsequent to a TURBOMOLE structure optimization, all files which are necessary for TURBOMOLE single-point calculations must be provided (i. e. `control`, `coord`, `mos` or `alpha` and `beta`, `basis` and, for RI-accelerated calculations, `auxbasis`).

GAUSSIAN input file

If GAUSSIAN shall be used for the single-point calculations, the user must provide a GAUSSIAN input file named `akira.com` which contains the keyword `#p force`, the keywords for the method and the basis set to be used, information on molecular charge and spin state and the equilibrium geometry in cartesian coordinates. For an ethanol molecule, the `akira.com` file might look like this (with the last line being a blank line, of course):

```
#p force bp86/TZVP

ethanol

0 1
C -1.225747   -0.225340   0.000002
H -1.296072   -0.865520  -0.891479
H -1.296085   -0.865499   0.891496
H -2.083095    0.464915  -0.000013
```

```

C  0.079992    0.551234   -0.000000
H  0.131474    1.203598    0.893871
H  0.131470    1.203596   -0.893875
O  1.161283   -0.399398   -0.000003
H  1.996579    0.098737    0.000020

```

This file is also contained in the directory `examples/files`.

ADF input file

Since AKIRA will normally be run after performing a geometry optimization, it is designed to reuse simply the ADF script which has been used to run the geometry optimization. This script has to fulfill several conditions:

1. It must be named `adf.in`.
2. It be executable.
3. It must contain exactly the settings used in the preceding geometry optimization.
4. The geometry must be specified in cartesian coordinates *using Angström*
5. The binary output file must be named `TAPE21`.
6. It should not contain any blocks requesting other calculations than a geometry optimization.

AKIRADefine will check (and modify, if necessary) the following blocks in the `adf.in` file:

- `GEOMETRY` — delete it (if present) and write instead:

```

Geometry
  GO
  iterations 1
End

```

- `INTEGRATION` — if the grid settings are below `6.0 6.0`, replace them by `6.0 6.0`.
- `SCF` — if the convergence criteria are below `converge 1.0e-6 1.0e-6`, replace them by `converge 1.0e-6 1.0e-6`.
- Sort the list of cartesian coordinates according to their nuclear charge. (This will facilitate reading in the data of the distorted structure-single point calculations in the AKIRA run.) **Attention: the reordering of atoms will affect the choice of internal coordinates in AKIRADefine as well as the usability of ADF restart files!**

If AKIRAEFINE modifies the integration or convergence parameters, it will print a warning message to the screen remembering the user to redo a geometry optimization with the new settings.

Starting from version 3.1.0, AKIRA can deal with QM/MM gradients. Furthermore, it is now possible to provide an ADF restart file, which is distributed onto the slave nodes in order to accelerate the single-point calculation for the displaced structures. Based on the keywords in the `adf.in` input file, AKIRAEFINE will recognize automatically whether a QM/MM calculation is requested or whether a restart file shall be copied onto the slaves. In case an ADF restart file shall be used, the user must necessarily specify the option `nogeo` (like in the example below), since otherwise, all displaced structure calculations will be performed for the geometry given in the restart file, thus leading to nonsense results. It is not possible to use ADF restart files in combination with QM/MM calculations.

The `adf.in` input file you provide might look like this:

```
#!/bin/sh

$ADFBIN/adf -n1 << eor
Create 0  $ADFRESOURCES/TZP/0.1s
End Input
eor
mv TAPE21 t21.0

$ADFBIN/adf -n1 << eor
Create H  $ADFRESOURCES/TZP/H
End Input
eor
mv TAPE21 t21.H

$ADFBIN/adf << eor
Title  H2O

INTEGRATION 6.0  6.0

Atoms
  O      -0.004404  0.000000  -0.003115
  H       0.038319  0.000000   0.969951
  H       0.927252  0.000000  -0.287190
End

Fragments
  O t21.0
  H t21.H
```

```
End

Geometry
  GO
End

STOPAFTER GGRADS

RESTART restartfile &
  nogeo
END

symmetry nosym

XC
  LDA VWN
End

savefile TAPE21

SCF
  converge 1.0e-6 1.0e-6
End

end input
eor
```

This example can also be found in the `examples/files` directory.

5.2.2 Program selection

AKIRADEFINE first of all will ask for the user's preferences concerning the quantum chemical program package which shall be used in the single-point calculations of the distorted structures:

```
Which program would you like to use ?

tm          : use TURBOMOLE for single points
d[alton]    : use DALTON for single points
g98         : use GAUSSIAN 98 for single points
g03         : use GAUSSIAN 03 for single points
a[df]       : use ADF for single points

(<return> = default = tm)
```

The default program for the single-points is TURBOMOLE. You can select another program package by typing the abbreviation for this package. For example, g03 will select GAUSSIAN03 for the single-point-calculations.

AKIRADEFINE then checks the existence of the TURBOMOLE input files named `control` in case of TURBOMOLE or DALTON single-point calculations, `akira.com` in case of GAUSSIAN single-point calculations and `adf.in` if ADF shall be used. If the molecule under study has been optimized with GAUSSIAN or ADF and thus no TURBOMOLE `control` and `coord` files are present, AKIRADEFINE will generate a fake TURBOMOLE `control` and `coord` file from the parameters found in `akira.com` or `adf.in`, respectively.

AKIRADEFINE then reads `control` and `coord` and prints the general molecule information as well as information about the thresholds employed in the electronic structure calculations.

In the next menu, an overview of the AKIRA input parameters is given, which may be changed by the user:

Program settings menu:

Current settings:

```
cstep   = 0.01000000
numberiv = 3
scfconv = 8
maxnbm  = 21
mtype   = 1
tmpcl   = off
logcl   = off
```

Choose one of the following commands:

```
-----
cstep <real>           : set cstep [<real> in bohr]
maxnbm <int>          : choose max. no. of basis modes
numberiv <int>        : choose <int> point central differences bickley formula
scfconv <int>         : choose scfconv parameter
tmpcl                 : switch on/off removal of temporary directories
logcl                 : switch on/off removal of log directories
mtype <int>           : choose type of masses to be used
                      (1: most abundant isotopes, 2: average atomic masses,
                      3: most abundant isotopes, but deuterium mass for hydrogen)
<return>              : leave this menu
```

The parameters which can be modified in this menu are the following:

cstep: step size for the displacements from the equilibrium structure (given as the norm of the displacement vector in [bohr]; default = 0.01, larger values are recommended for low-frequency modes).

maxnbm: maximum number of basis vectors for this calculation. This number cannot be chosen larger than $3N$, the number of degrees of freedom

for this molecule (which would correspond to a full harmonic force field calculation).

numberiv: number of grid points used for the numerical differentiation (default = 3).

scfconv: SCF convergence threshold parameter (cf. TURBOMOLE manual); $\text{scfconv} \geq 8$ is strongly recommended.

tmpcl: temporary directories on slave nodes are removed after the calculation (default = off).

logcl: log files for slave node calculations are removed after the calculation (default = off).

mtype: Type of masses to be used in the calculation (1 = masses of most abundant isotopes, 2 = average atomic masses (not recommended), 3 = masses of most abundant isotopes but deuterium mass for hydrogen atoms). Note that it is not yet possible in AKIRA to specify individual atomic masses for all atoms in the molecule.

