



AMS Manual

Amsterdam Modeling Suite 2021.1

www.scm.com

May 21, 2021

CONTENTS

1	General	1
1.1	Overview	1
1.2	What's new in the AMS driver?	1
1.2.1	New in AMS2021.1	1
1.2.2	New in AMS2020.1	1
1.2.3	New in AMS2019.3	2
1.2.4	New in AMS2019.1	3
1.3	Motivation and progress	3
1.4	Input, execution and output	4
2	Input, execution and output	7
2.1	Input	7
2.1.1	Example	7
2.1.2	Tasks	8
2.1.3	Properties	8
2.1.4	General remarks on input structure and parsing	9
2.1.5	Keys	9
2.1.6	Blocks	11
2.1.7	Including an external file	12
2.1.8	Units	12
2.2	Execution	13
2.2.1	Shell script	13
2.2.2	Interactive input file	13
2.2.3	Running AMS on compute clusters	14
2.3	Output	16
2.3.1	Results directory	16
2.3.2	Logfile ams.log	16
2.3.3	Binary output files	16
2.3.4	Standard output	17
2.3.5	Optimized geometry in xyz format	17
2.3.6	AMS environment variables	18
2.4	Driver level parallelism	18
2.5	Python interface	22
2.6	Pipe interface	22
2.6.1	AMSPipe protocol specification	22
	Low-level message encoding	23
	Return messages and error handling	23
	Methods	24
2.6.2	AMS as a pipe master	26
2.6.3	AMS as a pipe worker	27

3	Geometry, System definition	29
3.1	Geometry, Lattice	29
3.1.1	Modifying the geometry	31
3.2	Symmetry	33
3.3	Regions	34
3.4	Charge, atomic masses, input bond orders	35
3.5	Homogeneous electric field and multipole charges	35
3.6	Load a System from file	37
3.7	Atom attributes	38
3.8	Force field related extensions	38
3.8.1	Load charges for a forcefield into regions	38
3.8.2	Load forcefield atom types	39
4	Structure and Reactivity, Molecular Dynamics	41
4.1	Single point calculations	41
4.2	Bond energy calculations	41
4.2.1	Ground state energy	41
4.2.2	Formation energy	42
4.2.3	Atomization energies	42
4.2.4	Chemisorption energies	42
4.2.5	Atomic corrections	42
4.2.6	Open shell systems	43
4.2.7	Impurities	43
4.3	Geometry optimization	43
4.3.1	Constrained optimization	46
	Restraints	48
4.3.2	Optimization under pressure / external stress	48
4.3.3	Optimization methods	48
	Quasi-Newton	49
	FIRE	52
	SCMGO	54
	Limited-memory BFGS	55
	Conjugate gradients	56
4.3.4	Troubleshooting	56
	Failure to converge	56
	Restarting a geometry optimization	57
4.4	Transition state search	57
4.5	Linear Transit, PES scan	60
4.5.1	Troubleshooting	62
4.6	Nudged Elastic Band (NEB)	63
4.6.1	Input	64
4.6.2	Frozen atom constraints	69
4.6.3	Optimizations and convergence criteria	69
4.6.4	Output	70
4.6.5	Troubleshooting	70
4.7	Intrinsic Reaction Coordinate (IRC)	70
4.7.1	Method details	71
4.7.2	Input	72
4.7.3	Output	76
4.8	Excited state optimizations	76
4.9	Molecular dynamics	77
4.9.1	General	81
4.9.2	Constrained molecular dynamics	82
4.9.3	(Re-)Starting a simulation	83

4.9.4	Thermostats and barostats	84
	Temperature and pressure regimes	88
4.9.5	Trajectory sampling and output	89
4.9.6	Lattice deformations (volume regimes)	91
4.9.7	Molecule Gun: adding molecules during simulation	94
4.9.8	Removing molecules during simulation	98
4.9.9	Accelerated dynamics	101
	The PLUMED library support in AMS	101
	Metadynamics for Conformer-Rotamer Ensemble Sampling (CREST-MTD)	102
	Collective Variable-driven HyperDynamics (CVHD)	103
	Temperature Replica Exchange	106
	Bond Boost Method	108
	Force bias Monte Carlo (fbMC)	110
4.9.10	Non-equilibrium MD (NEMD)	112
	T-NEMD for thermoconductivity: heat exchange	112
4.10	Grand Canonical Monte Carlo (GCMC)	116
4.10.1	General info	116
4.10.2	Method Details	116
4.10.3	Input	118
4.10.4	Output	123
4.11	Automated PES Exploration	124
4.11.1	Overview	125
4.11.2	Job selection and main options	126
4.11.3	Results: the “Energy Landscape”	128
	Results on the text output	128
4.11.4	Continue a PES exploration from a previous calculation	129
4.11.5	Structure comparison	130
4.11.6	Process Search job	132
	Overview	132
	Input options	132
4.11.7	Saddle Search job	133
	Overview	133
	Input options	133
4.11.8	Basin Hopping job	135
	Overview	135
	Input options	135
4.11.9	Landscape Refinement	137
4.11.10	Optimizer	137
4.11.11	Binding Sites	138
	Input options	139
4.11.12	References	141
5	Gradients, Hessian, Stress tensor, Elasticity	143
5.1	Nuclear gradients	143
5.2	Hessian	144
5.3	PES point character	144
5.4	Thermodynamics, gas phase Gibbs free energy	147
5.5	Stress tensor	147
5.6	Elastic tensor	148
5.7	Numerical differentiation options	150
6	Vibrational Spectroscopy	153
6.1	General	153
6.1.1	Where are the results?	153

6.2	IR frequencies and normal modes	154
6.2.1	All vibrational Modes	154
	Rescanning Imaginary modes	155
6.2.2	Symmetric Displacements	156
6.2.3	Mobile Block Hessian (MBH)	157
6.2.4	Mode Scanning	159
	Theory	159
	Input	160
6.2.5	Mode Refinement	160
	Theory	161
	Input	162
6.2.6	Mode Tracking	163
	Theory	163
	Input	165
	Input: Tracking methods	167
	Input: Selecting modes	169
	Input: Convergence	170
6.2.7	Selecting modes	171
6.2.8	Thermodynamics (ideal gas)	174
	Gibbs free energy change for a gas phase reaction	176
6.2.9	Moments of inertia	177
6.2.10	Partial Vibrational Spectra (PVDOS)	177
6.3	Phonons	177
6.4	(Resonance) Raman	181
6.4.1	Raman	181
6.4.2	Resonance Raman: excited-state finite lifetime	182
6.4.3	Resonance Raman: VG-FC	183
	Theory	183
	Input	184
6.5	VROA: (Resonance) vibrational Raman optical activity	186
6.5.1	Engine ADF	187
6.6	VCD: Vibrational Circular Dichroism	187
6.6.1	Atomic polar tensor (APT) model	187
6.6.2	Analytical VCD in ADF	188
7	Vibrationally resolved electronic spectra	189
7.1	AH-FC: Adiabatic Hessian Franck-Condon	189
7.1.1	FCF module: Franck-Condon Factors	189
	Theory	190
	Input	191
	Result: TAPE61	192
7.1.2	FCF example absorption and fluorescence	193
7.1.3	FCF Example phosphorescence	195
7.2	VG-FC: Vertical Gradient Franck-Condon	196
7.2.1	Theory	197
	Theory: Vibronic-Structure Tracking	198
	Theory: Vibronic-Structure Refinement	200
	Theory: Adiabatic excitation energy	200
7.2.2	Input: Vibronic-Structure all modes	200
7.2.3	Input: Vibronic-Structure Tracking	201
	Input: Restarting VST	202
7.2.4	Input: Vibronic-Structure Refinement	202
7.2.5	Input: Excited State	204
7.2.6	Input: Producing the spectrum	205

8	Dipole moment, Polarizability, Bond orders	207
8.1	Charges, Dipole Moment, Polarizability	207
8.2	Bond orders & Molecule detection	208
9	Engines	211
9.1	Available engines	211
9.2	Summary of engine capabilities	212
9.3	External programs as engines	214
9.4	Toy engines	217
9.5	Engine add-ons	218
9.5.1	Dispersion corrections	218
9.5.2	Pressure	220
9.5.3	Non-isotropic external stress	221
9.5.4	Atom energies	223
9.5.5	Restraints	223
9.5.6	Wall potential	225
10	Utilities	227
10.1	ChemTraYzer	227
10.1.1	General information	227
10.1.2	Graphical user interface	227
10.1.3	Command line execution	227
10.1.4	Results	228
10.2	Trajectory Analysis	228
10.2.1	Radial Distribution Function (RDF)	230
	Description	230
	Options	230
10.2.2	Histogram	231
10.2.3	Autocorrelation Functions	233
	Description	236
	Options	236
10.2.4	Diffusion Coefficient	237
10.3	VCD Analysis: VCDtools	238
10.3.1	General Theory	239
10.3.2	General Coupled Oscillator Analysis	239
10.3.3	Available options	240
11	Examples	241
11.1	Geometry optimization	241
11.1.1	Example: Simple geometry optimization	241
11.1.2	Example: Two-stage geometry optimization with initial Hessian	241
11.1.3	Example: Periodic lattice optimization under pressure	243
11.1.4	Example: Phase Transition Due To External Nonuniform Stress	244
11.1.5	Example: Boron nitride optimization under external stress	245
11.1.6	Example: Graphene optimization under external stress	247
11.1.7	Example: Constrained optimizations	249
11.2	Transition state search	264
11.2.1	Example: TS search starting from initial Hessian	264
11.2.2	Example: PES scan and TS search for H2 on graphene	265
11.3	Nudged Elastic Band (NEB)	269
11.3.1	Nudged Elastic Band (NEB) Examples	269
	HCN isomerization reaction with NEB	269
	H2 dissociation on graphene	269
	Running multiple NEB calculations using PLAMS	269

11.4	Intrinsic reaction coordinate (IRC)	270
11.4.1	Example: IRC for HCN	270
11.4.2	Example: TS and IRC for Claisen reaction	271
11.5	PES scan	274
11.5.1	Example: Linear transit	274
11.5.2	Example: 2D PES scan	278
11.6	PES Exploration	282
11.6.1	Example: Basin Hopping for Ar 13 cluster	282
11.6.2	Example: PES Exploration, Process Search for alanine with PLAMS	283
11.6.3	Example: PES Exploration, Binding Sites for O on Pt 111	283
11.7	Molecular dynamics	287
11.7.1	Example: Simple MD for H2	287
11.7.2	Example: MD for a box of water	287
11.7.3	Example: Lattice deformations in MD	287
11.8	Vibrational analysis	293
11.8.1	Example: Mode Refinement	293
11.8.2	Example: Mode Tracking	295
11.8.3	Example: Vibronic-Structure Tracking	298
11.9	PES point properties	301
11.9.1	Example: Phonons for graphene	301
11.9.2	Example: Phonons with isotopes	302
11.9.3	Example: User-defined Brillouin zone for phonon dispersion	304
11.9.4	Example: Elastic tensor	306
11.10	Pipe interface	309
11.10.1	Example: ASE calculator as a pipe worker	309
11.10.2	Example: AMS as a pipe worker	310
12	Appendices	313
12.1	Extended XYZ file format	313
12.2	Developer options	313
12.3	Symmetry	315
12.3.1	Schönflies symbols and symmetry labels	315
12.3.2	Molecular orientation requirements	317
13	Required citations	319
13.1	General references	319
13.2	Feature references	319
13.2.1	Frequencies, IR Intensities, Raman, VCD	319
13.2.2	PES Exploration	320
14	External programs and Libraries	321
15	Keywords	323
15.1	Links to manual entries	323
15.2	Summary of all keywords	324

1.1 Overview

AMS is the new driver program introduced in the 2018 release of the Amsterdam Modeling Suite. The job of AMS is to handle all changes in the simulated system's geometry, e.g. during a geometry optimization or molecular dynamics calculation, using the so-called "engines" like ADF, BAND or DFTB for the calculation of energies and forces. In summary, one might say that the AMS driver steers the engines across the potential energy surface.

Prior to the 2018 release of the Amsterdam Modeling Suite, what we now call engines used to be separate programs, each with their own input and output files and formats. Starting with the 2018 release, the engines are only accessible through the AMS driver program, that provides a unified interface to all of them.

1.2 What's new in the AMS driver?

1.2.1 New in AMS2021.1

- *PES exploration* (page 124) tasks for the automated discovery of stationary points on the potential energy surface.
 1. *Process Search* (page 132): a composite method for finding escape mechanisms from a state. This will find both local minima and their connecting saddle points.
 2. *Basin Hopping* (page 135): a Monte Carlo method for finding local minima.
 3. *Saddle Search* (page 133): a single-ended method for finding nearby saddle points.
- *Force bias Monte Carlo* (page 110) is now available in AMS, enabling pure MC or mixed MD/MC simulations with any engine.
- The *molecule gun* (page 94) can now generate random shooting directions in a given cone.
- A new *wall potential* (page 225) engine add-on can be used to simulate spherical nanoreactors.
- The *D4 dispersion add-on* (page 218) can now also be used for calculations on periodic systems.
- Graceful interactive termination of running jobs through the *interactive input file* (page 13).
- *Geometry optimizations* (page 43) write the optimized geometry to the results directory as an *extended XYZ file* (page 313).

1.2.2 New in AMS2020.1

- ADF has been fully integrated as an AMS engine.

- New engines: *MLPotential* implementing several different type of machine learning (ML) potentials. A *Hybrid engine* for combining other engines in QM/MM calculations. Enhanced *ForceField* engine (formerly UFF).
- Improvements to the *Quasi-Newton geometry optimizer* (page 49): Periodic systems can now be optimized in delocalized coordinates. Better performance for systems made up of disconnected fragments.
- Geometry optimization can now be performed with *frozen and equal strain constraints* (page 48) for the lattice degrees of freedom. This option is currently only available with the *FIRE optimizer* (page 52).
- Support for *external electric fields* (page 35): Homogeneous as well as multipole charges.
- Enhanced support for *vibrational spectroscopy* (page 151): (resonance) Raman, (resonance) vibrational Raman optical activity, vibrational circular dichroism.
- The MD driver now supports time-dependent *lattice deformations* (page 91) as well as a method for accelerating bonding reactions (*Bond Boost method* (page 108)).
- Reactive MD calculations with the AMS driver can now be analyzed with *ChemTraYzer* (page 227).
- It is now possible to selectively disable writing some parts of a *MD trajectory* (page 89) to save space.
- More *trajectory analysis* (page 228) options: *autocorrelation functions* (page 233) and *diffusion coefficient* (page 237)
- The *molecular composition analysis* (page 208) can now be done with respect to an adsorption support region.
- The *transition state search* (page 57) task now allows the users to specify an approximate reaction coordinate (TSRC).
- For the calculation of normal modes, the *Mobile Block Hessian* (page 156) method now allows to treat parts of the system as rigid blocks.
- New coordinates for *constrained geometry optimizations* (page 46) and *PES scans* (page 60): Sum of distances and difference of distances.
- Added the ability to include additional potential terms (i.e. springs) through the *Restrains engine add-on* (page 223).
- A running AMS driver process can be used from Python through the new *AMSWorker class* in the *PLAMS* library. The communication between *PLAMS* and the AMS driver happens via the new *Pipe interface* (page 22) protocol.
- Introduced *Regions* (page 34) to simplify the input syntax for options that apply to a subset of atoms only.
- Symmetrization of systems from the input and the option to input a Z-matrix.
- Low frequencies contribution to *thermodynamics properties* (page 173) can be corrected using a free rotor interpolation method.

1.2.3 New in AMS2019.3

- The *Nudged elastic band method (NEB)* (page 63) for finding minimum energy paths of transitions has been added.
- The new *PES point character property* (page 144) can be used to quickly calculate a few of the lowest vibrational modes of a system and to verify the success of a geometry optimization or transitions state search.
- *Driver level parallelism* (page 18) is now enabled and managed automatically, improving the performance and scalability of many applications.
- A *Mode Scanning* (page 159) calculation can now be *started automatically* (page 155) for all modes within a specific frequency range.

- Methods for the quick calculation of the *vibrationally resolved electronic spectra* (page 188): *Vibronic-Structure Refinement* (page 200) and *Vibronic-Structure Tracking* (page 198).
- New geometry optimizer available: *Limited-memory BFGS* (page 55)
- Input keywords that expect lists of numbers can now be specified as *ranges using a Python slice-like notation* (page 10). Input keywords that expect a single real number now also accept fractions (of integers).
- New option to include a *non-isotropic external stress* (page 221) for 1D,2D and 3D periodic systems. This can be used to study structural deformation and mechanical properties of materials under non-isotropic stress.
- New *add-on system* (page 217) for manipulating and augmenting the results returned from the *engines* (page 209):
 1. Grimme's D4 and D3 dispersion corrections can be used with any engine through the *D4Dispersion and D3Dispersion add-ons* (page 218).