5.2.3 Convergence criteria menu

In this menu, the thresholds for the convergence checks can be set:

Convergence criteria menu:

current thresholds:

```
|max. component of residuum vector| :    0.00050000
change of max. component of r      : 0.50000000E-07
orthonormalization parameter      :                8
```

choose one of the following commands:

```
rthres <real>      : set threshold for max. component of residuum vector
rabsthres <real>  : set threshold for norm of residuum vector/(3*natoms)
dvthres <real>    : set threshold for change in wavenumber [1/cm]
ecthres <real>    : set threshold for expansion coef. of last basis vector
rchthres <real>   : set threshold for change of max. component of r
iortho <int>      : set orthogonalization parameter
d                 : default settings
&                : go back to program settings menu
<return>         : leave this menu
```

There are several different thresholds to control the convergence of the results:

rthres: maximum component of the residuum vector (default = 0.0005)

rabsthres: norm of the residuum vector divided by the number of degrees of freedom of the molecule, i.e., $3N$.

dvthres: change in wavenumber between two subsequent calculations

ecthres: expansion coefficient for last basis vector

rchthres: change in the maximum component of the residuum vector

iortho: set orthogonalization parameter

While the first four criteria are direct measures for the quality of the eigenvectors or eigenvalues of the Hessian, the last value, **rchthres** is a measure for the improvement of the eigenvector between two subsequent iterations. If this improvement is smaller than this threshold, the algorithm assumes that no further improvement for this vector is possible and stops the iterations for this vector. This can often be traced back to wrong setups in the calculation and does *not* mean that the vector is converged. If a very tight threshold for the residuum has been selected, it may, however, happen that this convergence criterion cannot be fulfilled: Due to numerical noise, it will never be possible to reduce the residuum vector to zero. Besides the convergence criteria, it is also possible to select a threshold for the orthonormality check in AKIRA, the parameter **iortho**. This parameter determines the threshold for the orthonormality of the basis vectors employed in the subspace iteration ($10^{-\text{iortho}}$). AKIRA uses a Gram–Schmidt orthogonalization to keep new basis vectors orthogonal to previous ones. For some preconditioners, the new basis vectors determined by the Davidson procedure will be almost parallel to the set of basis vectors in use. In those cases, up to **iortho** subsequent orthonormalization cycles and orthonormality checks will be performed to fulfill the orthonormality threshold. If this cannot be achieved, the program will stop with an error message since it is not possible to find a new, linearly independent basis vector.

5.2.4 Output menu

In this menu, you can modify the output of the normal mode calculated by AKIRA and switch on/off a more detailed output on the subspace iterations:

Output menu:

choose one of the following commands:

```
g98it off      : switch off g98 normal mode output
g98bs off      : switch off g98 basis vector output
tmout on       : switch on  TM output of final normal modes
xmout on       : switch on  xmol output of final normal modes
prall on       : switch on  detailed output of subspace iteration
```

```

d           : default settings
&          : go back to program settings menu
<return>   : leave this menu

```

It contains the following options:

g98it on/off switch output of normal mode approximations in G98 format after each iteration on/off. Creates files `g98.out.itn` for each iteration n .

g98bs on/off switch output of basis modes (vectors) in G98 format after each iteration on/off¹. Creates files `g98.out.bsn` for each iteration n .

tmout on/off switch output of converged normal modes in TURBOMOLE format on/off. Creates file `tm_normmodes` after convergence.

xmout on/off switch output of converged normal modes in XMOL format on/off. Creates file `xmol.XYZ.out` after convergence.

prall on/off switch detailed output of subspace iteration on/off.

The following menu will be discussed in a separate section (Sec. 5.3), since it is the most important preparation step in a mode-tracking calculation: the selection of the initial basis vectors. In this section, we proceed with a description of the following menus.

5.2.5 Hessian guess and rigid modes

After the selection of initial basis modes, which is described in Sec. 5.3, you have to decide whether the basis vectors should be kept orthogonal to translational and rotational (“rigid”) modes of the molecule:

```

Do you want to orthogonalize your guess vectors
to translational and rotational modes ?
(y/n, default = yes)

```

This is always recommended, since it reduces the computational cost and increases the accuracy of the calculation. However, it might be desired for test purposes to omit the orthogonalization on these rigid modes, or if, e.g., the full cartesian Hessian shall be calculated explicitly.

If a Hessian guess is available from the “Initial guess selection”, AKI-RADEFINE will suggest to use this guess for preconditioning purposes.

¹Wave numbers do not have a physical meaning in this case.

5.2.6 Root homing and preconditioning

After this input step follows the “Root homing and preconditioning menu” ndexroot homing menu:

```

Root homing and preconditioning menu:
-----
current settings:
root homing by overlap with selected vector of previous iteration
preconditioning scheme: backtra
  nbt          :          6

choose one of the following commands:
root <string>   : set root homing scheme; possible values:
                  lastsel = default; overlap with last eigenvector
                  trackgv = overlap with guess vector
                  minresi = choose vector with minimal residuum
                  testmin = test for minimum; optimize lowest root
prec <string>   : set preconditioner; possible values:
                  backtra = backtransformation of Davidson Matrix
                  unitmat = Davidson with unit matrix
                  lanczos = Lanczos type diagonalization
nbt <int>       : min. no. of vectors for backtransformation
genstart on     : only one iteration for generation of start vectors
d               : default settings
&              : go back to convergence criteria menu
<return>       : leave this menu

```

For root homing, the following choices can be made:

lastsel: default; the program selects as a new vector to be optimized that eigenvector approximation which has the largest overlap with the last vector selected.

trackgv: the vector with the largest overlap with the initial guess vector will be chosen for further optimization.

minresi: the vector with the lowest residuum will be further optimized, irrespectively of its nature, i.e., the type of vibration involved.

testmin: the vector with the lowest eigenvalue will be selected in order to test for a minimum or saddle point.

Note that for each basis vector given in the first iteration, one vector for further optimization will be selected.

For preconditioning, the following options are available:

backtra: backtransformation of the Davidson Matrix according to Eq. (4.11). Default if no approximate Hessian is available.

unitmat: a unit matrix preconditioner will be applied.

lanczos: test option: Lanczos algorithm. Only one vector should be selected in this case, since the block-Lanczos algorithm, which allows to optimize several vectors simultaneously, is not yet tested.

If an approximate Hessian matrix is available, three additional options will appear on the screen:

jacobid: use Jacobi–Davidson diagonalization scheme. Default if approximate Hessian is available.

aphsinv: Davidson algorithm in which the approximate Hessian will be explicitly inverted to get the preconditioner. Yields similar performance as Jacobi–Davidson.

aphsdia: test option: original Davidson algorithm in which the inverse diagonal elements are used as a preconditioner. Cannot be recommended for calculations, since this assumes that the matrix to be diagonalized is diagonally dominant, which is in general not the case for Hessian matrices.

Note that the first three options can be selected irrespectively of the availability of an approximate Hessian, while the latter three require such a guess (and will thus not appear on the screen if no approximate Hessian is present). The **jacobid** and **aphsinv** options usually lead to the best convergence, if appropriate guess Hessians are available. The **backtra** option sometimes shows bad performance during the first iterations, even worse than the most simple unit matrix preconditioner. Therefore, it is possible to use a combined approach in which the first n iterations will employ a unit matrix preconditioner, while in the following iterations the backtransformation will be used. The number n can be set with the keyword **nbt**.

Note that experience with these options is still somewhat limited. It appears, however, that no big differences occur between the **unitmat** and **backtra** preconditioning schemes.

One last option, **genstart on/off**, can be selected to do exactly one iteration. After this iteration, you can use one of the vectors created in this cycle as a better approximation for the desired normal mode.

5.2.7 Additional keywords

Some keywords can only be introduced manually into the file **akira_control**. At the moment, the following options are available:

sleeptime <time-in-seconds>: sets the time in seconds to wait for the master process between two subsequent checks of the slave processes.

Default: 5. Smaller values decrease the overhead, in particular in fast calculations, larger values can be helpful to decrease the network traffic when running the master job via a network file system, in particular for large calculations with long single-point calculations.

5.2.8 Data collector options

The next and last step of AKIRADEFINE creates the files necessary for the parallel execution of the data collection:

```
=====
Input completed, calling prepdc ...
=====

ELIGIBLE file to be used for choose_nodes call:
$CIPROC/ELIGIBLE

Specify name of alternative ELIGIBLE file or press <return>
```

The file DCINPUT will be created by calling the script `mkrdcinput`. The DCINPUT file can be further modified manually afterwards, see Section 3.1.2 for details. A directory for logfiles, named `log`, will be created. The logfiles contain valuable information on what is going on on the slave nodes (see also section A in the appendix).

5.3 AKIRADEFINE: Setting up the initial guess

While AKIRA itself can be used mainly as a black-box program, the setup of the initial guess must be done by the user, and it requires some idea of the specific problem to be analyzed. If there is no such specific problem, it is probably not useful to run a mode-tracking calculation, since either all normal modes are wanted to get an overview, or no modes at all are of interest.

If, however, the problem is identified, AKIRADEFINE offers a lot of help to construct an appropriate guess for the modes to be studied in the initial basis vector menu:

```
Selection of initial basis vectors
-----
No. of basis modes selected:  0
choose one of the following commands:

-----
pint menu  :  show internal coordinate options
-----
psym menu  :  show symmetry coordinate options
```

```

-----
pcart menu : show cartesian coordinate options
-----
pfile menu : show basis-vector-from-file options
-----
pmpop menu : show basis-vectors-from-MOPAC options
-----
trarot      : create translational and rotational modes
submenu     : enter subsystem menu
<return>    : quit this menu

```

By typing `xxx menu`, you can display the menu `xxx`, and by typing `xxx hide`, you can hide it again. In the following, the individual menus are explained in detail.

5.3.1 Internal coordinates

In many cases, the user might be interested in a local vibration of the molecule, like a stretching mode of a particular bond between two atoms, or a valence angle bending. In the internal coordinate submenu which can be displayed by typing `pint menu`, AKIRADEFINE offers the following possibilities to select such modes:

```

str i j select a stretching mode between two atoms

bd i j k select a bending mode between three atoms

tor i j k l select a torsional mode between four atoms

oop i j k l select an out-of-plane mode, characterized by four atoms

br select a totally symmetric breathing mode of the molecule

brm select a mass-weighted totally symmetric breathing mode of the molecule

```

5.3.2 Symmetry coordinates

In some cases, also symmetry coordinates may be a good starting point for the subspace iteration: Indeed, they sometimes completely determine the vibrational modes.

```

sym          : generate symmetry coordinates as basis vectors
spesym       : generate normal modes for special symmetries
psym hide    : hide symmetry coordinate options

```

Some special cases are included with the `spesym` option, while general symmetry coordinates for practically every point-group symmetry can be constructed with the `sym` option. Note that in AKIRADEFINE does not check the symmetry, which the user has to specify manually. However, giving a wrong input for the symmetry of your molecule will only result in an empty set of symmetry coordinates.

All symmetry coordinates for the chosen irrep of a particular symmetry will be put in a buffer, and may be selected from there as basis vectors in the mode-tracking calculation (see section 5.3.6).

5.3.3 Cartesian coordinates

A very simple way to construct a guess for a normal coordinate is simply to select a Cartesian nuclear coordinate. This might be useful if you want to investigate *all* vibrations of a molecule in which a particular atom or a set of atoms is involved. Since this is usually not very specific, especially when a large number of cartesian coordinates is selected, we recommend to use the `genstart` option (Sect. 5.2.6) or a similar procedure: With that option, the calculation stops automatically after the first iteration (while it has to be stopped manually after several iterations). After *every* iteration, an output of the approximate normal modes is performed². If you specified several cartesian basis vectors as initial guesses, then the first iteration will produce linear combinations of these basis vectors, which are already to a certain extent adapted to the specific molecule. These might be much better approximations for the normal modes you are looking for than the cartesian basis vectors themselves, and the interesting linear combinations can be selected as basis vectors in a second mode-tracking calculation.

```
at <i>      : select all cartesian basis vectors for atom i
atx <i>    : select cartesian x-basis vector for atom i
aty <i>    : select cartesian y-basis vector for atom i
atz <i>    : select cartesian z-basis vector for atom i
aatz      : select all cartesian z-basis vectors
cfull     : select all cartesian basis vectors
            (full frequency analysis)
pcart hide : hide cartesian coordinate options
```

The command `cfull` can be used to specify a full frequency analysis. In that case, all $3N$ cartesian normal modes will be taken as basis vectors. The rigid motions will be removed by orthogonalization, so that only $3N - 6$ (for non-linear molecules) basis vectors are employed.