1.2.4 New in AMS2019.1

- *Intrinsic Reaction Coordinate (IRC) Scan* (page 70) is now available in the AMS driver for molecular and periodic systems.
- Support for the *Grand Canonical Monte Carlo (GCMC)* (page 116) method has been added in the AMS driver.
- Molecular composition analysis for molecular dynamics simulations (see [tutorial](#))
- *Collective Variable-driven Hyperdynamics (CVHD)* (page 103)
- *Molecule gun* (page 94) and *molecule sink* (page 98) for molecular dynamics
- *PLUMED library support* (page 101) for MD analysis and a wide variety of free energy methods
- The initial symmetry of a system is enforced during geometry optimizations with the Quasi-Newton optimizer.
- *Thermodynamic properties* (page 173) (assuming an ideal gas) are automatically computed after normal modes calculations.
- *Partial vibrational density of states (PVDOS)* (page 177) for normal modes.
- The system's symmetry is used to accelerate numerical nuclear derivatives and to provide symmetry labels for normal modes.
- The AMS driver starts up much faster, significantly speeding up scripting applications that launch AMS many times.
- New tools for mode selective vibrational analysis:
 1. *Mode Scanning* (page 159) (aka ADF's ScanFreq)
 2. *Mode Refinement* (page 160) (aka "Frequency range selection")
 3. *Mode Tracking* (page 163)

1.3 Motivation and progress

The Amsterdam Modeling Suite has grown substantially over the last decade, and in the 2017 release included programs implementing methods all the way from accurate density functional theory, through semi-empirical methods, to fast reactive force fields. Many of these programs have originally been developed by academic groups and are now maintained and expanded by SCM in collaboration with the original authors.

This rapid growth of the Amsterdam Modeling Suite had, however, led to a certain degree of unnecessary inhomogeneity within the suite: The input for the same task, e.g. a geometry optimization, differed quite a lot between the different programs in the suite. While this problem was mostly hidden for users of the graphical interface, it constituted a barrier for users of the new scripting frameworks such as PLAMS. Furthermore, the different programs produced rather different output files for the same task, making the automated extraction of results unnecessarily difficult. Finally, and most importantly, the rapid growth of the AMS suite had also led to a certain level of feature fragmentation, where some features were available in one program but not the other: ADF, for example, was able to do a linear transit calculation, while BAND was not. Constrained geometry optimization was supported in DFTB, but not in UFF. ReaxFF could be used for Grand Canonical Monte Carlo simulations, but DFTB could not.

In order to overcome these issues and make the Amsterdam Modeling Suite more powerful and user friendly, we introduced the AMS driver program with the 2018 release of the suite. The idea of this reorganization is to have only a single program called the AMS driver that under the hood uses the so-called “engines” like ADF, BAND and DFTB for the calculation of energies and gradients, where the engines are technically no longer separate programs but just libraries used by the AMS driver. In this way much of the input and output of AMS is the same, no matter which particular engine is used for a calculation. It also avoids the feature fragmentation, since any new feature in the AMS driver can immediately be used with all engines in the suite. Furthermore, the AMS driver also allows running *external programs as an engine* (page 213) providing energies and gradients, allowing end-users to perform all calculations supported by AMS with virtually any atomistic modeling program they have access to and to visualize the results in the graphical user interface of the Amsterdam Modeling Suite.

Converting all the programs of the Amsterdam Modeling Suite into engine libraries that are used by the AMS driver was a big reorganization of the entire suite, which was spread out over multiple releases between 2018 and 2020. As of the AMS2020 release, ADF itself has been integrated into the AMS driver, making the integration of all engines into the AMS driver finally complete. (The only exception is QuantumEspresso, which is integrated into the GUI, but has no connection to the AMS driver in the 2020 release.)

As with any large reorganization, it is unavoidable that some things change. For GUI users this should not create any issues, but users familiar with the existing command line and scripting interfaces will notice these changes and their existing workflows might need to be adjusted to the new setup. We know that these kind of changes can be disrupting for existing users, and where possible we try to keep backwards compatibility with previous versions, but unfortunately this is not always possible. However, overall AMS provides a much more consistent and convenient interface to command line and scripting users, and we believe that the new simplicity and expanded feature set of AMS make transitioning to the new framework well worth the effort.

1.4 Input, execution and output

With the introduction of AMS in the 2018 release of the Amsterdam Modeling Suite, there were some changes in the input and output files and formats used by our software. Users of the graphical interface should not notice these changes, but people using the software from the command line or through the scripting frameworks need to be aware of them.

This section contains the main changes. For a more complete description see:

- The extended *Input, execution and output section* (page 7).

Generally the input for AMS has the block and keyword structure that most programs in the Amsterdam Modeling Suite have already been using. See the *General remarks on input structure and parsing* (page 9) section for more details. The only new construct in the AMS input is a special `Engine` block, that selects which engine is used for the simulation and also contains all the details of its configuration. This is probably best illustrated by an example. Let us look at the following AMS input, which optimizes the geometry of the methane molecule and calculates its normal modes of vibration at the optimized geometry:

```
$AMSBIN/ams << EOF
  Task GeometryOptimization
```

(continues on next page)

(continued from previous page)

```

GeometryOptimization
  Convergence
    Gradients 1.0e-4
  End
End

Properties
  NormalModes true
End

System
  Atoms
    C      0.00000000    0.00000000    0.00000000
    H      0.63294000   -0.63294000   -0.63294000
    H     -0.63294000    0.63294000   -0.63294000
    H      0.63294000    0.63294000    0.63294000
    H     -0.63294000   -0.63294000    0.63294000
  End
End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-3-1
EndEngine
EOF

```

Note how DFTB is selected as the engine in the `Engine DFTB` line that opens the Engine block. All DFTB specific configuration is contained within this engine block, which is terminated by `EndEngine`. The fact that we want to run a geometry optimization with normal modes for methane and things like convergence criteria for the optimization are of course completely independent from which engine is actually used to perform this calculation. Therefore they are all found *outside* of the Engine block. In this sense, the AMS input is split up into the driver level input (everything outside of the engine block) and the engine input, which is just a single Engine block. This separation makes it easy to perform the same calculation at a different level of theory, by simply switching out the Engine block in the input. We could, for example, repeat the same calculation at the DFT-GGA level using the Band engine:

```

Engine BAND
  XC
    GGA PBE
  End
EndEngine

```

Engines like ADF and BAND that have many options and can calculate many properties, consequently also have a large number of possible keywords in their input. In order to have a better structured documentation we have split off the description of the engine inputs into separate *engine specific manuals* (page 209), while this AMS manual only documents the driver level keywords outside of the Engine block. All the engine specific options are found in the respective engine's manual, which documents the keywords in its Engine block. In general all engines can be used with all tasks in AMS. There are only a few rather obvious restrictions, for example that only engines which can handle periodic systems can be used for the calculation of phonons.

The introduction of the Engine block is the only real change AMS brings to the input side of things. On the output side there are a few more changes.

The first change to the output is that AMS does not put any of its output files into the present working directory, as virtually all of the standalone programs in the suite did. Instead AMS creates a `*.results` directory, which collects all result file associated with a job. Here `*` is replaced by the jobname, which is set with the `AMS_JOBNAME`

environment variable:

```
AMS_JOBNAME=methane $AMSBIN/ams << EOF
... see above ...
EOF
```

This would put all results related to our geometry optimization of methane into the newly created folder `methane.results`. (The default name of the results folder is `ams.results` if `AMS_JOBNAME` is not set, see the *environment variables* (page 17) section of this manual for documentation of all environment variables used by AMS.) In this way users can easily run multiple jobs in the same directory without danger of clashing output files, which was a common problem before the introduction of AMS. This new setup is also more consistent with the graphical user interface, which already collected all files associated with a specific job into a dedicated results directory. Note that AMS will by default **not overwrite** results directories if a job is rerun or another job is run with the same jobname.

Inside of the results directory users will always find the logfile `ams.log`, which is written during a running calculation and can be used to monitor its progress. Furthermore the results directory contains binary result files in the KF format, which can be opened and inspected with the KFBrowser GUI component.

- The main `ams.rkf` written by the AMS driver. It contains high level information about the trajectory that the AMS driver took over the potential energy surface. For example, for a molecular dynamics simulation it would contain the full trajectory. The format in which this information is written is independent from which engine was used for a calculation.
- The engine specific main binary output file written by the engine (and partly by the AMS driver). This file is kept for only one special point, e.g. the final geometry in a geometry optimization. For example, for ADF this engine file is called `adf.rkf` (instead of `TAPE21` in ADF<=2019), for BAND `band.rkf`, for DFTB `dftb.rkf`. If a property, like vibrational modes, is tied to the special point on the potential energy surface, it is stored in this file. Also all engine specific properties are written to this file, like orbitals in case of a quantum mechanical engine.
- Additionally there might be a binary output file for every point on the potential energy surface that was visited during the calculation.
- Other engine specific binary (and ASCII) output files written by the engine.

Having multiple different binary output files could be confusing for people that are used to the single result file that was written by the standalone programs in ADF<=2017. After all, it brings up the question in which file the desired property is stored. The general rule is: if the property is tied to a particular point on the potential energy surface, it is stored in the engine output file belonging to that particular point. If the information depends on the entire trajectory over the PES, it is found in the main `ams.rkf` written by the AMS driver.

INPUT, EXECUTION AND OUTPUT

2.1 Input

The input for AMS has a block and keyword structure. See the *General remarks on input structure and parsing* (page 9) section for more details.

The input keys *System* (page 29), *Task* (page 8), and *Engine* (page 209), are obligatory. Most other input keys in AMS are optional, like *Properties* (page 8). In the rest of this manual one can find all input keys for AMS.

The Engine specific input can be found in the respective Engine Manuals, for example, the manual for *ADF*, *BAND*, *DFTB*, *ForceField*, *MOPAC*, and *ReaxFF*.

2.1.1 Example

The AMS input has a special *Engine* block, that selects which engine is used for the simulation and also contains all the details of its configuration. This is probably best illustrated by an example. Let us look at the following AMS input, which optimizes the geometry of the methane molecule and calculates its normal modes of vibration at the optimized geometry:

```

$AMSBIN/ams << EOF
Task GeometryOptimization

GeometryOptimization
  Convergence
    Gradients 1.0e-4
  End
End

Properties
  NormalModes true
End

System
  Atoms
    C      0.00000000    0.00000000    0.00000000
    H      0.63294000   -0.63294000   -0.63294000
    H     -0.63294000    0.63294000   -0.63294000
    H      0.63294000    0.63294000    0.63294000
    H     -0.63294000   -0.63294000    0.63294000
  End
End

Engine DFTB

```

(continues on next page)

(continued from previous page)

```

Model DFTB3
ResourcesDir DFTB.org/3ob-3-1
EndEngine
EOF

```

Note how DFTB is selected as the engine in the `Engine DFTB` line that opens the `Engine` block. All DFTB specific configuration is contained within this engine block, which is terminated by `EndEngine`. The fact that we want to run a geometry optimization with normal modes for methane and things like convergence criteria for the optimization are of course completely independent from which engine is actually used to perform this calculation. Therefore they are all found *outside* of the `Engine` block. In this sense, the AMS input is split up into the driver level input (everything outside of the engine block) and the engine input, which is just a single `Engine` block. This separation makes it easy to perform the same calculation at a different level of theory, by simply switching out the `Engine` block in the input. We could, for example, repeat the same calculation at the DFT-GGA level using the Band engine:

```

Engine BAND
  XC
  GGA PBE
End
EndEngine

```

Engines like ADF or BAND that have many options and can calculate many properties, consequently also have a large number of possible keywords in their input. In order to have a better structured documentation we have split off the description of the engine inputs into separate *engine specific manuals* (page 209), while this AMS manual only documents the driver level keywords outside of the `Engine` block. All the engine specific options are found in the respective engine's manual, which documents the keywords in its `Engine` block. In general all engines can be used with all tasks in AMS. There are only a few rather obvious restrictions, for example that only engines which can handle periodic systems can be used for the calculation of phonons.

2.1.2 Tasks

The key `Task` is described in more detail in the section *Structure, Reactivity, and Molecular Dynamics* (page 41). The `Task VibrationalAnalysis` is described in more detail in the section *Vibrational Spectroscopy* (page 151) (Mode Scanning, Mode Refinement, Mode Tracking, VG-FC Resonance Raman) and in the section *Vibrationally resolved electronic spectra* (page 188) (VG-FC: Vertical Gradient Franck-Condon).

Below the possible arguments for `Task` are given.

```

Task [SinglePoint | GeometryOptimization | TransitionStateSearch | IRC | PESScan | ↵
↵NEB | VibrationalAnalysis | MolecularDynamics | GCMC | PESExploration]

```

2.1.3 Properties

The block key `Properties` is described in more detail in the section *Gradients, Hessian, Stress tensor, Elasticity* (page 141), in the section *Vibrational Spectroscopy* (page 151), and in the section *Dipole moment, Polarizability, Bond orders* (page 206).

Below the possible `Properties` are summarized.

```

Properties
BondOrders Yes/No
Charges Yes/No
DipoleGradients Yes/No
DipoleMoment Yes/No

```

(continues on next page)

(continued from previous page)

```

ElasticTensor Yes/No
Gradients Yes/No
Hessian Yes/No
Molecules Yes/No
NormalModes Yes/No
Other Yes/No
PESPointCharacter Yes/No
Phonons Yes/No
Polarizability Yes/No
Raman Yes/No
SelectedRegionForHessian string
StressTensor Yes/No
VCD Yes/No
VROA Yes/No
End

```

2.1.4 General remarks on input structure and parsing

- Most keys are optional. Defaults values will be used for keys that are not specified in the input
- Keys/blocks can either be *unique* (i.e. they can appear in the input only once) or *non-unique*. (i.e. they can appear multiple times in the input)
- The order in which keys or blocks are specified in the input does not matter. Possible exceptions to this rule are a) the content of non-standard blocks b) some non-unique keys/blocks)
- Comments in the input file start with one of the following characters: #, !, ::

```

# this is a comment
! this is also a comment
:: yet another comment

```

- Empty lines are ignored
- The input parsing is **case insensitive** (except for string values):

```

# this:
UseSymmetry false
# is equivalent to this:
USESMMETRY FALSE

```

- Indentation does not matter and multiple spaces are treaded as a single space (except for string values):

```

# this:
    UseSymmetry    false
# is equivalent to this:
UseSymmetry false

```

2.1.5 Keys

Key-value pairs have the following structure:

```
KeyName Value
```

Possible types of keys:

bool key The value is a single Boolean (logical) value. The value can be `True` (equivalently `Yes`) or `False` (equivalently `No`). Not specifying any value is equivalent to specifying `True`. Example:

```
KeyName Yes
```

integer key The value is a single integer number. Example:

```
KeyName 3
```

float key The value is a single float number. For scientific notation, the E-notation is used (e.g. -2.5×10^{-3} can be expressed as `-2.5E-3`). The decimal separator should be a dot (`.`), and **not** a comma (`,`). Example:

```
KeyName -2.5E-3
```

Note that fractions (of integers) can also be used:

```
KeyName 1/3      (equivalent to: 0.333333333333...)
```

string key The value is a string, which can include white spaces. Only ASCII characters are allowed. Example:

```
KeyName Lorem ipsum dolor sit amet
```

multiple_choice key The value should be a single word among the list options for that key (the options are listed in the documentation of the key). Example:

```
KeyName SomeOption
```

integer_list key The value is list of integer numbers. Example:

```
KeyName 1 6 0 9 -10
```

Note that one can also specify ranges of integers by specifying the interval and (optionally) the step size separated by colons:

```
KeyName 1:5      (equivalent to: 1 2 3 4 5)
KeyName 2:10:2   (equivalent to: 2 4 6 8 10)
KeyName 20:10:-2 (equivalent to: 20 18 16 14 12 10)
```

Note also that ranges can be freely combined with individual numbers:

```
KeyName 1:5 10 20 (equivalent to: 1 2 3 4 5 10 20)
```

float_list key The value is list of float numbers. The convention for float numbers is the same as for Float keys. Example:

```
KeyName 0.1 1.0E-2 1.3
```

Float lists can also be specified as a range with equidistant points, by specifying the interval's boundaries (inclusive) as well as the number of desired subintervals separated by colons:

```
KeyName 1.0:1.5:5 (equivalent to: 1.0 1.1 1.2 1.3 1.4 1.5)
```

Range specifications can be freely combined with each other and single numbers:

```
KeyName 0.0 1.0:1.5:5 2.0:3.0:10
```

2.1.6 Blocks

Blocks give a hierarchical structure to the input, grouping together related keys (and possibly sub-blocks). In the input, blocks generally span multiple lines, and have the following structure:

```
BlockName
  KeyName1 value1
  KeyName2 value2
  ...
End
```

Headers

For some blocks it is possible (or necessary) to specify a *header* next to the block name:

```
BlockName someHeader
  KeyName1 value1
  KeyName2 value2
  ...
End
```

Compact notation

It is possible to specify multiple key-value pairs of a block on a single line using the following notation:

```
# This:
BlockName KeyName1=value1 KeyName2=value2

# is equivalent to this:
BlockName
  KeyName1 value1
  KeyName2 value2
End
```

Notes on compact notation:

- The compact notation cannot be used for blocks with headers.
- Spaces (blanks) between the key, the equal sign and the value are ignored. However, if a value itself needs to contain spaces (e.g. because it is a list, or a number followed by a unit), the entire value must be put in either single or double quotes:

```
# This is OK:
BlockName Key1=value Key2 = "5.6 [eV]" Key3='5 7 3 2'
# ... and equivalent to:
BlockName
  Key1 value
  Key2 5.6 [eV]
  Key3 5 7 3 2
End

# This is NOT OK:
BlockName Key1=value Key2 = 5.6 [eV] Key3=5 7 3 2
```

Non-standard Blocks

A special type of block is the *non-standard block*. These blocks are used for parts of the input that do not follow the usual key-value paradigm.