5.3.4 Normal coordinates from file

As mentioned earlier in this manual, results from cheap low-level calculations, like force-field, semiempiric, or small-basis set calculations might

²If not explicitly suppressed by the user.

already be available for your molecule. For such cases, it is possible to simply read the output files of some molecular mechanics or quantum chemistry programs as an input for AKIRA.

```
re <i>      : read <i> basis vectors from file
resub <i>   : read <i> basis vectors for subsystem from file
pfile hide : hide basis-vector-from-file options
```

`re` can be used to read normal modes from a file which has to be specified later. It is also possible to use the `resub` command and read normal modes for a subsystem. In that case, it is necessary to specify the number n of atoms in this subsystem:

```
Enter No. of atoms in subsystem:
(the first <No.> atoms will be taken !!!)
```

It is always assumed that the first n atoms belong to the chosen subsystem. General subsystems can be handled by the subsystem menu submenu, Sect. 5.3.5. Next, AKIRADEFINE will ask the user for the name of the file which contains the normal modes to be read in as initial basis vectors,

```
Enter filename:
```

Then, the format of these basis vectors can be selected:

```
Select format of basis vectors
-----

snf      : SNF normal modes (massweighted MOPAC)
tmn      : TURBOMOLE normal modes
tmb      : Basis vectors in TURBOMOLE format
tmi      : Internal coordinates in TURBOMOLE format
g98      : Gaussian98 normal modes
mop      : MOPAC normal modes (not massweighted)
pcm      : PCM normal modes (massweighted)
adf      : ADF normal modes (not massweighted)
free     : free format modes (not massweighted)
<return> : return to main menu
```

AKIRADEFINE contains routines to read the following input formats:

- SNF - normal modes (massweighted MOPAC style)
- TURBOMOLE - normal modes
- GAUSSIAN98 normal modes (Gaussian03 is also supported)

- MOPAC - normal modes
- PCMODEL - normal modes
- ADF - normal modes
- free format normal modes (one number per line, modes separated by wavenumbers)

The modes read from file are stored in a buffer, from which they can be selected as basis vectors for the AKIRA calculation (see section 5.3.6).

5.3.5 Subsystem menu

One of the possible applications of mode-tracking is to study vibrations of a large molecule which are local in that sense that they are more or less restricted to a certain subunit of the system. There may still be couplings with motions of the rest of the molecule, but as a first approximation, only the atoms of the subunit may be involved. In those cases, it might actually be possible to find a small model for the full (super-)molecule, which only consists of the subunit (plus a model for the rest of the molecule), and for which a full frequency analysis is possible. Then these subsystem normal modes can be used as a guess for the full molecule. Indeed, it might be possible to identify several subunits. The subsystem menu allows to define a number of subunits, and to read normal coordinates and wavenumbers for them, which then are used as basis vectors for the full molecule.

Additionally, it is possible to combine the Hessians for these subsystems and copy them to the corresponding blocks of the full Hessian, so that a guess for the full molecule is obtained, which can be used for preconditioning purposes. By typing `submenu` in the main basis vector menu, you get to the subsystem menu:

```

=====
No. of subsystems specified so far:  0
=====

Select one of the following commands:

ns <i> <j>      : create new subsystem, ranging from
                atom <i> to <j>
rnm <i>        : read normal modes for subsystem <i>
chs <i>        : create Hessian for subsystem <i>
trarot <i>     : create trarot-modes for subsystem <i>
n2fs <i>       : copy modes for subsystem <i> to full system
showpr        : show print options
c2fh <i>       : copy Hessian for subsystem <i> to full Hessian
ints <i j r>   : set interaction strength between blocks <i> and <j> to <r>
inta <r>       : set interaction strength for all blocks to <r>

```

```

whs      : write (appr.) hessian to akira_control
          for preconditioning purposes
cnm      : create normal modes by diagonalization
          of current full system (appr.) Hessian
cl       : clear all subsystem entries
*,q     : return to basis vector menu

```

In that menu, the following commands are possible:

```

ns <i j> define a new subsystem, ranging from atom i to atom j.

rnm <i> read normal modes from file for subsystem no. i. For possible
input formats, see Sect. 5.3.4.

chs <i> create Hessian for subsystem no. i from normal modes and wave-
numbers. Only possible if normal modes and wavenumbers have al-
ready been read.

trarot <i> create translational and rotational modes for subsystem no. i.
This option is useful when it can be assumed that modes of a super-
molecule can be described by out-of-phase translations or rotations
of different molecules within the supermolecule. These modes can
be constructed automatically, no preceding subsystem calculation is
necessary.

n2fs <i> copy normal modes for subsystem i to set of normal modes for
full system.

showpr show print options (for normal modes, wavenumbers, and Hessians)

c2fh <i> copy Hessian for subsystem i to full Hessian. Only possible if
subsystem Hessian has been constructed before.

ints <i j r> artificially set the interaction strength, i.e., the off-diagonal
elements between blocks for subsystems i and j to the constant value
r. These blocks are otherwise always zero in the model Hessian for the
full molecule, so that no coupling exists between different subunits.

inta <r> artificially set the interaction strength for all off-diagonal blocks
(blocks between subsystems) to the constant value r. Like ints, this
command allows to empirically correct the model Hessian.

whs write the model Hessian for the full molecule to file akira_control in
order to be able to use it for preconditioning purposes.

cnm create model normal coordinates by diagonalization of the model Hes-
sian for the full system.

```

`c1` clear all subsystem entries

To keep an better overview over the subsystems, a table is shown with the relevant atom numbers, the number of modes read, as well as information about whether the normal modes have been copied to the full system modes, whether the Hessian is created, and whether the Hessian is copied to the full system. After leaving this menu, the modes which are either copied from a subsystem to the full system, or which are created by diagonalization of the model system hessian, will be stored in a buffer. From there, they may be selected as basis vectors.

5.3.6 Other options

Further options which are available in the basis vector menu, are the following:

`trarot` create translational and rotational modes. For test purposes, they might be included in the mode-tracking calculation.

`cbv` `<i>` copy vector `i` from buffer to set of selected basis vectors. Only available if there are modes in buffer.

`acbv` copy all vectors in buffer to set of selected basis vectors. Only available if there are modes in buffer.

`ort` orthogonalize basis vectors to translational and rotational modes. Only available if basis vectors have been selected. Orthogonalization is carried out automatically afterwards anyway, unless suppressed by the user.

The following options are available for the set of selected basis vectors (without preceding “`b`”) or for the set of buffer vectors (with preceding “`b`”):

`[b]disp` display modes using MOLDEN [14]

`[b]c` clear set of modes

`[b]p` print set of modes

`[b]pm` print set of mass-weighted modes

`[b]g98` write G98 fake output of modes to file `g98.out.it0` or `g98.out.buf`.
Is done automatically when using command `[b]disp`.

`[b]xmol` write XMOL fake output of modes to file `xmol.XYZ.it0` or `xmol.XYZ.buf`.

5.3.7 Restricting the Subspace

Under certain circumstances it may be helpful to restrict the search space for the subspace iteration. Typical examples could be the restriction to basis vectors within a certain irreducible representation or the exclusion of exact normal modes found in a previous run. One restriction that is by default always employed is that the basis vectors are kept orthogonal to translational and rotational motions.

Other restrictions can be applied by specifying a set of orthogonal vectors in the guess vector menu. Any vector that is available in the buffer (see above) can be copied into the set of orthogonal vectors with the commands:

`cov <i>` copy vector *i* from buffer to set of orthogonal vectors. Only available if there are modes in buffer.

`acov` copy all vectors in buffer to set of orthogonal vectors. Only available if there are modes in buffer.

If some vectors have been chosen for the orthogonal complement, their number will be displayed in the menu. Since it is sometimes easier to specify the space in which the possible solutions should lie (instead of its orthogonal complement), there is an additional menu appearing after quitting the basis vector selection that allows the user to either keep the solution vectors within the set of vectors specified in the “orthogonal subspace” selection, or to keep them orthogonal to that set,

```
You have selected an additional set of vectors
to restrict the search space for the solution
vectors. Do you want to ...
sub - keep the solution vectors within this set
      (default)
ort - keep the solution orthogonal to this set
q   - quit without using this set of vectors
```

If `sub` is chosen, AKIRADefine constructs the complementary space to the one selected before and keeps the solution vectors orthogonal to that set. Note that the orthogonal subset vectors are also orthogonalized to translational and rotational modes, as all basis vectors are per default kept orthogonal to these rigid modes. The orthogonal modes are written to the file `akira_control`.

5.4 Subspace iteration with AKIRA

The general procedure for a subspace iteration is outlined in the flow-chart in Fig. 5.1. To request a subspace iteration with AKIRA simply call

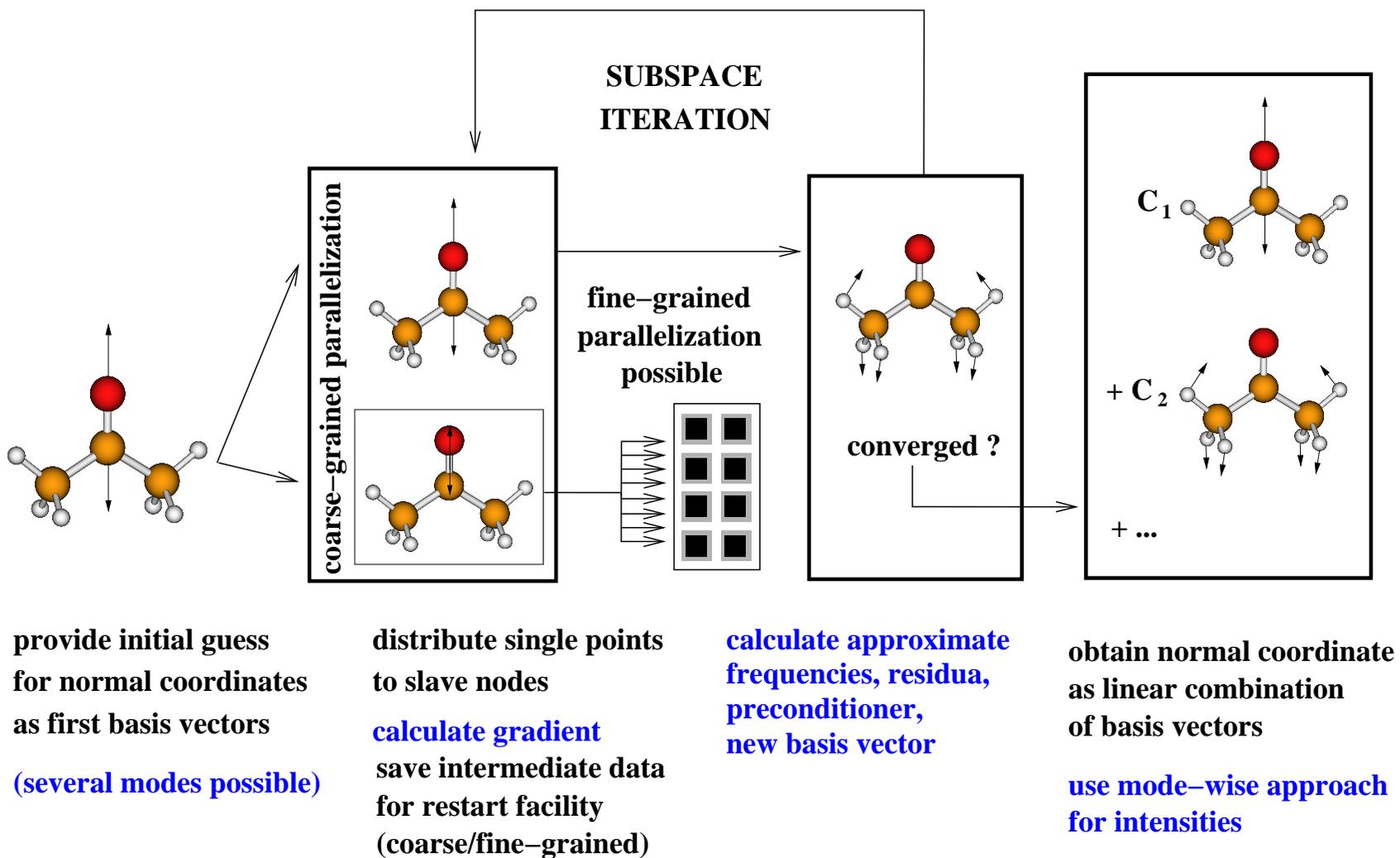


Figure 5.1: Outline of the parallelized mode-tracking algorithm as implemented in AKIRA (figure taken from Ref. [22]).

akira

without any further program options. These must be specified in the AKIRA control file `akira_control` and can be selected interactively in AKIRADE-FINE.