A notable example of a non-standard block is the `Atoms` block (in which the atomic coordinates and atom types are defined).

2.1.7 Including an external file

You can include an external ASCII file in the input with the `@include` directive:

```
@include FileName.in
@include "file name with spaces.in"
```

The file name should include the path, either absolute or relative to the run-directory. The content of the file is included in the input at the point where the `@include` directive occurs. The `@include` directive may occur any number of times in the input.

The `@include` feature makes it easy to pack your preferred settings in one file and use them in every run with minimum input-typing effort.

2.1.8 Units

Some keys have a default unit associated (not all keys have units). For such keys, the default unit is mentioned in the key documentation. One can specify a different unit within square brackets at the end of the line:

```
KeyName value [unit]
```

For example, assuming the key `EnergyThreshold` has as default unit `Hartree`, then the following definitions are equivalent:

```
# Use defaults unit:
EnergyThreshold 1.0

# use eV as unit:
EnergyThreshold 27.211 [eV]

# use kcal/mol as unit:
EnergyThreshold 627.5 [kcal/mol]

# Hartree is the atomic unit of energy:
EnergyThreshold 1.0 [Hartree]
```

Available units:

- **Energy:** Hartree, Joule, eV, kJ/mol, kcal/mol, cm1, MHz
- **Length:** Bohr, Angstrom, meter
- **Angles:** radian, degree
- **Mass:** el, proton, atomic, kg
- **Pressure:** atm, Pascal, GPa, a.u., bar, kbar
- **Electric field:** V/Angstrom, V/meter, a.u.

2.2 Execution

2.2.1 Shell script

The AMS driver reads its input from standard input, i.e. what is called `STDIN` on Unix-like systems. Technically it is possible to run AMS and type the input file in interactively. This is however highly impractical and most people run AMS from a small shell script that contains the AMS text input and sends it directly to the AMS executable. For example, the content of the file ‘example.run’ could be like:

```
#!/bin/sh

$AMSBIN/ams << EOF

... AMS text input goes here:

Block
  Keyword value
  OtherKeyword value
End

EOF
```

The shell script ‘example.run’ needs to be executable, if it isn’t you will need to make it executable, e.g. `chmod u+x example.run`. The ‘example.run’ file needs to be executed as a shell script, not as input to AMS.

```
./example.run > example.out
```

2.2.2 Interactive input file

As of the 2021.1 version the AMS driver has the concept of an interactive input file. The interactive input file is a second input file, that users may create in the *results directory* (page 16) of an already running job to interact with it. The name of the interactive input file is `interactive.in`. Note that the file does not exist by default, but has to be created by the user wanting to interact with the running AMS job.

In AMS2021.1 the interactive input file may be used to request the graceful and orderly termination of a job.

```
echo "Stop" > ams.results/interactive.in
```

Stop

Type Bool

Default value No

Description Makes the AMS driver stop execution in an ordered manner at the next opportunity. This will not interrupt the execution in the middle of the numerical calculation of a PES point property (e.g. a Hessian).

It is generally preferable to request the termination through the interactive input file over killing a job at the operating system level. This makes sure that the job stops at a logical point and still has time to write intermediate results to disk, making restarting from such jobs easier and more reliable. From the graphical user interface, the stopping can be requested in AMSjobs by selecting the running job and clicking **Job** → **Request Early Stop** in the menu bar.

As of AMS2021.1, the tasks that can be stopped through the interactive input file are:

- *Molecular dynamics* (page 77) simulations will stop at the next sampling or checkpoint as if the maximum number of steps had been reached.

- *Geometry optimizations* (page 43) will stop after the current iteration and be considered converged. This can be used if a geometry optimization does not make progress anymore, e.g. because the convergence thresholds are set tighter than the noise level of the used engine.
- *Nudged elastic band* (page 63) calculations will stop after the current iteration (i.e. the calculations for all images) is complete.
- *PES scans* (page 60) will interrupt the currently running geometry optimization and skip the optimization for all further PES points.
- *PES explorations* (page 124) will terminate after the currently running expedition. No new explorers will be dispatched, but currently active explorers will be allowed to finish.

Note that the `Stop` keyword in the interactive input file will not interrupt the job in the middle of the numerical calculation of a *PES point property* (page 206), such as a Hessian or stress tensor. It will also never stop a job that is currently running an engine, and will hence not stop an engine that is calculating these properties analytically. It is possible to interrupt the numerical calculation of *PES point properties* (page 206) using the `StopProperties` keyword. We advise users to be careful with this keyword and always combine it with the `Stop` keyword, as it is generally not possible to skip the evaluation of properties, but continue with a task as the AMS driver level. (After all, the driver may need the properties that we interrupt.)

```
echo "Stop" > ams.results/interactive.in
echo "StopProperties" >> ams.results/interactive.in
```

StopProperties

Type Bool

Default value No

Description Makes the AMS driver stop execution during the evaluation of PES point properties. This may for example be used to interrupt the numerical calculation of a Hessian. Note that this just works at the level of the loops used for the `**numerical**` calculation of the properties, so engines which calculate a property analytically may not be interruptible using this keyword. It is recommended to use this keyword in combination with the generic `Stop` keyword.

As of AMS2021.1 it is not possible to restart the calculation of properties that were interrupted like this. Hence the use of the `StopProperties` keyword is limited.

2.2.3 Running AMS on compute clusters

AMS is parallelized with MPI and can therefore be run in parallel on distributed memory machines, aka compute clusters. See the [installation manual](#) for general documentation on how to set up and run all the programs from the Amsterdam Modeling Suite on compute clusters. In this section we give some more advice that is specific to the AMS driver and its engines.

Normally users use the login node to prepare their jobs and input files somewhere in their home directory, and also want the results of their jobs to end up there. Quite often, compute clusters are set up such that the user's home directory is also mounted on the compute nodes, usually via NFS (Network File System). Before the introduction of the AMS driver it was not recommended to `cd` to the home directory in the submission script and have the compute nodes execute the job directly there. This was simply due to the fact that a lot of file I/O was done on temporary files in the present working directory, which in this case would be on a slow network-mounted file system.

On the other hand, with AMS, switching to the home directory is the preferred way of running on a cluster where the home directory is mounted on the compute nodes. Running in the home directory mounted over NFS does not come with a performance penalty for AMS, but has many advantages. This is because AMS and its engines are already built under the assumption that access to this directory is slow. Basically there are three directories that are used by the AMS driver and its engines:

1. The **starting directory**, i.e. the present working directory at the time the AMS driver is started. This folder is generally read-only for AMS, except for creating the results directory there at the beginning of a calculation. Note that all relative paths in the AMS input, e.g. for loading results from previous calculations, are relative to the starting directory. The starting directory is assumed to be on a slow filesystem, but since data is normally only read once from there in the beginning of a calculation, this is in practice not a problem.
2. The **results directory**, where the results of a calculation as well as important intermediate steps (e.g. restart files) are collected. It also contains the log file which can be used to monitor a running calculations. The results directory is assumed to be on a slow filesystem, so AMS and its engines will be very careful not to do much disk I/O there. Generally something is only written to the results directory when AMS is sure that it should remain on disk when the calculation finishes. The results directory can also contain some intermediate restart files, so the contents of the result directory should be all that is needed in case the calculation crashes or is killed before it finishes normally.
3. The **scratch directory**, the location of which is set with the `$SCM_TMPDIR` environment variable, see also the [installation manual](#). This directory should be put on a fast disk, e.g. an SSD in the compute node, as it will be used to store temporary results on disk. Users do not really need to care or know about the temporary files in the scratch directory. Normally, any files and directories created in the scratch directory are cleaned up at the end of the calculation. In case of errors, AMS tries to copy anything useful (e.g. the text output of all the different ranks) to the results directory in order to make finding the problem easier. However, for some kinds of crashes (or if the `SIGKILL` signal is sent to AMS), the cleanup of the scratch directory might not be performed, in which case users might want to manually check or remove the `amstmp_*` folders in the scratch directory.

With this setup there is no performance penalty for running directly on a network mounted home directory: Results will just be put there immediately, instead of being copied there at the end of a calculation.

Normally all batch systems provide an environment variable that is set to the directory from which the job was submitted, which is then where one should `cd` in the run script:

```
#!/bin/sh

if [ -z "$PBS_O_WORKDIR" ]; then
  # PBS batch system
  cd "$PBS_O_WORKDIR"
elif [ -z "$SLURM_SUBMIT_DIR" ]; then
  # Slurm batch system
  cd "$SLURM_SUBMIT_DIR"
elif [ -z "..." ]; then
  # add other batch systems as necessary ...
  cd "..."
fi

export AMS_JOBNAME=myJob

$AMSBIN/ams << EOF

# Normal AMS text input, but with all paths
# relative to where the job was submitted from, e.g.:
LoadSystem previousJob.results/ams.rkf

EOF
```

With this runscript the AMS driver would make a `myJob.results` folder in the directory where the job was submitted from, and there is no need to copy results around manually in the run script. Furthermore this runscript always produces exactly the same files in the same locations, no matter if it is run interactively or submitted to a compute node through the batch system. Furthermore all paths in the input file can be specified relative to the location from where the runscript is submitted (normally the folder in which the runscript is located). This removes the need to copy or specify absolute paths to previous results, e.g. when restarting calculations. Finally, files useful for monitoring the

running calculation are also conveniently there and not hidden somewhere on the compute node.

2.3 Output

AMS produces two ASCII files: standard output and the `ams.log` file, and produces several binary output data files. The `ams.log` file is a very concise summary of the calculation's progress during the run. The binary output data files contain job characteristics and computational results produced by AMS and the Engine. Part of what is written to the binary output files is also written in a human readable form to standard output.

2.3.1 Results directory

Note that AMS does not put any of its binary output files and the `ams.log` file into the present working directory, as virtually all of the standalone programs in the suite did. Instead AMS creates a `*.results` directory, which collects all result file associated with a job. Here `*` is replaced by the jobname, which is set with the `AMS_JOBNAME` environment variable:

```
AMS_JOBNAME=methane $AMSBIN/ams << EOF

... see Input example before ...

EOF
```

This would put all results related to our geometry optimization of methane into the newly created folder `methane.results`. (The default name of the results folder is `ams.results` if `AMS_JOBNAME` is not set, see the *environment variables* (page 17) section of this manual for documentation of all environment variables used by AMS.) In this way users can easily run multiple jobs in the same directory without danger of clashing output files, which was a common problem before the introduction of AMS. This new setup is also more consistent with the graphical user interface, which already collected all files associated with a specific job into a dedicated results directory. Note that AMS will by default **not overwrite** results directories if a job is rerun or another job is run with the same jobname.

2.3.2 Logfile `ams.log`

Inside of the results directory users will always find the logfile `ams.log`, which is written during a running calculation and can be used to monitor its progress.

The logfile `ams.log` is generated during the calculation and flushed after (almost) each message that is sent to it by the program. Consequently, the user can inspect it and see what is going on without being delayed by potentially large system I/O buffers. Each message contains date and time of the message plus additional info.

Be alert on error messages. Take them seriously: inspect the standard output carefully and try to understand what has gone wrong. Be also alert to warnings. They are not necessarily fatal but you should understand what they are about before being satisfied with the results of the calculation. Do not ignore them just because the program has not aborted: in some cases the program may not be able to determine whether or not you really want to do what appears to be wrong or suspicious. If you believe that the program displays erratic behavior, then the standard output file may contain more detailed information. Therefore, in such case save the complete standard output file, together with the logfile `ams.log`, in case we need these files for further analysis.

2.3.3 Binary output files

The results directory contains binary result files in the KF format, which can be opened and inspected with the KF-Browser GUI component. KF stands for Keyed File: KF files are keyword oriented, which makes them easy to process by simple procedures. KF files are Direct Access binary files.

- The main `ams.rkf` written by the AMS driver. It contains high level information about the trajectory that the AMS driver took over the potential energy surface. For a geometry optimization it would for example contain the history of how the systems geometry changed during the optimization as well as the final optimized geometry. For a molecular dynamics simulation it would contain the full trajectory. The format in which this information is written is independent from which engine was used for a calculation.
- The engine specific main binary output file written by the engine (and partly by the AMS driver). This file is kept for only one special point, e.g. the final geometry in a geometry optimization. The ADF engine writes `adf.rkf` (instead of `TAPE21` in older versions). The BAND engine writes `band.rkf` (instead of `RUNKF` in older versions). The DFTB engine writes `dftb.rkf`. If a property, like vibrational modes, is tied to this special point on the potential energy surface, it is stored in this file. Also all engine specific properties are written to the main binary output file, like orbitals in case of a quantum mechanical engine.

Table 2.1: Engine specific main binary output file

Engine	main file
ADF	<code>adf.rkf</code>
BAND	<code>band.rkf</code>
DFTB	<code>dftb.rkf</code>
ForceField	<code>forcefield.rkf</code>
MOPAC	<code>mopac.rkf</code>
ReaxFF	<code>reaxff.rkf</code>
External	<code>external.rkf</code>

- Additionally there might be an engine specific binary output file for every point on the potential energy surface that was visited during the calculation. Like the engine specific main binary output file they contain information tied to a specific point on the potential energy surface. These engine output files all have the extension `.rkf`, but their filename is usually somehow descriptive of the point on the PES that they correspond to. Note that one does not always get an engine output file for every PES point that was visited during the calculation. For most applications this would just be too much data.
- Other engine specific binary (and ASCII) output files written by the engine.

Having multiple different binary output files could be confusing for people that are used to the single result file that was written by the standalone programs in ADF<=2017. After all, it brings up the question in which file the desired property is stored. The general rule is: If the property is tied to a particular point on the potential energy surface, it is stored in the engine output file belonging to that particular point. This includes the Hessian, stress tensor, elastic tensor, normal modes of vibration, phonons, Raman intensities and other vibrational properties. If the information depends on the entire trajectory over the PES, it is found in the main `ams.rkf` written by the AMS driver.

2.3.4 Standard output

The standard output file contains in a human readable form part of the job characteristics and computational results produced by AMS and the Engine.

2.3.5 Optimized geometry in xyz format

In case of *Geometry Optimizations* (page 43), the optimized geometry is saved as an *extended XYZ file* (page 313) in the results directory with the name `output.xyz`.

Note that the optimized geometry is also stored in the binary file `ams.rkf` and is printed to the standard output.

2.3.6 AMS environment variables

The behavior of AMS related to the output can be modified through a number of environment variables.

AMS_JOBNAME Sets the name of a job. This name is used to determine the name of the results folder AMS creates, which is `$AMS_JOBNAME.results` or `ams.results` if this environment variable is not set.

AMS_RESULTSDIR If this environment variable is set, instead of creating a new results folder, AMS will use the set directory as the results folder. Note that the directory set here will *not* be created by AMS and therefore has to exist before starting AMS. Note that this environment variable can be used to prevent AMS from creating result folders, by setting `AMS_RESULTSDIR=.` This reproduces the pre-AMS behavior of putting all result files into the directory from which a job is started.

AMS_SWITCH_LOGFILE_AND_STDOUT If this environment variable is set, AMS will redirect what is normally printed on standard output to a file (`ams.out`) in the results directory. Instead the contents of the log file (`ams.log`) will be printed to standard output while a job is running, allowing users to easily monitor the jobs progress. Note that the log file will still be created normally as if this environment variable was not set. This environment variable is just a convenience feature for users that would always redirect their output into a file and then use `tail -f` on the log file to monitor the running calculation.

This is an example run-script using the `AMS_SWITCH_LOGFILE_AND_STDOUT` and `AMS_JOBNAME` environment variables:

```
#!/bin/sh

# By setting AMS_SWITCH_LOGFILE_AND_STDOUT, the (more compact)
# logfile will be printed to standard output while the full
# text output of the calculation is redirected to the file
# ams.out in the ams results folder

export AMS_SWITCH_LOGFILE_AND_STDOUT=true

# By default ams creates a folder 'ams.results' and puts the
# results of the calculation there. If we set AMS_JOBNAME, the
# results folder will instead be called $AMS_JOBNAME.results
# (in this case, 'H2_optimization.results')

AMS_JOBNAME=H2_optimization $AMSBIN/ams <<eor
  System
    Atoms
      H 0 0 0
      H 0 0 1
    End
  End
  Task GeometryOptimization
  Engine DFTB
    Model GFN1-xTB
  EndEngine
eor
```

2.4 Driver level parallelism

See also:

See also the [GUI tutorial](#) on the parallel scalability of the calculation of elastic tensors.

AMS is a parallel program using MPI for efficient execution on distributed memory machines, aka compute clusters. For most jobs, the AMS driver part of a calculation is computationally not particularly costly and most of the execution time is spent inside of the *compute engines* (page 209). Therefore the main parallelization of AMS is inside of the engines, making sure that a good performance is obtained for *tasks* (page 41) such as *molecular dynamics* (page 77) or *geometry optimizations* (page 43), which consist of a series of interdependent engine invocations: We need to have completed step n before we can continue with step $n + 1$.

However, not all workloads are of this sequentially dependent type. Some jobs have a lot of independent work, that can be done in parallel. This kind of trivial parallelizability can be exploited at the AMS driver level: Instead of having all cores collaborate on a single PES point and then doing all needed PES points sequentially, we can just distribute the available PES points over the all the available cores. Normally this leads to a better parallel scaling than the default parallelization inside of the engines: Parallelizing the engines is relatively complicated and often requires a lot of communication between cores. Parallelizing on the driver level on the other hand is very easy, and often the only communication required is at the very end of the calculation, when results are collected.

Note that it is perfectly possible to combine both the in-engine parallelization and the driver level parallelism: At the driver level we could split our e.g. in total 32 cores into 4 groups of 8 cores, and then have each group of 8 use the in-engine parallelization to collaborate on a specific calculation. This is especially useful if the total number of cores is larger than then number of independent calculations we have to do. It might also be that we have a very large number of calculations to do, but not enough memory to let every core work alone on its own calculation, as would be ideal from a parallel scaling point of view.

Because of the two levels of parallelism – both at the driver and the engine level – we call this setup **double parallelization**.

Starting with the AMS2019.3 release, driver level parallelism is used and configured automatically. That means that the AMS driver will automatically parallelize at the driver level when it is possible and considered advantageous. As such it should normally not be necessary for users to explicitly configure the driver level parallelism.

Driver level parallelism can be used for the calculation of the *PES point properties* (page 141) which are derivatives, if these need to be done numerically:

- Numerical calculation of *forces / nuclear gradients* (page 143). With a double sided derivative this requires $6 \times n_{\text{atoms}}$ independent calculations on geometries with one atom displaced along a cartesian coordinate.
- Numerical calculation of the *stress tensor* (page 147) for periodic systems. This requires up to 12 calculations for a double sided derivative along the 6 strain directions, but might require less in case some of the strains are symmetry equivalent.
- Numerical calculation of the *Hessian* (page 144) and normal modes of vibration. This is currently only supported for engines that calculate nuclear gradients analytically and done by numerically differentiating this first (analytic) derivative. As such it requires $6 \times n_{\text{atoms}}$ independent calculations on geometries with one atom displaced along a cartesian coordinate.
- Numerical calculation of the *elastic tensor* (page 148). This requires 84 independent geometry optimizations on systems with differently strained lattices, with each optimization having a variable number of steps.
- Numerical calculation of *phonons* (page 177). This requires at most $6 \times n_{\text{atoms}}$ displacements, but might require less in case some of the displacements are symmetry equivalent. Note that the displacements are done in a super cell system, which for many engines will increase the memory requirements, but also improve the in-engine parallel scalability.
- The forward and backward displacements along normal modes for the *Mode Scanning* (page 159), *Mode Refinement* (page 160), and *Mode Tracking* (page 163).

There are also tasks using driver level parallelism, e.g. *Nudged Elastic Band* (page 63), for which the calculations of all the images is trivially parallel.

Details of the driver level parallelism, i.e. how much to parallelize at the driver level, are generally configured for the above mentioned cases on an individual basis, because one might want a different grouping strategy for each case. For

each case there is a separate `Parallel` block somewhere in the input (e.g. `ElasticTensor%Parallel` for the calculation of the elastic tensor), which has the following keywords:

```
Parallel
  nGroups integer
  nCoresPerGroup integer
  nNodesPerGroup integer
End
```

Note that only one of them should be specified in the input, depending of course on what is the desired strategy for parallelization.

nGroups n Splits all cores evenly into n groups. We recommend choosing n such that it divides the total number of cores without a remainder.

nCoresPerGroup n Each group consists of n cores. As such `nCoresPerGroup 1` results in the maximum possible parallelism at the driver level. We recommend choosing n such that it divides the total number of cores without a remainder.

nNodesPerGroup n Makes groups from all cores within n nodes, e.g. `nNodesPerGroup 1` would make every cluster node into a separate group. Note that this option should *only be used on homogeneous compute clusters*, where all used nodes have the same number of cores. Otherwise cores from different nodes will be grouped together in very surprising and unintended ways, probably resulting in suboptimal performance.

The optimal grouping strategy and number of groups depends on the total number of cores used in the calculation, the amount of independent tasks to be done in parallel, as well as the parallel scalability of the engine itself. In practice it can be a bit tricky, which is why the grouping strategy is determined automatically since AMS2019.3.

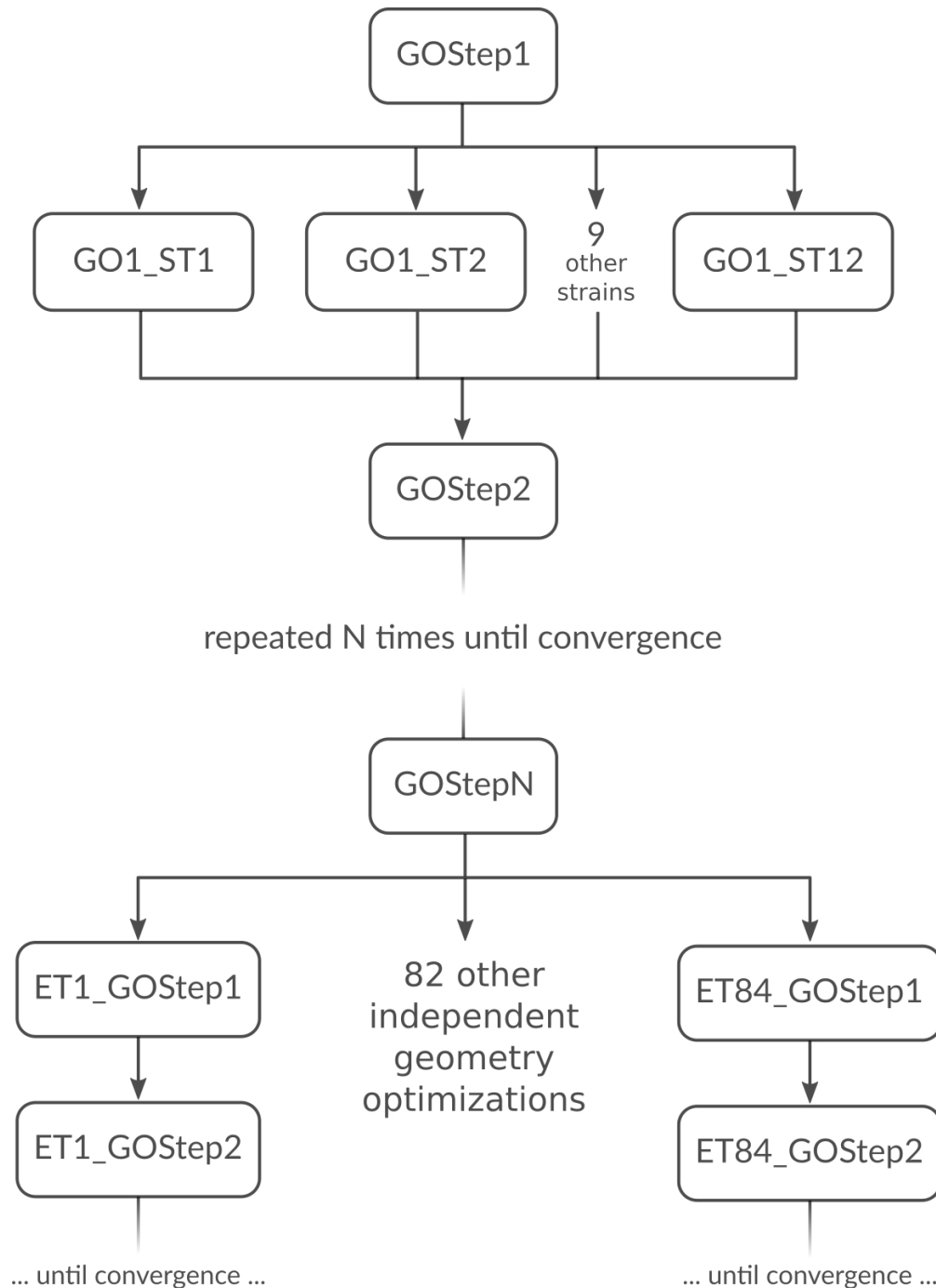
However, sometimes it can be useful to configure the groups manually. Suppose, as an example, that we want to calculate the elastic properties of a bulk material on a 32 core machine. The calculation of the *elastic tensor* (page 148) should be done on a relaxed geometry, including relaxed lattice degrees of freedom. We therefore first perform a geometry optimization, before calculating the elastic tensor. In AMS this can easily be done with the following input:

```
Task GeometryOptimization

GeometryOptimization
  OptimizeLattice True
End

Properties
  ElasticTensor True
End
```

But what is the most optimal parallel setup for this calculation? First we recognize that performing a lattice optimization requires the calculation of the *stress tensor* (page 147) at every step of the optimization. Assuming that our bulk system does not have any symmetries AMS can exploit, the numerical calculation of the stress tensor (assuming the engine can not calculate it analytically) would require 12 independent strained calculations for every step in the geometry optimization. Once the geometry optimization is converged, we have to perform 84 independent geometry optimizations to determine the elements of the elastic tensor. In summary, the graph of dependencies between all these tasks looks like this:



How do we best parallelize this? For the main steps, e.g. `GOSTep1` there is no question: We have nothing to do in parallel and all 32 cores work on it together to finish it as quickly as possible. For the numerical calculation of the stress tensor we have 12 tasks that can be done in parallel by the 32 cores in our machine. Now 12 obviously does not divide 32 without a remainder, so there is no way to split into equally sized groups and do all 12 strains in parallel. The greatest common divisor of 12 and 32 is 4, so it's probably best to split into 4 groups of 8 cores each. This is done with `nGroups 4`. Each group would then do 3 of the 12 strained calculations sequentially, using the in-engine parallelization to speed up the individual calculations. Once the stress tensor is computed in this way all groups merge and all 32 cores work together on `GOSTep2`. This splitting and merging now continues until the geometry optimization is converged. For the elastic tensor we now have 84 tasks to perform in parallel, where each task is a completely separate geometry optimization (without optimizing the lattice) of a strained system. 84 tasks is

more than double the number of cores we have. In this case it is probably not too bad to just run as parallel as possible at the driver level and make 32 “groups” of just one core to throw the 84 tasks at. This is easily done by setting `nCoresPerGroup 1` in the `ElasticTensor` block. Putting everything together we should add the following to our input file in order to optimally utilize our machine for this example calculation:

```
NumericalDifferentiation
  Parallel
    nGroups 4
  End
End

ElasticTensor
  Parallel
    nCoresPerGroup 1
  End
End
```

2.5 Python interface

There is a complete Python interface to AMS, which allows users to set up and run arbitrary AMS jobs, and to conveniently analyze the calculation results directly from Python. In this way AMS jobs can be automatized and complex multi-stage workflows implemented.

The scripting framework is called **PLAMS** as in “Python Library for Automating Molecular Simulation”, which conveniently can also be read as “Python Layer for AMS”. It is documented in a separate manual:

- [PLAMS introduction](#)
- [Running AMS through PLAMS](#)

2.6 Pipe interface

AMS can interact with other programs using a *custom communication protocol* (page 22). This enables two independent processes to communicate over a pair of data pipes (FIFOs), exchanging data in a highly efficient manner. One of the processes is the “pipe master”, driving the calculation and sending the atoms, coordinates etc. to the other process to perform calculations. The other process is the “pipe worker”, receiving requests from the master, performing the requested calculations and returning the results such as energies and gradients.

For technical details on the AMSPipe protocol see:

2.6.1 AMSPipe protocol specification

The AMSPipe protocol is a remote procedure call protocol used for communication between two processes on the same computer. One of these processes is the “pipe master”, submitting geometries and requests for calculations and receiving the results. The other process is the “pipe worker”, which listens for calls from the master, executes the requested calculations and returns their results.

Communication takes place over a pair of pipes, a “call pipe” (master to worker) and a “reply pipe” (worker to master). Each pipe carries a sequence of messages, which are processed in order. Each message is an associative object (like a JSON object or a Python dict) containing a single item. The key of this item denotes the name of the message, while the value of this item is another object containing the payload.

Each message sent over the call pipe constitutes a method call. The name of the message is equal to the name of the method being called, while the payload object contains any arguments. For methods without arguments, the payload is an empty object.

Once the worker receives a call message, it will execute the specified method and potentially write a sequence of zero or more messages to the reply pipe, containing the results of the calculation. After the execution of the method is completed, the worker will write a “return” messages to the reply pipe, denoting the success or failure of the method call.

Any method whose name starts with “Set” doesn’t send a “return” message. Instead, errors encountered during the execution of a Set method are buffered by the worker to be returned the next time a “return” message is generated. While an error is buffered, any further method calls are ignored by the worker and are not executed. Once a non-Set call is seen by the worker, such a call will also be discarded without being executed, but a “return” message will be immediately generated with the buffered error data. The buffered error will then be cleared and normal processing of calls resumes.

As a special case, the “Exit” method may be called at any time and will be immediately executed by the worker, terminating the protocol session. The “Exit” method never returns a “return” message, discarding any potentially buffered error.

All values are always in atomic units: Hartree for energies, Bohr for distances, Hartree/Bohr for gradients etc.

Low-level message encoding

All messages are encoded using Universal Binary JSON. AMSPipe implementations MAY encode messages using the UBJSON optimized container format (with explicit length and/or strong typing). Implementations MUST support decoding UBJSON containers whether or not they’re written using optimized format.

UBJSON arrays in messages MUST NOT contain other arrays or objects. Multi-dimensional arrays MUST be flattened on encoding (and unflattened on decoding as necessary). Flattening SHOULD preserve the original order of elements in memory. All arrays MUST be accompanied by an additional integer array holding the original dimensions before flattening. The name of this auxiliary array is equal to the name of the main array with a `_dim_` suffix. The dimensions are written in Fortran (column-major) order, so that the first value in the `_dim_` array corresponds to the index that changes the fastest when iterating over consecutive elements of the flattened array.

Arrays also MUST NOT contain elements of incompatible types. The elements of an array MUST be either all integers, all real numbers, all Boolean, or all strings. UBJSON type “char” is equivalent to a UBJSON “string” of length 1 (elements of type “char” and “string” MAY thus be mixed in an array). An empty array is equivalent to an array that is not present at all.

When sending messages over stream transport mechanisms (such as UNIX pipes), each message MUST be prefixed with a 32-bit native endian integer length.

Return messages and error handling

The “return” message consists of the following fields:

- “status” (integer): Zero for success or one of the error codes listed below.
- “method” (string, optional): Name of the method in which the error occurred.
- “argument” (string, optional): Name of the argument in error.
- “message” (string, optional): Human-readable error message.

The “status” field can have one of the following values:

0: success No error, method executed successfully.

- 1: decode_error** Message could not be decoded correctly (invalid UBJSON encoding or a violation of one of the constraints above).
- 2: logic_error** Programmer error, such as methods called in an incorrect sequence or with an invalid worker state.
- 3: runtime_error** An error occurred during the execution of a method (outside of the AMSPipe protocol).
- 4: unknown_version** (Only returned from “Hello”.) Requested protocol version is not supported by the worker.
- 5: unknown_method** A method of the requested name is not supported by the worker.
- 6: unknown_argument** The argument in “argument” is not known to the worker and couldn’t be processed. If multiple arguments to given single call are unknown, “argument” will be set to the one at the lowest nesting depth (if an argument contains nested objects). If multiple fields at the same nesting depth are unknown, the first one in ASCII sort order will be returned.
- 7: invalid_argument** An argument doesn’t have the correct type or dimensions, or it has an invalid value.

Methods

Hello(version)

- “version” (integer):

Attempt to activate a given version of the AMSPipe protocol. Only version 1 is defined at the moment.

Other methods (with the exception of “Hello” and “Exit”) MUST NOT be called until a Hello has completed successfully. Pipe masters SHOULD attempt a Hello with the highest supported protocol version and iterate downwards if an unknown_version error is returned. Once a Hello has succeeded, it MUST NOT be called again during the lifetime of a pipe session.

Exit()

Terminate the worker and disconnect both pipes. This method never returns.

The worker MAY also discard any remembered calculations.

SetCoords(coords)

- “coords” (real(3,:)):

Replace the Cartesian coordinates in the current chemical system. The number of atoms must match.