Akira writes an output file named `akira.out`, which contains intermediate data for every iteration. In particular, the following quantities are reported:

- vector σ^i , Eq. (4.3)
- approximate Hessian matrix for subspace (Davidson matrix), $\tilde{\mathbf{H}}^{(m),i}$, Eq. (4.5)
- approximate eigenvalues and eigenvectors of the Davidson matrix, ρ^i and \mathbf{u}^i , Eq. (4.6)
- approximate wavenumbers calculated from the eigenvalues
- approximate normal modes, \mathbf{v}_s^i , Eq. (4.8)
- information about the root-homing procedure; three quantities are calculated in order to select the eigenvector to be optimized from the set of the \mathbf{v}_s^i :
 - the unsigned difference between the eigenvalues of current iteration and the eigenvalue(s) of the eigenvector(s) selected in the last iteration
 - the squared norm of the difference vector between the eigenvectors in the current iteration and the eigenvector(s) selected in the previous iteration.
 - the unsigned overlap of the current eigenvectors and the eigenvector(s) selected in the previous iteration.

Note that the root-homing is necessary for all eigenvectors which shall be optimized simultaneously. If the three criteria lead to different eigenvectors, the overlap criterion will be used to determine the new eigenvector(s) to be optimized.

- residuum vectors \mathbf{r}_s^i for all approximate eigenvectors \mathbf{v}_s^i , Eq. (4.7)
- convergence control parameters:
 - norm of residuum vector(s) for selected eigenvector(s)
 - change of residuum vector norm
 - maximum component of the residuum vector

- contribution of the last basis vector to the new approximate normal mode vector
- new Cartesian basis vector(s) \mathbf{b}^{i+1} , Eq. (4.9)
- intermediate results: wavenumbers, $|r|$, $|r_{\max}|$ and convergence status for *all* approximate normal modes in the current iteration

Furthermore, the file `wavenumbersis` is updated after each iteration, which contains the wavenumber of residuum vector for each approximate normal mode.

5.5 Restart facilities

The single-point data obtained for structures perturbed along the basis vectors are stored in the file `restart.akira`. Besides the gradients for all displaced structures, which enter Eq. (4.3), the energies, the equilibrium structure and all perturbed structures are written to this file. The general structure of the restart file is essentially the same as for the program package SNF (see introduction), with the difference that gradients and energies are collected in groups for every basis vector, not in groups for Cartesian displacements of a particular atom (as in SNF) or in groups for the normal modes of the molecule. In earlier AKIRA versions, the number of basis-vector entries in the restart file was equal to the number of degrees of freedom of the molecule for convenience. In version 3.1.2, this was changed into the number of basis modes present up to the current iteration. This was necessary for the QM/MM interface of AKIRA, which allows to treat systems with a huge number of degrees of freedom, in order to keep the size of the restart file small.

The stepflag entries in the restart file determine which calculations have to be done in a particular iteration step. Further restart information is written in the program control file `akira_control` in the following form:

```
$nbnm    3
          1
          1
          1
```

This means that the current iteration is the third, and the three “1” entries specify the number of basis vectors added in each iteration, i.e., there are now three basis vectors, one of which was generated per iteration step. This information allows — in combination with the stepflags — to restart the calculation at every point of the subspace iteration, irrespectively of the current process (single point calculations for perturbed structures or

solution of the Davidson matrix eigenvalue problem/generation of new basis vectors).

It is sometimes necessary or convenient to restart the calculation at an earlier iteration. This is possible by introducing the additional keyword

`$restartit n`

in the file `akira_control`, where `n` is the iteration number after which the calculation should be restarted, i.e., restart information will only be used up to iteration `n`. **CAUTION: This will remove all previous raw data from iteration `n+1` on your restart file !!!**

Summary AKIRADEFINE:

- ⇒ If you have performed a geometry optimization with TURBOMOLE, just type `akiradefine` in your working directory.
- ⇒ If you have used GAUSSIAN for the optimization, create a `akira.com` file (see Sect. 5.2.1) and then type `akiradefine` in your working directory.
- ⇒ For ADF users: assure that you have a runscript named `adf.in` in your working directory which contains all settings used in the preceding geometry optimization.
- ⇒ Select the options in the menus (see Sect. 5.2 for details).
- ⇒ Note possible warnings in the output of AKIRADEFINE.
- ⇒ For TURBOMOLE users: If the symmetry mentioned in the original `control` file is higher than C_1 , create a C_1 MO file by using TURBOMOLE'S `DEFINE`.
- Mandatory files for AKIRADEFINE:
`control`, `coord` (TURBOMOLE, DALTON) or `akira.com` (GAUSSIAN) or `adf.in` (ADF).
- Mandatory files and scripts for automatical generation of AKIRA input files by AKIRADEFINE:
`$CIPROC/ELIGIBLE`, `choose_nodes`, `mkrdcinput` or `mkparainput`
- Files created by AKIRADEFINE:
`restart`, `akira_control`, `control.bak`, `DCINPUT`, `USE_NODES`, `MACHINES`, `mpi.sub`

Summary AKIRA:

- ⇒ Create the files `USE_NODES` using the script `choose_nodes` (this is also done automatically by `AKIRADefine`).
- ⇒ If necessary, stop all old PVM demons and remove PVM log-files. This can be done by the `clear_pvm` script.
- ⇒ Start the AKIRA calculation by typing `akira` in your working directory, or, in order to prevent the AKIRA from crashing when you log out, to send the command to the background and to pipe the screen output to a file:
`nohup akira > filename &`
- ⇒ You may check the progress of the calculation via the status file `TMPdcstat`, which can be constantly viewed by the command `dcmore`.
- Mandatory files for AKIRA:
`control`, `coord`, `restart`, `DCINPUT`, `akira_control`;
`TURBOMOLE`, `DALTON`: `basis`[and `auxbasis`], `mos` [or `alpha`, `beta`];
`GAUSSIAN`: `akira.com`
`ADF`: `adf.in`
PVM version: `$CIPROC/ELIGIBLE`, `USE_NODES`
MPI versions: `MACHINES` (for `mpirun`)
- Mandatory scripts for AKIRA:
`choose_nodes`, `ruptime`
- Files created by AKIRA in the working directory:
`TMPdcstat`, `fort.41`, `g98.out.itn`, `g98.out.bsn`
- Files created by AKIRA in the log directory:
`PREFIXdclog.XXXXXXX`
- Files created by PVM during the parallelized calculation:
`pvml.<uid>`, `pvmd.<uid>`
- ⇒ Note possible warnings in the output of AKIRA.

5.6 Double-parallel runs

The maximum number of processors in the parallel machine is restricted by the number of tracked vibrations (2 per vibration, if a 3-point central-difference formula is used for numerical differentiation). Thus, it may be desirable to run also the single-point calculations in parallel to exploit the full capacity of a computer cluster. At present, this has only been tested for the AKIRA—ADF interface on a PC cluster built from machines with 2 dual-core AMD Opteron processors (\Rightarrow 4 processors) each, using the MPI version of AKIRA. The ADF calculations were parallized employing PVM, so any interference of the two parallel processes was avoided. To control the parallel execution of the ADF jobs, the following lines were inserted at the very beginning of the ADF input file `adf.in`:

```
/usr/bin/pvm << eor
quit
eor

echo 'uname -n' > adfhosts
echo 'uname -n' 4 > nodeinfo

export SCMSPAWNSCRIPT=$ADFBIN/adfs
export NSCM=4
```

Furthermore, it should be checked that the main ADF run is not started as `$ADFBIN/adf -n1 << eor`, but as `$ADFBIN/adf << eor` by `adf.in`.