SetLattice(vectors)

- “vectors” (real(:,,:)):

Replace the lattice matrix of the current chemical system. If “vectors” is absent or of dimensions (0,0), make the system non-periodic.

SetSystem(atomSymbols, coords, totalCharge)

- “atomSymbols” (string(:)):
- “coords” (real(3,:)):

- “totalCharge” (real):

Define a new chemical system.

Solve(request, keepResults, prevTitle)

- **“request” (object):**
 - “title” (string): Unique string key identifying this calculation.
 - “quiet” (bool): If true, the worker SHOULD keep any standard output from the calculation to a minimum.
 - “gradients” (bool): Calculate gradients on atoms.
 - “stressTensor” (bool): Calculate the stress tensor.
 - “elasticTensor” (bool):
 - “hessian” (bool):
 - “dipoleMoment” (bool):
 - “dipoleGradients” (bool):
- “keepResults” (bool, default false): Remember worker state for future restart.
- “prevTitle” (string, optional): Title of a previously stored calculation to restart from.

Run a single point calculation on the current chemical system and return a “results” object if successful. If the calculation fails with a runtime error, a “results” object MAY still be returned (possibly with just some of the requested properties).

All Boolean fields in “request” default to false if not present. All non-Boolean fields in “request” except for “title” are optional and their default values are worker-dependent. The master SHOULD NOT explicitly set any Boolean fields in “request” to False. The worker MAY raise an unknown argument error if an unknown Boolean is set to False. The worker SHOULD raise an unknown argument error as usual if an unknown Boolean is set to True or if an unknown non-Boolean is set. Workers SHOULD raise these errors before performing any time-consuming calculations so that the master can efficiently retry the call.

If “keepResults” is not specified or set to false, the worker will discard all data from the calculation after returning a “results” object. If “keepResults” is set to true, the worker will remember any internal state related to the calculation. This internal state can later be reused for restart by passing the “title” of the stored calculation as “prevTitle”. The pipe master SHOULD call DeleteResults to discard the stored state as soon as it is no longer needed.

A “results” object consists of the following fields. Workers MAY include additional fields not listed here. A master MUST NOT signal an error due to any fields it does not expect or understand.

- **“results” (object):**
 - “messages” (string(:)): Runtime error or warning messages generated by the calculation.
 - “energy” (real):
 - “gradients” (real(3,:)):
 - “stressTensor” (real(:,,:)):
 - “elasticTensor” (real(:,,:)):
 - “hessian” (real(:,,:)):
 - “dipoleMoment” (real(:,,:)):
 - “dipoleGradients” (real(:,,:)):

- “charges” (real(:)):

DeleteResults(title)

- “title” (string): Title of a previously remembered calculation.

Discard any worker state corresponding to a previously remembered calculation.

Forward and backward compatibility considerations

The AMSPipe protocol is designed to support combining new masters with old workers and vice versa. Major, incompatible changes in the protocol will be handled by increasing the protocol version number. Negotiating a suitable protocol version is then handled by call(s) to Hello at the beginning of a pipe session.

The following requirements are in place to ensure that the protocol stays extensible within a single protocol version:

- Workers **MUST** raise an `unknown_method` error on any call to a method they don’t implement. If the method name starts with `Set`, such an error will be buffered to be returned later. If the method name doesn’t start with `Set`, a “return” message will be generated immediately.
- Workers **MUST** raise `unknown_argument` errors any time they encounter an argument to a known method that they don’t know how to handle. The master **SHOULD** then retry the call without such an argument or choose an alternative sequence of calls if possible.
- The master **SHOULD** ignore unexpected messages of unsupported type on the reply pipe.

Namely, the following changes are permitted without increasing the protocol version number:

- Adding new methods to the protocol.
- Adding new optional arguments to existing methods.
- Adding new fields to returned objects.
- Adding new reply message types.

2.6.2 AMS as a pipe master

The AMS driver can play the role of a pipe master, allowing users to combine the features of the AMS driver with potentials implemented in external programs. Unlike a *traditional external engine* (page 213), the overhead introduced by the pipe interface is entirely negligible, because the external program is only started once at the beginning of the run and all communication is handled by an efficient binary protocol instead of text files. This mode is enabled by using `Engine Pipe` in the input for the master.

```
Engine Pipe
  WorkerCommand /path/to/pipe/worker
EndEngine
```

WorkerCommand

Type String

Description The command to execute to run the external worker. The command is executed in a subdirectory of the results directory.

All calculations requested by the driver will then be forwarded over the pipe to the worker for processing.

Note: AMS currently must be run in serial (NSCM=1) when serving as a pipe master.

A Python module implementing the worker side of the AMSPipe protocol is available in `scm.ampipe`. To facilitate interfacing with various existing computational engines, this module provides the `ASEPipeWorker` class. This class can wrap any ASE calculator object and make it serve as a pipe worker.

```
calculator = ase.calculators.lj.LennardJones()
# calculator.parameters = ...

engine = scm.ampipe.ASEPipeWorker(calculator=calculator)
engine.run()
```

See also:

Example: ASE calculator as a pipe worker (page 309)

2.6.3 AMS as a pipe worker

AMS can also serve as a pipe worker, allowing external drivers to take advantage of its *engines* (page 209). This mode is enabled by using `Task Pipe`. No *Geometry*, *System definition* (page 29) is required on the input because the system will be supplied by the pipe master.

Hint: For most users it will easiest to use this functionality through the new `AMSWorker` class in the `PLAMS` library. This class hides all the details of the underlying AMSPipe protocol and provides users an easy way to get very fast access to energies, gradients and other properties from any of the engines in the AMS driver. See the [respective page](#) in the PLAMS manual for details.

See also:

Example: AMS as a pipe worker (page 310)

Additionally AMS can also be driven through the `FlexMD` library using `AMSPipeForceJob`:

```
forcejob = AMSPipeForceJob(mdmol)
forcejob.settings.engine = 'ReaxFF'
forcejob.settings.engineSettings = { 'ForceField': 'Glycine.ff' }
```


GEOMETRY, SYSTEM DEFINITION

The definition of the system to simulate, i.e. the positions and types of the nuclei, the total charge, and potentially lattice vectors, is enclosed in the `System` block:

```
System header
  Atoms header # Non-standard block. See details.
  ...
End
  Lattice header # Non-standard block. See details.
  ...
End
  FractionalCoords Yes/No
  AllowCloseAtoms Yes/No
  GeometryFile string
  Symmetrize Yes/No
  LatticeStrain float_list
  SuperCell integer_list
  Charge float
  BondOrders # Non-standard block. See details.
  ...
End
End
```

3.1 Geometry, Lattice

The geometry of the system is specified with the `Atoms` and `Lattice` blocks.

System

Type Block

Recurring True

Description Specification of the chemical system. For some applications more than one system may be present in the input. In this case, all systems except one must have a non-empty string ID specified after the `System` keyword. The system without an ID is considered the main one.

Atoms

Type Non-standard block

Description The atom types and coordinates. Unit can be specified in the header. Default unit is Angstrom.

Lattice

Type Non-standard block

Description Up to three lattice vectors. Unit can be specified in the header. Default unit is Angstrom.

FractionalCoords

Type Bool

Default value No

Description Whether the atomic coordinates in the Atoms block are given in fractional coordinates of the lattice vectors. Requires the presence of the Lattice block.

The `Atoms` block contains one line per atoms, similar to the lines found in an `.xyz` file: First the name of the element, then three real numbers representing the coordinates of that atom in Angstrom. The following `Atoms` block shows how one would define a water molecule:

```
System
  Atoms
    O  0.0  0.0  0.59372
    H  0.0  0.76544 -0.00836
    H  0.0 -0.76544 -0.00836
  End
End
```

Note that it is possible to specify a different unit of length in the header of the block (that is in the line after the keyword opening the block) by putting the name of the unit in [and] brackets. So the same water molecule could also be specified as follows:

```
System
  Atoms [Bohr]
    O  0.0  0.0  1.12197
    H  0.0  1.44647 -0.01580
    H  0.0 -1.44647 -0.01580
  End
End
```

It is also possible to specify the input geometry as a Z-Matrix, by putting the string `Z-Matrix` in the header of the block:

```
System
  Atoms Z-Matrix
    C
    H  1 1.089000
    H  1 1.089000 2 109.4710
    H  1 1.089000 2 109.4710 3 120.0000
    H  1 1.089000 2 109.4710 3 -120.0000
  End
End
```

Periodic systems require the specification of 1 (for chains), 2 (for slabs) or 3 (for bulk) lattice vectors in addition to the nuclear coordinates. Every lattice vector is specified on a separate line of three numbers, representing the vectors x,y and z-component. Note that for chain systems, the single lattice vector **must** point along the x-axis, while for slab systems the two lattice vectors **must** be in the xy-plane. Consider the following input for graphene:

```
System
  Atoms
    C  0.0  0.0  0.0
```

(continues on next page)

(continued from previous page)

```

    C  1.23  0.71014  0.0
  End
  Lattice
    2.46  0.0      0.0
    1.23  2.13042  0.0
  End
End

```

As with the `Atoms` block, the length unit in which the lattice vectors are given can be changed by specifying the desired unit in the header of the block (enclosed in `[]`). It is also possible to define a system given the fractional coordinates of the atoms using the `FractionalCoordinates` keyword. The numbers in the `Atoms` block are then interpreted as fractional coordinates according to the lattice vectors in the `Lattice` block. Note that for chain and slab systems, the coordinates perpendicular to the periodic direction (z and y for chains, z for slabs) are of course still in Angstrom (or alternatively the unit set in the header of the `Atoms` block). Using the `FractionalCoordinates` keyword we could specify the geometry of table salt (NaCl) as follows:

```

System
  Lattice
    0.0  2.75  2.75
    2.75  0.0  2.75
    2.75  2.75  0.0
  End
  FractionalCoordinates True
  Atoms
    Na  0.0  0.0  0.0
    Cl  0.5  0.5  0.5
  End
End

```

Instead of specifying the geometry of the system directly in the input file it can also be read from an external file.

System

GeometryFile

Type String

Description Read the geometry from a file (instead of from `Atoms` and `Lattice` blocks). Supported formats: `.xyz`

Note that the `GeometryFile` key replaces both the `Atoms` and the `Lattice` blocks in the input. So if you specify the `GeometryFile` keyword in the input, the `Atoms` and `Lattice` blocks must not appear there. At the moment only the *extended XYZ file format* (page 313) is supported.

3.1.1 Modifying the geometry

Finally there are a number of keywords that *modify* the system geometry:

System

Symmetrize

Type Bool

Default value No

Description Whether to symmetrize the input structure. This might also rototranslate the structure into a standard orientation. This will symmetrize the atomic coordinates to machine

precision. Useful if the system is almost symmetric or to rototranslate a symmetric molecule into a standard orientation.

LatticeStrain

Type Float List

Description Deform the input system by the specified strain. The strain elements are in Voigt notation, so one should specify 6 numbers for 3D periodic system (order: xx,yy,zz,yz,xz,xy), 3 numbers for 2D periodic systems (order: xx,yy,xy) or 1 number for 1D periodic systems.

SuperCell

Type Integer List

Description Create a supercell of the input system (only possible for periodic systems). The integer numbers represent the diagonal elements of the supercell transformation; you should specify as many numbers as lattice vectors (i.e. 1 number for 1D, 2 numbers for 2D and 3 numbers for 3D periodic systems).

SuperCellTrafo

Type Integer List

Description Create a supercell of the input system (only possible for periodic systems) $\vec{a}'_i = \sum_j T_{ij} \vec{a}_j$. The integer numbers represent the supercell transformation T_{ij} : 1 number for 1D PBC, 4 numbers for 2D PBC corresponding to a 2x2 matrix (order: (1,1),(1,2),(2,1),(2,2)) and 9 numbers for 3D PBC corresponding to a 3x3 matrix (order: (1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)).

PerturbCoordinates

Type Float

Default value 0.0

Unit Angstrom

Description Perturb the atomic coordinates by adding random numbers between [-PerturbCoordinates,PerturbCoordinates] to each Cartesian component. This can be useful if you want to break the symmetry of your system (e.g. for a geometry optimization).

PerturbLattice

Type Float

Default value 0.0

Description Perturb the lattice vectors by applying random strain with matrix elements between [-PerturbLattice,PerturbLattice]. This can be useful if you want to deviate from an ideal symmetric geometry, for example if you look for a phase change due to high pressure.

MapAtomsToUnitCell

Type Bool

Default value No

Description For periodic systems the atoms will be moved to the central cell.

These modifications are applied immediately after the system block is read. To the rest of AMS (and the input) it looks exactly as if the modified system was specified explicitly in the `System` block input. That means that the `SuperCell` keyword is not easily usable with input options that require the specification of atom indices, e.g. the `constraints` (page 46) block. Note that the randomization of the coordinates is applied after a potential supercell creation.

3.2 Symmetry

Symmetry can be used in AMS optimizations of molecules and of periodic structures.

In case of molecules at the start of an AMS calculation one can symmetrize an almost symmetric structure. In the *Appendix Symmetry* (page 317) one can find molecular orientation requirements that AMS needs such that AMS can use the (sub)symmetry of a molecule. If the system is symmetrized (and no symmetry is given in the System block key) the molecular structure is rotranslated into this standard orientation. In the *Appendix Symmetry* (page 315) one can also find Schönflies symbols for molecular point groups and symmetry labels that are used in AMS for molecules to label normal modes.

```
System header
  Symmetrize Yes/No
  Symmetry [...]
End
```

```
Symmetry
  SymmetrizeTolerance float
  Tolerance float
End
```

```
UseSymmetry Yes/No
```

System

Symmetrize

Type Bool

Default value No

Description Whether to symmetrize the input structure. This might also rototranslate the structure into a standard orientation. This will symmetrize the atomic coordinates to machine precision. Useful if the system is almost symmetric or to rototranslate a symmetric molecule into a standard orientation.

Symmetry

Type Multiple Choice

Default value AUTO

Options [AUTO, NOSYM, C(LIN), D(LIN), C(I), C(S), C(2), C(3), C(4), C(5), C(6), C(7), C(8), C(2V), C(3V), C(4V), C(5V), C(6V), C(7V), C(8V), C(2H), C(3H), C(4H), C(5H), C(6H), C(7H), C(8H), D(2), D(3), D(4), D(5), D(6), D(7), D(8), D(2D), D(3D), D(4D), D(5D), D(6D), D(7D), D(8D), D(2H), D(3H), D(4H), D(5H), D(6H), D(7H), D(8H), I, I(H), O, O(H), T, T(D), T(H), S(4), S(6), S(8)]

Description Use (sub)symmetry with this Schoenflies symbol. Can only be used for molecules. Orientation should be correct for the (sub)symmetry. If used icw Symmetrize, the symmetrization will not reorient the molecule.

Symmetry

SymmetrizeTolerance

Type Float

Default value 0.05

Description Tolerance used to detect symmetry in case symmetrize is requested.

Tolerance**Type** Float**Default value** 1e-07**Description** Tolerance used to detect symmetry in the system.**UseSymmetry****Type** Bool**Default value** Yes**Description** Whether to use the system's symmetry in AMS. Symmetry is recognized within a tolerance as given in the Symmetry key.

3.3 Regions

Some options of the AMS driver and its engines require specifying a subset of a system's atoms. For example one might want to freeze part of the system in a molecular dynamics calculation. We refer to these subsets of atoms as "regions". Atoms are assigned to regions by specifying the region names behind the atomic coordinates in the `Atoms` block of the input:

```
System
  Atoms
    C      1.22110608      -0.00000000      1.65928963      region=ring_1
    C      0.37734253      -1.16134090      1.65928963      region=ring_1
    C      0.37734253      1.16134090      1.65928963      region=ring_1
    C      -0.98789557      -0.71774815      1.65928963      region=ring_1
    C      -0.98789557      0.71774815      1.65928963      region=ring_1
    H      2.30849293      -0.00000000      1.64874914      region=ring_1
    H      0.71336355      -2.19550724      1.64874914      region=ring_1
    H      0.71336355      2.19550724      1.64874914      region=ring_1
    H      -1.86761001      -1.35689810      1.64874914      region=ring_1
    H      -1.86761001      1.35689810      1.64874914      region=ring_1
    Fe     0.00000000      0.00000000      0.00000000
    H      0.71336355      2.19550724      -1.64874914      region=ring_2
    H      0.71336355      -2.19550724      -1.64874914      region=ring_2
    C      1.22110608      -0.00000000      -1.65928963      region=ring_2
    C      0.37734253      1.16134090      -1.65928963      region=ring_2
    C      0.37734253      -1.16134090      -1.65928963      region=ring_2
    C      -0.98789557      0.71774815      -1.65928963      region=ring_2
    C      -0.98789557      -0.71774815      -1.65928963      region=ring_2
    H      2.30849293      -0.00000000      -1.64874914      region=ring_2
    H      -1.86761001      1.35689810      -1.64874914      region=ring_2
    H      -1.86761001      -1.35689810      -1.64874914      region=ring_2
  End
End
```

In the above example of a ferrocene molecule, we have created two separate regions for the two cyclopentadienyl rings. The region names (`ring_1` and `ring_2` in the example above) can be freely chosen by the user. Note that an atom can be in more than one region, in which case the region names should be separated by commas:

```
System
  Atoms
    ...
    Mg  0.0  0.0  0.0  region=metal_centers,mol_1
```

(continues on next page)

(continued from previous page)

```

    ...
End
End

```

The region names given by the user can then be referred to for input keywords requiring the specification of a region. Technically the region is handled as an *atomic attribute* (page 38).