A. First aid

Since parallel calculations on a computer cluster can be sensitive to disturbances on the nodes (for example, a slave node might crash during the calculation, or the installation of the quantum chemical program package you chose for the distorted-structure calculations might be inappropriate or deficient on one some nodes), it is possible that the AKIRA mode-tracking calculation will crash. Due to the restart facilities provided by AKIRA, this will in general not pose a serious problem. The following hints are given assuming that the PVM version of AKIRA is used: After executing the `clear_pvm` script, you can restart the calculation by simply calling `choose_nodes` and then `akira` in your working directory. All information which has been collected from the slaves before the crash is stored in the `restart.akira` file and will be used in the restarted calculation.

In order to know why the calculation crashed and to prevent it next time, there are some files and directories which will help you.

- AKIRA distributes perturbed molecular structure single point calculations to the processors chosen in the file `USE_NODES`. If you think that there is one special slave node in your cluster which is causing the calculation to crash, you can take this node out of your the `ELIGIBLE` file. No single-point calculations will be performed on this node anymore. Of course, the calculation will also crash if there are no free nodes available und thus no entries in the `USE_NODES` file.
- Temporary directories are created in the `TMPDIR/TMPxxxxx` directories (`TMPDIR` is specified in the `DCINPUT` file) on the slaves, which contain the input and output files of the distorted-structure single-point calculations. They will be removed by executing the `clear_pvm` script.
- A logfile is generated for every slave process in the `LOGDIR` directory. These logfiles can be used to get information about errors which occur during the run of AKIRA in the master process and the slave processes.
- The file `fort.41`, which is created in the working directory (`MYDIR`), can also be useful to get information about errors in the master process and the slave processes.

- To check for errors in the PVM machine, the PVM files `pvm1.<uid>` and `pvm2.<uid>` can be used.

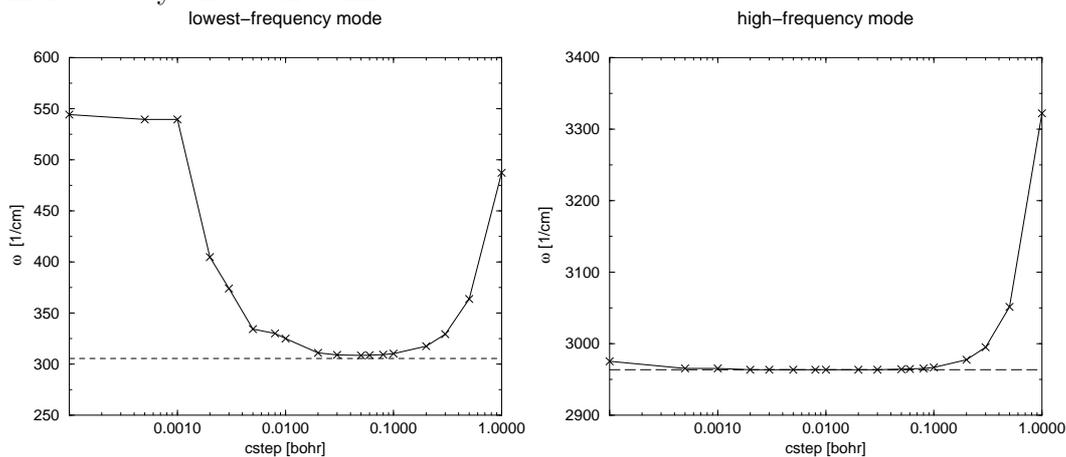
Some error messages you might get, and possible remedies:

- Prints “Killed” when trying to start `AKIRADefine` or `AKIRA` ⇒ check `mxwrk` in `defs.h` — this variable specifies the total memory taken and might be too large for your machine. Lower the value of `mxwrk` and recompile (`make clean; make`)

B. Parameter analysis: step size for numerical differentiation

To determine an optimum step size for the numerical differentiation, we carried out displacements of the atoms in the ethane molecule along the lowest-frequency normal mode, and along one high-frequency mode, and calculated the force constants as numerical derivatives of the analytical gradients (BP86/TZVP). The resulting frequencies as a function of the step size are displayed in Fig. B.1. Note that these issues have been discussed in detail in Ref. [2].

Figure B.1: Vibrational wavenumbers in cm^{-1} as a function of the step size for the numerical differentiation for the lowest-frequency (left) and a high-frequency mode (right) of ethane. The analytical wavenumbers are indicated by the dashed lines.



From this figure it can be seen that it might be advantageous — though the general recommendation is 0.01 — to increase the step size `cstep` to values of up to 0.1 for *low-frequency* modes.

Bibliography

- [1] M. Reiher, J. Neugebauer. A mode-selective quantum chemical method for tracking molecular vibrations applied to functionalized carbon nanotubes. *J. Chem. Phys.*, **118** (2003) 1634–1641.
- [2] Markus Reiher, Johannes Neugebauer. Convergence Characteristics and Efficiency of Mode-Tracking Calculations on Pre-Selected Molecular Vibrations. *Phys. Chem. Chem. Phys.*, **6** (2004) 4621–4629.
- [3] J. Neugebauer, M. Reiher, C. Kind, B. A. Hess. Quantum Chemical Calculation of Vibrational Spectra of Large Molecules — Raman and IR Spectra for Buckminsterfullerene. *J. Comput. Chem.*, **23** (2002) 895–910.
- [4] Ernest R. Davidson. The Iterative Calculation of a Few of the Lowest Eigenvalues and Corresponding Eigenvectors of Large Real-Symmetric Matrices. *J. Comp. Phys.*, **17** (1975) 87–94.
- [5] H. A. Van der Vorst G. L. G. Sleijpen. A Jacobi–Davidson Iteration Method for Linear Eigenvalue Problems. *SIAM Review*, **42** (2000) 267–293.
- [6] C. Lanczos. *J. Res. Nat. Bur. Stand.*, **45** (1950) 255.
- [7] Bernd A. Hess. Memmgr (Version 1.19), 1999.
- [8] Bernd A. Hess. Delrem (Revision 1.66), 2001.
- [9] G. te Velde, F. M. Bickelhaupt, E. J. Baerends, C. Fonseca Guerra, S. J. A. van Gisbergen, J. G. Snijders, T. Ziegler. *J. Comp. Chem.*, **22**(9) (2001) 931–967.
- [10] T. Helgaker, H. J. Aa. Jensen, P. Jørgensen, J. Olsen, K. Ruud, H. Ågren, A. A. Auer, K. L. Bak, V. Bakken, O. Christiansen, S. Coriani, P. Dahle, E. K. Dalskov, T. Enevoldsen, B. Fernandez, C. Hättig, K. Hald, A. Halkier, H. Heiberg, H. Hettema, D. Jonsson, S. Kirpekar, R. Kobayashi, H. Koch, K. V. Mikkelsen, P. Norman, M. J. Packer, T. B. Pedersen, T. A. Ruden, A. Sanchez, T. Saue, S. P. A. Sauer, B. Schimmelpfennig, K. O. Sylvester-Hvid, P. R. Taylor, O. Vahtras. DALTON, a molecular electronic structure program, Release 1.2, 2001.