3.4 Charge, atomic masses, input bond orders

AMS allows to set user-defined masses for particular atoms. This can be used to simulate isotopes of different atoms. Masses are specified by adding the desired mass (in Dalton) at the end of the atom's line. The following input shows the system specification for a heavy water molecule:

```

System
  Atoms
    O  0.0  0.0      0.59372
    H  0.0  0.76544 -0.00836  mass=2.014
    H  0.0 -0.76544 -0.00836  mass=2.014
  End
End

```

(Observe that the mass specified this way is an example of the *atomic attributes* (page 38) system.)

Finally the `System` block also contains the specification of the system's total charge as well as optionally defined bond orders, which might be needed by engines implementing force fields.

System

Charge

Type Float

Default value 0.0

GUI name Total charge

Description The system's total charge in atomic units.

BondOrders

Type Non-standard block

Description Defined bond orders. May be used by MM engines.

Note that the specified bond orders are currently only used by the ForceField engine.

3.5 Homogeneous electric field and multipole charges

A homogeneous electric field and multipole charges can be requested at the AMS level. Currently this option is supported by the engines ADF, BAND, DFTB, and MOPAC.

Homogeneous electric field:

```

System
  ElectrostaticEmbedding
    ElectricField ex ey ez
  End
End

```

ElectrostaticEmbedding**ElectricField****Type** Float List**Unit** V/Angstrom

Description External homogeneous electric field with three Cartesian components: ex , ey , ez , the default unit being V/. In atomic units: Hartree/(e bohr) = 27.211 V/bohr; the relation to SI units is: 1 Hartree/(e bohr) = 5.14 ... e11 V/m. Supported by the engines adf, band, dftb and mopac. For periodic systems the field may only have nonzero components orthogonal to the direction(s) of periodicity (i.e. for 1D periodic system the x-component of the electric field should be zero, while for 2D periodic systems both the x and y components should be zero. This options cannot be used for 3D periodic systems.

Point and multipole charges:

```

System
  ElectrostaticEmbedding
    MultipolePotential
      Coordinates
        x y z q py pz px
        x y z q py pz px
        ...
      End
    End
  End
End

```

ElectrostaticEmbedding**MultipolePotential****Type** Block**Description** External point charges (and dipoles).**ChargeModel****Type** Multiple Choice**Default value** Point**Options** [Point, Gaussian]

Description The charges may represented as simple points (with a singular potential at the charge location) or may represented by a spherical Gaussian distribution.

ChargeWidth**Type** Float**Default value** -1.0

Description A width parameter in a.u. in case a Gaussian charge model is chosen. A negative value means that the width will be chosen automatically.

Coordinates**Type** Non-standard block**Description** Positions (in Å) and values of multipole charges, one per line. Each line describes a single point charge like: $x\ y\ z\ q$, or $x\ y\ z\ q\ p_y\ p_z\ p_x$. Here x, y, z are the coordinates, q is the charge (in atomic units of charge) and p_y, p_z, p_x are the (optional) dipole components (in atomic units, i.e. e/Bohr). Periodic systems are not supported.

Note: When running geometry optimizations (or other similar tasks) in combination with point charges, you should be aware that the system might end up on top of the point charge(s), resulting in non-physical situations. You should consider using *constraints* (page 46), *restraints* (page 223) or explicitly setting *rigid motions* (page 399) options.

3.6 Load a System from file

Instead of specifying the system to simulate in the `System` block of the input, it is also possible to read the system used in a previous calculation from the binary `.rkf` result files of AMS. This is done with the `LoadSystem` block in the input:

```
LoadSystem header
  File string
  Section string
End
```

LoadSystem**Type** Block**Recurring** True**Description** Block that controls reading the chemical system from a KF file instead of the [System] block.**File****Type** String**Description** The path of the KF file from which to load the system. It may also be the results directory containing it.**Section****Type** String**Default value** Molecule**Description** The section on the KF file from which to load the system.

Note that the `LoadSystem` block is mutually exclusive with the `System` block: The system *either* needs to be specified in the input, *or* loaded from a previous results file.

Any `.rkf` file written by AMS should be suitable to load a system from. For *engine output files* (page 17) the loaded geometry is just the one for which the engine was invoked when it wrote this file. For the *main result file* (page 16) `ams.rkf` written by the AMS driver, which geometry is loaded depends on the *task* (page 41) that AMS was performing when this file was written. Generally the `ams.rkf` file contains two systems:

- The input system corresponding just to the `System` block that was read in by AMS. This system is written to the `InputMolecule` section on the `ams.rkf`, and can be loaded from there using the `LoadSystem%Section`

keyword. This can be useful in order to repeat a previous AMS calculation for the same system, but with different settings, e.g. a different engine.

- The system which was the result of a previous AMS calculation, e.g. a geometry optimization or transition state search. This system is written to the `Molecule` section on the `ams.rkf`. What exactly is considered the resulting geometry of a calculation depends in the *task* (page 41) of the previous calculation. (For tasks that do not change the geometry (like a single point calculation) or where no configuration is particularly special (e.g. a PES scan), the result system is normally just the same as the input system.)

3.7 Atom attributes

There is in general the possibility to add text after the coordinates of an atom, we call these atomic attributes. The text should be of the form `key=value`, separated by spaces:

```
System
  Atoms
    O  0.0  0.0      0.59372  key1=1 key2=y
    H  0.0  0.76544 -0.00836  file=ams.rkf
    H  0.0 -0.76544 -0.00836
  End
End
```

You can enter any arbitrary key/value list, but that will have no effect in general. Although there is one important effect that atoms with different atomic attributes (even unrecognized ones) are considered different elements during the symmetry detection.

The order in which keys are specified has no effect.

Examples of atomic attributes are the region, and mass.

3.8 Force field related extensions

For forcefield there are two dedicated *atomic attributes* (page 38), the `ForceField.Type` and `ForceField.Charge`. Obviously these attributes are ignored by all other engines, although it can reduce the symmetry of the system:

```
System
  Atoms
    O  0.0  0.0      0.59372  ForceField.Type=O ForceField.Charge=-0.1
    H  0.0  0.76544 -0.00836  ForceField.Type=H ForceField.Charge=+0.05
    H  0.0 -0.76544 -0.00836  ForceField.Type=H ForceField.Charge=+0.05
  End
End
```

3.8.1 Load charges for a forcefield into regions

When you want to use a forcefield, the charges obtained by a previous calculation can be loaded into a specific region (with the same molecule). Currently the order of the atoms has to be the same for this to work.

Once loaded, it is as if the user had typed in the `ForceField.Charge` attributes.

System

LoadForceFieldCharges

Type Block

Recurring True

Description This is a mechanism to set the ForceField.Charge attribute in the input. This information is currently only used by the ForceField engine.

CheckGeometryRMSD

Type Bool

Default value No

Description Whether the geometry RMSD test should be performed, see MaxGeometryRMSD. Otherwise only basic tests are performed, such as number and atom types. Not doing the RMSD test allows you to load molecular charges in a periodic system.

File

Type String

Description Name of the (kf) file

MaxGeometryRMSD

Type Float

Default value 0.1

Unit Angstrom

Description The geometry of the charge producing calculation is compared to the one of the region, and need to be the same within this tolerance.

Region

Type String

Default value *

Description Region for which the charges should be loaded

Section

Type String

Default value AMSResults

Description Section name of the kf file

Variable

Type String

Default value Charges

Description Variable name of the kf file

3.8.2 Load forcefield atom types

After you have done a forcefield calculation the atom types are stored. You can load those for a next calculation. Once loaded, it is as if the user had typed in the ForceField.Type atom attributes.

System

LoadForceFieldAtomTypes

Type Block

Description This is a mechanism to set the ForceField.Type attribute in the input. This information is currently only used by the ForceField engine.

File

Type String

Description Name of the (kf) file. It needs to be the result of a forcefield calculation.

STRUCTURE AND REACTIVITY, MOLECULAR DYNAMICS

4.1 Single point calculations

A single point calculation is the simplest task available in the AMS driver. It simply runs the *engine* (page 209) once for the given geometry. In other words, the AMS driver does not explore the potential energy surface (PES), but simply samples a “single point” of it.

A single point calculation is performed by selecting it with the Task keyword:

```
Task SinglePoint
```

Note that a single point calculation in AMS includes the calculation of *PES point properties* (page 141). Many of these, such as the nuclear gradients and the Hessian, are derivatives at this PES point with respect to nuclear displacements. These derivatives might be done numerically by the AMS driver, in which case it would technically run the engine multiple times and sample PES points around the initial point. However, in AMS this is still considered a single point calculation. Take for example the calculation of the normal modes of vibration of a molecule. This used to be a separate task in the 2017 release of the DFTB program, but in AMS is just a single point calculation with a request for normal modes:

```
Task SinglePoint
Properties
  NormalModes True
End
```

See the manual section on *PES point properties* (page 141) for an overview of which properties can be calculated with the SinglePoint task in AMS.

4.2 Bond energy calculations

The way to calculate bonding energies is always the same, regardless of the engine. It is about combining ground state energies for several systems.

4.2.1 Ground state energy

Let system A have a ground state energy $E(A)$

The ground state energy of a system is obtained by a full relaxation with respect to the geometry coordinates and (if relevant) to the electronic degrees of freedom. In case of periodic materials the lattice vectors need relaxation as well.

Electronic degrees of freedom are specific for the method underpinning the engine. A simple force field may have a charge equilibration scheme, whereas more advanced engines such as DFT- and DFTB-based ones have orbitals. The electronic relaxation can become a non-trivial problem in case of *open shell* (page 43) systems.

4.2.2 Formation energy

Say we have a reaction of two molecules (A and B) forming a new one (C).



The interaction energy follows from three *ground state* (page 41) calculations

$$E(\text{bond}) = E(C) - E(A) - E(B)$$

Let us look at a slightly more complicated reaction. The (metastable) material Octanitrocubane with the formula $C_8(NO_2)_8$ can be formed from 8 CO_2 molecules and four N_2 molecules. The formation energy is

$$E(\text{bond}) = E(\text{Octanitrocubane}) - 8 E(CO_2) - 4 E(N_2)$$

the result being positive as this reaction is highly endothermic.

4.2.3 Atomization energies

Let us now look at atomization energies. For instance, the cohesive energy of the NaCl crystal is

$$E(\text{cohesive energy}) = E(\text{NaCl-crystal}) - E(\text{Na-atom}) - E(\text{Cl-atom})$$

(Atomization energies can also be calculated for molecules.)

For engines with electronic degrees of freedom, the tricky part here is how to calculate the atomic energies, in particular $E(\text{Cl-atom})$, because they are open shell systems. See the notes on *atomic corrections* (page 42).

Of course you need to weigh the atomic energies by how often they occur in the system (molecule or crystal).

4.2.4 Chemisorption energies

What is the adsorption energy of a molecule on a surface?

The main thing here is whether the “real” system has translational symmetry. For instance in an experiment this may be the case and correspond to a certain coverage. This can be perfectly modeled in a periodic calculation

$$E(\text{chemisorption}) = E(\text{mol@surface}) - E(\text{mol}) - E(\text{bare surface})$$

You need to choose the right super cell to get the correct coverage (how many adsorbed atoms per unit cell).

If the experiment is about the adsorption of a single molecule you need to try to converge the result with progressively larger super cells.

The other main issue is that almost always in the experimental setup the surface is macroscopically thick. Therefore the slab thickness is an issue to be tested. Fortunately, often less than 10 layers are needed.

4.2.5 Atomic corrections

Sometimes we need to calculate the ground state energy of a single atom. As there are no geometric variables, the only degrees of freedom are electronic. In general atoms are *open shell* (page 43) systems. The idea is to let the engine run without any restrictions imposed on the spin-polarization, or the symmetry of the orbitals.

One possible trick is to run two atoms quite far away (10-20 angstrom), thus reducing the exact spherical symmetry.

Calculating atomic corrections can be very tricky, as there may be many nearly degenerate orbitals near the fermi level. Also SCF convergence can be very difficult. There is not a single solution for all atoms.

4.2.6 Open shell systems

The idea is to let the engine run without any restrictions imposed on the spin-polarization, or the symmetry of the orbitals.

Calculating open shell systems can be very tricky, as there may be many close lying states (of which the lowest is searched for). Also SCF convergence can be very difficult.

From a theoretical point of view these states should ultimately be described by multi-determinantal wave functions, lacking from methods such as DFT.

There may be no automatic way to find the absolute minimum. If possible try to avoid the need to do such calculations.

4.2.7 Impurities

Let's say that we want to introduce a single impurity in a crystal. Currently we do not support such a method. But using periodic boundary conditions (PBCs) we can approximate this.

Say we want to introduce a single Li atom in an Al crystal. This can be done by considering progressively larger super cells.

First we make a 2x2x2 super cell, with 8 Al atoms, then we insert one Li, thus modeling an "impurity", with a 1/8 Li/Al ratio, rather than a 1/infinity. With bigger super cells this approximation becomes better.

The insertion energy is then

$$E(8Al+Li) - E(8Al) - E(Li\text{-atom})$$

See also the comments for Atomization energies.

4.3 Geometry optimization

A geometry optimization is the process of changing the system's geometry (the nuclear coordinates and potentially the lattice vectors) to minimize the total energy of the systems. This is typically a local optimization, i.e. the optimization converges to the next local minimum on the potential energy surface (PES), given the initial system geometry specified in the `System` block. In other words: The geometry optimizer moves "downhill" on the PES into the local minimum.

See also:

Examples (page 241) and [diamond lattice optimization and phonons tutorial](#)

Geometry optimizations are performed by selecting them as the `Task`. The details of the optimization can be configured in the corresponding block:

```
Task GeometryOptimization
```

```
GeometryOptimization
  Convergence
    Energy float
    Gradients float
    Step float
    StressEnergyPerAtom float
  End
```

(continues on next page)

(continued from previous page)

```
MaxIterations integer
CalcPropertiesOnlyIfConverged Yes/No
OptimizeLattice Yes/No
KeepIntermediateResults Yes/No
PretendConverged Yes/No
End
```

GeometryOptimization**Type** Block**Description** Configures details of the geometry optimization and transition state searches.**Convergence****Type** Block**Description** Convergence is monitored for up to 4 quantities: the energy change, the Cartesian gradients, the Cartesian step size, and for lattice optimizations the stress energy per atom. Convergence criteria can be specified separately for each of these items.**Energy****Type** Float**Default value** 1e-05**Unit** Hartree**GUI name** Energy convergence**Description** The criterion for changes in the energy. The energy is considered converged when the change in energy is smaller than this threshold times the number of atoms.**Gradients****Type** Float**Default value** 0.001**Unit** Hartree/Angstrom**GUI name** Gradient convergence**Description** Threshold for nuclear gradients.**Step****Type** Float**Default value** 0.01**Unit** Angstrom**GUI name** Step convergence**Description** The maximum Cartesian step allowed for a converged geometry.**StressEnergyPerAtom****Type** Float**Default value** 0.0005**Unit** Hartree

Description Threshold used when optimizing the lattice vectors. The stress is considered ‘converged’ when the maximum value of $\text{stress_tensor} * \text{cell_volume} / \text{number_of_atoms}$ is smaller than this threshold (for 2D and 1D systems, the `cell_volume` is replaced by the `cell_area` and `cell_length` respectively).

A geometry optimization is considered converged when all the following criteria are met:

1. The difference between the bond energy at the current geometry and at the previous geometry step is smaller than `Convergence%Energy` times the number of atoms in the system.
2. The maximum Cartesian nuclear gradient is smaller than `Convergence%Gradient`.
3. The root mean square (RMS) of the Cartesian nuclear gradients is smaller than $2/3$ `Convergence%Gradient`.
4. The maximum Cartesian step is smaller than `Convergence%Step`.
5. The root mean square (RMS) of the Cartesian steps is smaller than $2/3$ `Convergence%Step`.

Note: If the maximum and RMS gradients are 10 times smaller than the convergence criterion, then criteria 4 and 5 are ignored.

Some remarks on the choice of the convergence thresholds:

- Molecules may differ very much in the stiffness around the energy minimum. Using the standard convergence thresholds without second thought is therefore not recommended. Strict criteria may require a large number of steps, while a loose threshold may yield geometries that are far from the minimum (with respect to atom-atom distances, bond-angles etc...) even when the total energy of the molecule might be very close to the value at the minimum. It is good practice to consider first what the objectives of the calculation are. The default settings in AMS are intended to be reasonable for most applications, but inevitably situations may arise where they are inadequate.
- The convergence threshold for the coordinates (`Convergence%Step`) is not a reliable measure for the precision of the final coordinates. Usually it yields a reasonable estimate (order of magnitude), but to get accurate results one should tighten the criterion on the gradients, rather than on the steps (coordinates). (The reason for this is that with the Quasi-Newton based optimizers the estimated uncertainty in the coordinates is related to the used Hessian, which is updated during the optimization. Quite often it stays rather far from an accurate representation of the true Hessian. This does usually not prevent the program from converging nicely, but it does imply a possibly incorrect calculation of the uncertainty in the coordinates.)
- Note that tight convergence criteria for the geometry optimization require accurate and noise-free gradients from the engine. For some engines this might mean that their numerical accuracy has to be increased for geometry optimization with tight convergence criteria, see e.g. the `NumericalQuality` keyword in the BAND manual.

The maximum number of geometry iterations allowed to locate the desired structure is specified with the `MaxIterations` keyword:

GeometryOptimization

MaxIterations

Type Integer

GUI name Maximum number of iterations

Description The maximum number of geometry iterations allowed to converge to the desired structure.

CalcPropertiesOnlyIfConverged

Type Bool

Default value Yes

Description Compute the properties requested in the ‘Properties’ block, e.g. Frequencies or Phonons, only if the optimization (or transition state search) converged. If False, the properties will be computed even if the optimization did not converge.

PretendConverged

Type Bool

Default value No

Description Normally a non-converged geometry optimization is considered an error. If this keyword is set to True, the optimizer will only produce a warning and still claim that the optimization is converged. (This is mostly useful for scripting applications, where one might want to consider non-converged optimizations still successful jobs.)

If the geometry optimization does not converge within this many steps it is considered failed and the iteration aborted, i.e. *PES point properties* (page 141) block will not be calculated at the last geometry. The default maximum number of steps is chosen automatically based on the used optimizer and the number of degrees of freedom to be optimized. The default is a fairly large number already, so if the geometry has not converged (at least to a reasonable extent) within that many iterations you should step back and consider the underlying cause rather than simply increase the allowed number of iterations and try again.

While a geometry optimization aims to find a (local) PES minimum, it may occur that it ends up finding a saddle point instead. The `PESPointCharacter` property keyword can be used to quickly calculate the lowest few Hessian eigenvalues to determine what kind of stationary PES point the optimization found. More information on this feature can be found on its *Documentation Page* (page 144).

For periodic systems the lattice degrees of freedom can be optimized in addition to the nuclear positions.

GeometryOptimization

OptimizeLattice

Type Bool

Default value No

Description Whether to also optimize the lattice for periodic structures. This is currently only supported with the Quasi-Newton, FIRE, L-BFGS and SCMGO optimizers.

Finally the `GeometryOptimization` block also contains some technical options:

GeometryOptimization

KeepIntermediateResults

Type Bool

Default value No

Description Whether the full engine result files of all intermediate steps are stored on disk. By default only the last step is kept, and only if the geometry optimization converged. This can easily lead to huge amounts of data being stored on disk, but it can sometimes be convenient to closely monitor a tricky optimization, e.g. excited state optimizations going through conical intersections, etc. ...

4.3.1 Constrained optimization

The AMS driver also allows to perform constrained optimizations, where a number of specified degrees of freedom are fixed to particular values.

See also:

Example demonstrating all supported constraints (page 249)

The desired constraints are specified in the `Constraints` block at the root level of the AMS input file:

```
Constraints
  Atom integer
  AtomList integer_list
  FixedRegion string
  Coordinate integer [x|y|z] float?
  Distance (integer){2} float
  Angle (integer){3} float
  Dihedral (integer){4} float
  SumDist (integer){4} float
  DifDist (integer){4} float
  BlockAtoms integer_list
  Block string
  FreezeStrain [xx] [xy] [xz] [yy] [yz] [zz]
  EqualStrain [xx] [xy] [xz] [yy] [yz] [zz]
End
```

Atom atomIdx Fix the atom with index `atomIdx` at the initial position, as given in the `System%Atoms` block.

AtomList [atomIdx1 .. atomIdxN] Fix all atoms in the list at the initial position, as given in the `System%Atoms` block.

FixedRegion regionName Fix all atoms in a *region* (page 34) to their initial positions.

Coordinate atomIdx [x|y|z] coordValue? Constrain the atom with index `atomIdx` (following the order in the `System%Atoms` block) to have a cartesian coordinate (x, y or z) of `coordValue` (given in Angstrom). If the `coordValue` is missing, the coordinate will be fixed to its initial value.

Distance atomIdx1 atomIdx2 distValue Constrain the distance between the atoms with index `atomIdx1` and `atomIdx2` (following the order in the `System%Atoms` block) to `distValue`, given in Angstrom.

Angle atomIdx1 atomIdx2 atomIdx3 angleValue Constrain the angle (1)–(2)–(3) between the atoms with indices `atomIdx1–3` (as given by their order in the `System%Atoms` block) to `angleValue`, given in degrees.

Dihedral atomIdx1 atomIdx2 atomIdx3 atomIdx4 dihedValue Constrain the dihedral angle (1)–(2)–(3)–(4) between the atoms with indices `atomIdx1–4` (as given by their order in the `System%Atoms` block) to `dihedValue`, given in degrees.

SumDist atomIdx1 atomIdx2 atomIdx3 atomIdx4 sumDistValue Constrain the sum of the distances $R(1,2)+R(3,4)$ between the atoms with indices `atomIdx1–4` (as given by their order in the `System%Atoms` block) to `sumDistValue`, given in Angstrom.

DifDist atomIdx1 atomIdx2 atomIdx3 atomIdx4 difDistValue Constrain the difference between the distances $R(1,2)-R(3,4)$ of the atoms with indices `atomIdx1–4` (as given by their order in the `System%Atoms` block) to `difDistValue`, given in Angstrom.

Note that the above constraints do **not** need to be satisfied at the beginning of the optimization.

BlockAtoms [atomIdx1 ... atomIdxN] Creates a block constraint (freezes all internal degrees of freedom) for a set of atoms identified by the list of integers `[atomIdx1 ... atomIdxN]`. These atom indices refer to the order of the atoms in the `System%Atoms` block.

Block regionName Creates a block constraint (freezes all internal degrees of freedom) for all atoms in a *region* (page 34) defined in the `System%Atoms` block. Example:

```

System
  Atoms
    C  0.0  0.0  0.0   region=myblock
    C  0.0  0.0  1.0   region=myblock
    C  0.0  1.0  0.0
  End
End
Constraints
  Block myblock
End

```

For lattice optimizations, the following constraints can be used on the lattice degrees of freedom:

FreezeStrain [**xx**] [**xy**] [**xz**] [**yy**] [**yz**] [**zz**] Exclusively for lattice optimizations: Freezes any lattice deformation corresponding to a particular component of the strain tensor. Accepts a set of strain components [xx, xy, xz, yy, yz, zz] to be frozen.

EqualStrain [**xx**] [**xy**] [**xz**] [**yy**] [**yz**] [**zz**] Exclusively for lattice optimizations: Accepts a set of strain components [xx, xy, xz, yy, yz, zz] which are to be kept equal. The applied strain will be determined by the average of the corresponding stress tensors components.

Note that in principle an arbitrary number of constraints can be specified and thus combined. However, it is the user's responsibility to ensure that the specified constraints are actually *compatible with each other*, meaning that it is theoretically possible to satisfy all of them at the same time. The AMS driver does **not** detect this kind of problems, but the optimization will show very unexpected results. Furthermore, for calculations involving block constraints the following restrictions apply:

- There should be no other constrained coordinates used together with block constraints although this may work in many situation.
- The user should absolutely avoid specifying other constraints that include atoms of a frozen block.

Restraints

Not all optimizers support constraints. An alternative is to use so-called restraints. These are not exact constraints, but rather a number of springs that pull the system towards the preferred constraints, see [Restraints](#) (page 223).

4.3.2 Optimization under pressure / external stress

Pressure (page 220) or *non-isotropic external stress* (page 221) can be included in your simulation via the corresponding *engine addons* (page 217).

4.3.3 Optimization methods

The AMS driver implements a few different geometry optimization algorithms. It also allows to choose the coordinate space in which the optimization is performed:

```

GeometryOptimization
  Method [Auto | Quasi-Newton | SCMGO | FIRE | L-BFGS | ConjugateGradients]
  CoordinateType [Auto | Delocalized | Cartesian]
End

```

GeometryOptimization

Method

Type Multiple Choice

Default value Auto

Options [Auto, Quasi-Newton, SCMGO, FIRE, L-BFGS, ConjugateGradients]

GUI name Optimization method

Description Select the optimization algorithm employed for the geometry relaxation. Currently supported are: the Hessian-based Quasi-Newton-type BFGS algorithm, the experimental SCMGO optimizer, the fast inertial relaxation method (FIRE), the limited-memory BFGS method, and the conjugate gradients method. The default is to choose an appropriate method automatically based on the engine's speed, the system size and the supported optimization options.

CoordinateType

Type Multiple Choice

Default value Auto

Options [Auto, Delocalized, Cartesian]

GUI name Optimization space

Description Select the type of coordinates in which to perform the optimization. 'Auto' automatically selects the most appropriate CoordinateType for a given Method. If 'Auto' is selected, Delocalized coordinates will be used for the Quasi-Newton and SCMGO methods, while Cartesian coordinates will be used for all other methods.

We **strongly** advise leaving both the Method as well as the Coordinate type on the Auto setting. There are many restrictions as to which optimizer and coordinate type can be used together with which kind of optimization. The following (roughly) sketches the compatibility of the different optimizers and options:

Optimizer	Constraints	Lattice opt.	Coordinate types
Quasi-Newton	All, except strain constraints	Yes	All
FIRE	Fixed coordinates, strain constraints	Yes	Cartesian
SCMGO	No	Yes	Delocalized
L-BFGS	No	Yes	Cartesian
Conjugate gradients	No	No	Cartesian

Furthermore for optimal performance the optimizer should be chosen with the speed of the engine: a faster engine in combination should use an optimizer with little overhead (FIRE), while slower engines should use optimizers that strictly minimize the number of steps (Quasi-Newton, SCMGO). This is all handled automatically by default, and we recommend changing Method and Coordinate only in case there are problems with the automatic choice.

The following subsections list the strengths and weaknesses of the individual optimizers in some more detail, motivating why which optimizer is chosen automatically under which circumstances.

Quasi-Newton

This optimizer implements a quasi Newton approach¹²³, using the Hessian for computing changes in the geometry so as to reach a local minimum. The Hessian itself is typically *approximated* (page 51) in the beginning and updated in the process of optimization. It uses delocalized coordinates by default both for molecules and periodic systems. The

¹ L. Versluis and T. Ziegler, *The determination of Molecular Structure by Density Functional Theory*, *Journal of Chemical Physics* 88, 322 (1988) (<https://doi.org/10.1063/1.454603>)

² L. Versluis, *The determination of molecular structures by the HFS method*, PhD thesis, University of Calgary, 1989

³ L. Fan and T. Ziegler, *Optimization of molecular structures by self consistent and non-local density functional theory*, *Journal of Chemical Physics* 95, 7401 (1991) (<https://doi.org/10.1063/1.461366>)

molecular part is based mainly on the work by Marcel Swart⁴. Cartesian coordinates are used in the presence of an external electric field and/or frozen atom constraints.

The Quasi-Newton (QN) optimizer supports all types of constraints and can be used for both molecular and periodic systems, including lattice optimizations. For cases where the optimization can be performed in delocalized coordinates, the number of steps taken to reach the local minimum is usually smaller than when optimizing in Cartesian ones. For fast compute *engines* (page 209), the overhead of the QN optimizer can become a bottleneck of the calculation, thus a more light-weight optimizer such as *FIRE* (page 52) may give an better overall performance. In principle, a QN optimization in delocalized coordinates may run out of memory for a very large system (say over 1000 atoms) because of the SVD step. However, since it is going to be used for a moderate-to-slow engine we still recommend sticking to it for the benefit of fewer steps. Because of these properties the QN optimizer is the default in AMS for all kinds of optimizations with moderate and slow engines, such DFTB and ADF. It is also used as the optimizer back-end for the *PES scan task* (page 60), the *transition state search* (page 57) as well as the calculation of the *elastic tensor* (page 148).

Details of the Quasi-Newton optimizer are configured in a dedicated block:

```
GeometryOptimization
  Quasi-Newton
    MaxGDIISVectors integer
    Step
      TrustRadius float
    End
    UpdateTSVectorEveryStep Yes/No
  End
End
```

GeometryOptimization

Quasi-Newton

Type Block

Description Configures details of the Quasi-Newton geometry optimizer.

MaxGDIISVectors

Type Integer

Default value 0

Description Sets the maximum number of GDIIS vectors. Setting this to a number >0 enables the GDIIS method.

Step

Type Block

Description

TrustRadius

Type Float

Description Initial value of the trust radius.

UpdateTSVectorEveryStep

Type Bool

Default value Yes

⁴ M. Swart and F.M. Bickelhaupt, *Optimization of strong and weak coordinates*, International Journal of Quantum Chemistry 106, 2536 (2006) (<https://doi.org/10.1002/qua.21049>)

GUI name Update TSRC vector every step

Description Whether to update the TS reaction coordinate at each step with the current eigenvector.

The Quasi-Newton optimizer uses the Hessian to compute the step of the geometry optimization. The Hessian is typically approximated in the beginning and then updated during the optimization. A very good initial Hessian can therefore increase the performance of the optimizer and lead to faster and more stable convergence. The choice of the initial Hessian can be configured in a dedicated block:

```
GeometryOptimization
  InitialHessian
    File string
    Type [Auto | UnitMatrix | Swart | FromFile | Calculate | ↵
↵CalculateWithFastEngine]
  End
End
```

GeometryOptimization

InitialHessian

Type Block

Description Options for initial model Hessian when optimizing systems with either the Quasi-Newton or the SCMGO method.

File

Type String

GUI name Initial Hessian from

Description KF file containing the initial Hessian (or the results dir. containing it). This can be used to load a Hessian calculated in a previously with the [Properties%Hessian] keyword.

Type

Type Multiple Choice

Default value Auto

Options [Auto, UnitMatrix, Swart, FromFile, Calculate, CalculateWithFastEngine]

GUI name Initial Hessian

Description Select the type of initial Hessian. Auto: let the program pick an initial model Hessian. UnitMatrix: simplest initial model Hessian, just a unit matrix in the optimization coordinates. Swart: model Hessian from M. Swart. FromFile: load the Hessian from the results of a previous calculation (see InitialHessian%File). Calculate: compute the initial Hessian (this may be computationally expensive and it is mostly recommended for TransitionStateSearch calculations). CalculateWithFastEngine: compute the initial Hessian with a faster engine.

While there are some options for the construction of approximate model Hessians, the best initial Hessians are often those calculated explicitly at a lower level of theory, e.g. the real DFTB Hessian can be used the initial Hessian for an optimization with the more accurate BAND engine. Using the CalculateWithFasterEngine keyword can be used to automatically chose a fast engine at a lower level of theory. What the lower level of theory is depends on the main engine used in the calculation: DFTB with the GFN1-xTB model is used as the lower level of theory for relatively slow engines, e.g. DFT based engines. For semi-empirical engines like DFTB or MOPAC, the lower level of theory is currently UFF. If more control over the lower level engine is needed, the initial Hessian can be calculated with a user defined engine and then loaded from file, see [this example](#) (page 241).

FIRE

The Fast Inertial Relaxation Engine⁵ based optimizer has basically no overhead per step, so that the speed of the optimization purely depends on the performance of the used compute *engine* (page 209). As such it is a good option for large systems or fast compute engines, where the overhead of the Quasi-Newton optimizer would be significant. Note that it also supports *fixed atom constraints* (page 47) and *coordinate constraints* (page 47) (as long as the value of the constrained coordinate is already satisfied in the input geometry), as well as lattice optimizations (with strain constraints).

FIRE is selected as the default optimizer for fast compute engines if it is compatible with all other settings of the optimization (i.e. no unsupported constraints or coordinate types).

Note: FIRE is a very robust optimizer. In case of convergence problems with the other methods, it is a good idea to see if the optimization converges with FIRE. If it does not, it is very likely that the problem is not the optimizer but the shape of the potential energy surface ...

The details of the FIRE optimizer are configured in a dedicated block. It is quite easy to make the optimization numerically unstable when tweaking these settings, so we strongly recommend leaving everything at the default values.

```

GeometryOptimization
  FIRE
    AllowOverallRotation Yes/No
    AllowOverallTranslation Yes/No
    MapAtomsToUnitCell Yes/No
    NMin integer
    alphaStart float
    dtMax float
    dtStart float
    fAlpha float
    fDec float
    fInc float
    strainMass float
  End
End

```

GeometryOptimization

FIRE

Type Block

Description This block configures the details of the FIRE optimizer. The keywords name correspond to the symbols used in the article describing the method, see PRL 97, 170201 (2006).

AllowOverallRotation

Type Bool

Default value Yes

Description Whether or not the system is allowed to freely rotate during the optimization. This is relevant when optimizing structures in the presence of external fields.