- [11] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, J. A. Montgomery, Jr., T. Vreven, K. N. Kudin, J. C. Burant, J. M. Millam, S. S. Iyengar, J. Tomasi, V. Barone, B. Mennucci, M. Cossi, G. Scalmani, N. Rega, G. A. Petersson, H. Nakatsuji, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, M. Klene, X. Li, J. E. Knox, H. P. Hratchian, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, P. Y. Ayala, K. Morokuma, G. A. Voth, P. Salvador, J. J. Dannenberg, V. G. Zakrzewski, S. Dapprich, A. D. Daniels, M. C. Strain, O. Farkas, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. V. Ortiz, Q. Cui, A. G. Baboul, S. Clifford, J. Cioslowski, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. L. Martin, D. J. Fox, T. Keith, M. A. Al-Laham, C. Y. Peng, A. Nanayakkara, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, C. Gonzalez, J. A. Pople. Gaussian 03, Revision C.02. Gaussian, Inc., Wallingford, CT, 2004.
- [12] Reinhart Ahlrichs, Michael Bär, Marco Häser, Hans Horn, Christoph Kölmel. Electronic structure calculations on workstation computers: The program system TURBOMOLE. *Chem. Phys. Lett.*, **162**(3) (1989) 165–169.
- [13] J. J. P. Stewart, I. Rossi, W.-P. Hu, G. C. Lynch, Y.-P. Liu, Y.-Y. Chuang, J. Li, C. J. Cramer, P. L. Fast, D. G. Truhlar. Mopac 5.09mn. University of Minnesota, Minneapolis, 1999.
- [14] Gijs Schaftenaar. Molden3.7 — a pre- and post processing program of molecular and electronic structure. CMBI, the Netherlands, 2001.
- [15] E. Bright Wilson, J. C. Decius, Paul C. Cross. *Molecular Vibrations*. McGraw-Hill, New York, 1955.
- [16] Ira N. Levine. *Molecular Spectroscopy*. Wiley, New York, 1975.
- [17] W. G. Bickley. Formulae for numerical differentiation. *Math. Gaz.*, **25** (1941) 19–27.
- [18] B. Liu. *Numerical Algorithms in Chemistry: Algebraic Methods* (edited by C. Moler and I. Shavitt). Lawrence Berkeley Laboratory LBL-8158. Livermore, CA, 1978.
- [19] Christopher W. Murray, Stephen C. Racine, Ernest R. Davidson. Improved Algorithms for the Lowest Few Eigenvalues and Associated Eigenvectors of Large Matrices. *J. Comp. Phys.*, **103** (1992) 382–389.

- [20] W. Butscher, W. J. E. Kammer. *J. Comput. Phys.*, **20** (1976) 13.
- [21] M. Reiher, J. Neugebauer, B. A. Hess. Quantum chemical calculation of Raman intensities for large molecules: The photoisomerization of $[\{\text{Fe}'\text{S}_4'(\text{PR}_3)\}_2(\text{N}_2\text{H}_2)]$ ($\text{S}_4'^{2-} = 1,2\text{-bis}(2\text{-mercaptophenylthio)ethane}(2-)$). *Z. Physik. Chem.*, **217** (2003) 91–103.
- [22] Johannes Neugebauer, Markus Reiher. Vibrational Center–Ligand Couplings in Transition Metal Complexes. *J. Comput. Chem.*, **25** (2004) 587–597.

Index

- DCINPUT, 14
- ELIGIBLE, 16
- MACHINES, 16
- TMPdcstat, 18
- USE_NODES, 16
- \$nnbm, 53
- adf.in, 37
- akira.com, 33
- akira.out, 52
- akira_control, 33
- akiradefine, 13
- akira, 13
- choose_nodes, 16
- clear_pvm, 13
- control, 37
- dcmore, 13, 19
- fort.41, 59
- mkparainput, 13
- mkrdcinput, 13
- mpi.sub, 17
- pvm1.<uid>, pvmd.<uid>, 60
- restart_akira, 53
- wavenumbers, 53
- ADF, 37
- AKIRADEFINE, 33
- AKIRA input parameters, 37
- DALTON, 32, 33
- GAUSSIAN, 32, 33
- MPI version, 16
- TURBOMOLE, 32, 33

- accuracy of preceding geometry optimization, 31

- backtransformation, 42

- cartesian coordinates menu, 45
- collective displacement, 21

- convergence criteria (Davidson algorithm), 23, 38
- crash of calculation, 59

- Davidson algorithm, 21
- Davidson matrix, 22
- Davidson–Liu or block-Davidson method, 28
- Displacements in units of length, 28
- display guess modes, 49

- exactly one iteration, 42
- excited states, 31

- full frequency analysis, 45

- Gram–Schmidt orthogonalization, 39

- Hessian guess, 23, 40

- initial guess, setup, 43
- internal coordinates menu, 44

- Jacobi–Davidson algorithm, 25

- Lanczos algorithm, 27
- logfiles from slave nodes, 59

- Mode-Tracking, 5, 7
- molecular symmetry, 32
- Monitoring AKIRA calculations, 18

- normal coordinates, read from file, 45
- normal modes, read in for subsystem, 46, 47
- numerical differentiation, 22, 29

- orthonormality check, 39
- output: normal modes, 39
- output: subspace iterations, 39

- parameter analysis, 61
- preconditioning, 23
- preconditioning menu, 41
- program selection, 33
- program selection menu, 36

- residuum vector, 22
- restart facilities, 53, 59
- root homing, 28

- semi-numerical differentiation, 32
- step size, 29, 61
- stepflags, 53
- subspace iteration, 50
- subspace iteration techniques, 21
- subsystem menu, 47
- symmetric vibrations, 32
- symmetry coordinates menu, 44

- translational and rotational modes,
inclusion of, 49
- translational and rotational modes,
orthogonalization to, 40, 49