AllowOverallTranslation

Type Bool

Default value No

⁵ E. Bitzek, P. Koskinen, F. Gähler, M. Moseler and P. Gumbusch, *Structural Relaxation Made Simple*, Physical Review Letters 97, 170201 (2006) (<https://doi.org/10.1103/PhysRevLett.97.170201>)

Description Whether or not the system is allowed to translate during the optimization. This is relevant when optimizing structures in the presence of external fields.

MapAtomsToUnitCell

Type Bool

Default value No

Description Map the atoms to the central cell at each geometry step.

NMin

Type Integer

Default value 5

Description Number of steps after stopping before increasing the time step again.

alphaStart

Type Float

Default value 0.1

Description Steering coefficient.

dtMax

Type Float

Default value 1.0

Unit Femtoseconds

Description Maximum time step used for the integration.

dtStart

Type Float

Default value 0.25

Unit Femtoseconds

Description Initial time step for the integration.

fAlpha

Type Float

Default value 0.99

Description Reduction factor for the steering coefficient.

fDec

Type Float

Default value 0.5

Description Reduction factor for reducing the time step in case of uphill movement.

fInc

Type Float

Default value 1.1

Description Growth factor for the integration time step.

strainMass**Type** Float**Default value** 0.5**Description** Fictitious relative mass of the lattice degrees of freedom. This controls the stiffness of the lattice degrees of freedom relative to the atomic degrees of freedom, with smaller values resulting in a more aggressive optimization of the lattice.**SCMGO**

The SCMGO optimizer is a new implementation of a Quasi-Newton style optimizer working in delocalized coordinates. In the 2020 release of the Amsterdam Modeling Suite it is still considered experimental and therefore never selected automatically. However, for molecules and fully connected periodic systems it already shows a quite good performance, and could be a reasonable alternative to the classic *Quasi-Newton* (page 49) optimizer.

```

GeometryOptimization
  SCMGO
    ContractPrimitives Yes/No
    NumericalBMatrix Yes/No
    Step
      TrustRadius float
      VariableTrustRadius Yes/No
    End
    logSCMGO Yes/No
    testSCMGO Yes/No
  End
End

```

GeometryOptimization**SCMGO****Type** Block**Description** Configures details SCMGO.**ContractPrimitives****Type** Bool**Default value** Yes**Description** Form non-redundant linear combinations of primitive coordinates sharing the same central atom**NumericalBMatrix****Type** Bool**Default value** No**Description** Calculation of the B-matrix, i.e. Jacobian of internal coordinates in terms of numerical differentiations**Step****Type** Block**Description****TrustRadius**

Type Float

Default value 0.2

Description Initial value of the trust radius.

VariableTrustRadius

Type Bool

Default value Yes

Description Whether or not the trust radius can be updated during the optimization.

logSCMGO

Type Bool

Default value No

Description Verbose output of SCMGO internal data

testSCMGO

Type Bool

Default value No

Description Run SCMGO in test mode.

Note that SCMGO also supports different initial Hessians, and uses the same options for the initial Hessian as the Quasi-Newton optimizer, see [above](#) (page 51).

Limited-memory BFGS

AMS also offers an L-BFGS based geometry optimizer. It usually converges faster than *FIRE* (page 52), but does not support constrained optimizations. For periodic systems it can be quite good for lattice optimizations. The new implementation has not been thoroughly tested yet, therefore never selected automatically. For large systems and fast engines you may want to disable symmetry: simply the detection of (non-existing) symmetry may be a huge overhead.

```

GeometryOptimization
  HessianFree
    Step
      MaxCartesianStep float
      MinRadius float
      TrialStep float
      TrustRadius float
    End
  End
End

```

GeometryOptimization

HessianFree

Type Block

Description Configures details of the Hessian-free (conjugate gradients or L-BFGS) geometry optimizer.

Step

Type Block

Description

MaxCartesianStep**Type** Float**Default value** 0.1**Unit** Angstrom**Description** Limit on a single Cartesian component of the step.**MinRadius****Type** Float**Default value** 0.0**Unit** Angstrom**Description** Minimum value for the trust radius.**TrialStep****Type** Float**Default value** 0.0005**Unit** Angstrom**Description** Length of the finite-difference step when determining curvature. Should be smaller than the step convergence criterion.**TrustRadius****Type** Float**Default value** 0.2**Unit** Angstrom**Description** Initial value of the trust radius.

Conjugate gradients

AMS also offers a conjugate gradients based geometry optimizer, as it was also implemented in the pre-2018 releases of the DFTB program. However, it is usually slightly slower than *FIRE* (page 52), and supports neither lattice nor constrained optimizations. It is therefore never selected automatically, and we do not recommend using it. Like L-BFGS, the conjugate gradients optimizer is also configured in the `HessianFree` block, see L-BFGS section above for details.

4.3.4 Troubleshooting

Failure to converge

First of all one should look how the energy changed during the latest ten or so iterations. If the energy is decreasing more or less consistently, possibly with occasional jumps, then there is probably nothing wrong with the optimization. This behavior is typical in the cases when the starting geometry was far away from the minimum and the optimization has a long way to go. Just increase the allowed number of iterations, restart from the latest geometry and see if the optimization converges.

If the energy oscillates around some value and the energy gradient hardly changes then you may need to look at the calculation setup.

The success of geometry optimization depends on the accuracy of the calculated forces. The default accuracy settings are sufficient in most cases. There are, however, cases when one has to increase the accuracy in order to get geometry optimization converged. First of all, this may be necessary if you tighten the optimization convergence criteria. In some cases it may be necessary to increase the accuracy also for the default criteria. Please refer to the *engine manuals* (page 209) for instructions on how to increase the accuracy of an engine's energies and gradients. Often this is done with the `NumericalQuality` keyword in the engine input.

A geometry optimization can also fail to converge because the underlying potential energy surface is problematic, e.g. it might be discontinuous or not have a minimum at which the gradients vanish. This often indicates real problems in the calculation setup, e.g. an electronic structure that changes fundamentally between subsequent steps in the optimization. In these cases it is advisable to run a single point calculation at the problematic geometries and carefully check if the results are physically actually sensible.

Finally it can also be a technical problem with the specific *optimization method* (page 48) used. In these cases switching to another method could help with convergence problems. We recommend first trying the *FIRE* (page 52) optimizer, as it is internally relatively simple and stable.

Restarting a geometry optimization

During a running optimization the system's geometry is written out to the AMS driver's output file `ams.rkf` after every step (in the `Molecule` section). This means that crashed or otherwise canceled geometry optimizations can be restarted by simply loading the last frame from there using the `LoadSystem` keyword, see *its documentation* (page 35) in the system definition section of this manual:

```
LoadSystem File=my_crashed_GO.results/ams.rkf
```

This can of course also be used to continue an optimization but e.g. with tighter convergence criteria or a different optimizer, as it essentially starts a new geometry optimization from the previous geometry, and does not propagate any information internal to the optimizer (e.g. the approximate Hessian for the Quasi-Newton optimizer or the FIRE velocities) to the new job. As such it might take a few more steps to convergence than if the original job had continued, but allows for additional flexibility.

4.4 Transition state search

A transition state (TS) search is very much like a *geometry optimization* (page 43): the purpose is to find a stationary point on the potential energy surface, primarily by monitoring the energy gradients, which should vanish. The difference between a transition state and a minimum is that at the transition state the Hessian has a negative eigenvalue: We are at a saddle point, not a minimum, with the "negative" mode connecting the two basins on the potential energy surface.

See also:

- *Examples* (page 241)
- Tutorial Transition state search and characterization of a Ziegler Natta Catalyst
- Tutorial Tips and Tricks for Transition State Searches for Click Reactions

A transition state search in AMS is performed by selecting the corresponding task:

```
Task TransitionStateSearch
```

Due to the similarities between energy minimization and transition state search, the `TransitionStateSearch` task in AMS is actually implemented as a special kind of geometry optimization using the *quasi-Newton* (page 49) optimizer. As such all the settings and keywords described on the *geometry optimization manual page* (page 43) also apply to transition state searches.

In a geometry optimization with a quasi-Newton based optimizer the Hessian is used to make a reasonably sized step in the “downhill” direction on the potential energy surface, as the goal is simply to minimize the energy. A transition state search is a bit different: In the first step a normal mode is picked along which the energy is to be *maximized*, while it is *minimized* along all other directions. Normally the mode with the lowest eigenvalue is picked, since we know that there should be exactly one negative eigenvalue at the TS geometry, but one can also choose an approximate reaction coordinate, see `ReactionCoordinate`. If the initial geometry is sufficiently close to the transition state, i.e. we are close to the saddle, the lowest mode is normally the correct one to follow in order to get to the ridge of the saddle. Alternatively a different mode can also be selected manually.

```
TransitionStateSearch
  ModeToFollow integer
  ReactionCoordinate
    Angle string
    Dihedral string
    Distance string
  End
End
```

TransitionStateSearch

Type Block

Description Configures some details of the transition state search.

ModeToFollow

Type Integer

Default value 1

Description In case of Transition State Search, here you can specify the index of the normal mode to follow (1 is the mode with the lowest frequency).

ReactionCoordinate

Type Block

Description Specify components of the transition state reaction coordinate (TSRC) as a linear combination of internal coordinates (distances or angles).

Angle

Type String

Recurring True

Description The TSRC contains the valence angle between the given atoms. Three atom indices followed by the weight.

Dihedral

Type String

Recurring True

Description The TSRC contains the dihedral angle between the given atoms. Four atom indices followed by the weight.

Distance

Type String

Recurring True

Description The TSRC contains the distance between the given atoms. Two atom indices followed by the weight.

This selection happens only in the first step. Subsequent steps will attempt to maximize along the mode that resembles most (by overlap) the previous maximization direction.

Practice shows that transition states are much harder to find than a minimum. For a large part this is due to the much stronger anharmonicities that usually occur near the TS, which threaten to invalidate the quasi-Newton methods to find the stationary point. For this reason it is good advice to be more cautious in the optimization strategy when approaching a transition state:

- We recommend starting the transition state search with an initial geometry that is already close to the transition state. One can use a *potential energy surface scan* (page 60) along something resembling the reaction coordinate to get a rough idea where the transition state is. This geometry can then be used as an initial geometry for the transition state search.
- It is strongly recommended to manually supply a good initial Hessian for the transition state search. Otherwise the first step of the search might not be taken in the correct direction and subsequent steps will attempt to keep steering in the wrong direction. In AMS this is easily possible by loading a Hessian from a previous calculation, see the *initial Hessian section* (page 51) of this manual. A good way to obtain a reasonable Hessian is to compute it explicitly with one of the fast engines (i.e. at a lower level of theory) and read that Hessian as the initial Hessian for the transition state search at a higher level of theory.
- When no accurate initial Hessian is available, it may be a good idea to specify an approximate normal mode vector using the `ReactionCoordinate` input block. A reaction coordinate is a linear combination of distances and/or valence and dihedral angles:

```
TransitionStateSearch
  ReactionCoordinate
    Distance  i  j      fac
    Distance  i  j      fac
    Angle     i  j  k    fac
    Dihedral  i  j  k  l  fac
  End
End
```

Here, *i*, *j*, *k*, and *l* are atom indices, and the `fac` is the factor with which the internal coordinate enters the linear combination.

One should be careful when specifying more than one bond or angle as a transition state reaction coordinate (TSRC). For example, suppose atom 2 is located between atoms 1 and 3. Then the following `ReactionCoordinate` block:

```
TransitionStateSearch
  ReactionCoordinate
    Distance  1  2  1.0
    Distance  2  3 -1.0
  End
End
```

means that the TSRC consists of two distances: $R(1-2)$ and $R(2-3)$. The positive direction of the TSRC is defined as an increase of the $R(1-2)$ and a decrease of the $R(2-3)$. In other words, this TSRC corresponds to atom 2 moving along the $R(1-3)$ axis.

- When the method converges it is usually a good idea to verify that the found geometry is indeed a transition state. This can be done by performing a frequency analysis and checking whether the Hessian has exactly one negative eigenvalue (represented by a mode with a negative frequency). Doing so is expensive however. Since we are really only interested in the lowest two normal modes calculating the full Hessian is however not necessary and one can use the faster PES point characterization instead. This uses a Davidson-type algorithm to

obtain the lowest few normal modes without constructing the full Hessian. The user is referred to the *PES point characterization* (page 144) documentation page for further details.

4.5 Linear Transit, PES scan

The PES scan task in AMS allows users to scan the potential energy surface of a system along one or multiple degrees of freedom, while relaxing all other degrees of freedom. If only one coordinate is scanned, this kind of calculation is usually just called a linear transit. However, since AMS allows scanning of multiple coordinates, and linear transit is just a special case of such a calculation, the task is always called a PES scan in AMS.

A linear transit may be used for instance to sketch an approximate path over the transition states between reactants and products. From this a reasonable guess for the Transition State can be obtained which may serve as starting point for a true transition state search for instance.

See also:

- *Examples* (page 241)
- Tutorial *Transition state search and characterization of a Ziegler Natta Catalyst*

The PES scan task is enabled by selecting it with the `Task` keyword:

```
Task PESScan
```

The `PESScan` block configures all details of the scan:

```
PESScan
  CalcPropertiesAtPESPoints [True | False]
  FillUnconvergedGaps [True | False]
  ScanCoordinate
    nPoints integer
    Coordinate integer [x|y|z] (float){2}
    Distance (integer){2} (float){2}
    Angle (integer){3} (float){2}
    Dihedral (integer){4} (float){2}
    SumDist (integer){4} (float){2}
    DifDist (integer){4} (float){2}
  End
End
```

The `PESScan` block needs to contain at least one `ScanCoordinate` block specifying which coordinate to scan, and how many points (keyword `nPoints`) to sample along this coordinate. By default, 10 points are sampled along each scanned coordinate (including the start and end point of the scan). The coordinate descriptors are very similar to the *constraint descriptors* (page 46) in the `Constraints` block used by the geometry optimization task, but are followed by two values delimiting the start and end of the coordinates, instead of just a single value:

Coordinate atomIdx [x|y|z] startValue endValue Moves the atom with index `atomIdx` (following the order in the `System` block) along the a cartesian coordinate (x, y or z), starting at `startValue` and ending at `endValue` (given in Angstrom).

Distance atomIdx1 atomIdx2 startDist endDist Scans the distance between the atoms with index `atomIdx1` and `atomIdx2`, starting from `startDist` and ending at `endDist`, both given in Angstrom.

Angle atomIdx1 atomIdx2 atomIdx3 startAngle endAngle Scans the angle (1)–(2)–(3) between the atoms with indices `atomIdx1–3`, as given by their order in the `System%Atoms` block. The scanned angle starts at `startAngle` and ends at `endAngle`, given in degrees.

Dihedral *atomIdx1 atomIdx2 atomIdx3 atomIdx4 startAngle endAngle* Scans the dihedral angle (1)–(2)–(3)–(4) between the atoms with indices *atomIdx1–4*, as given by their order in the *System%Atoms* block. The scanned dihedral starts at *startAngle* and ends at *endAngle*, given in degrees.

SumDist *atomIdx1 atomIdx2 atomIdx3 atomIdx4 start end* Scans the sum of the distances $R(1,2)+R(3,4)$ between the atoms with indices *atomIdx1–4*, as given by their order in the *System%Atoms* block. The values to be scanned start at *start* and end at *end*, given in Angstrom.

DifDist *atomIdx1 atomIdx2 atomIdx3 atomIdx4 start end* Scans the difference between the distances $R(1,2)-R(3,4)$ of the atoms with indices *atomIdx1–4*, as given by their order in the *System%Atoms* block. The values to be scanned start at *start* and end at *end*, given in Angstrom.

Note that multiple of these coordinate descriptors can be combined within a single *ScanCoordinate* block. This combines the individual coordinates into one compound coordinate, i.e. all coordinates will transit together through their respective ranges. In this way the symmetric stretch in water could be scanned by specifying the following single *ScanCoordinate* block (assuming that the oxygen atom is the first in the *System%Atoms* block):

```
ScanCoordinate
  Distance  1 2  0.8 1.1
  Distance  1 3  0.8 1.1
End
```

A multidimensional PES scan can be performed by specifying multiple *ScanCoordinate* blocks in the input. To scan the space spanned by the bending and symmetric stretch modes in water, one would use the following scan coordinates:

```
ScanCoordinate
  Distance  1 2  0.8 1.1
  Distance  1 3  0.8 1.1
End
ScanCoordinate
  Angle    2 1 3  90 130
End
```

In principle an arbitrary number of *ScanCoordinate* blocks can be combined to specify the scanned configuration space. However, the total number of sample points is the product of the number of points along all coordinates, and hence grows quickly with the number of dimensions. Furthermore, only 1D (linear transit) and 2D PES scans can be visualized in the GUI. We therefore suggest sticking with ≤ 2 dimensional PES scans. (Note that it is possible to constrain additional degrees of freedom through the *Constraints* block. This could be used to sample a few points along a third dimension “manually”, while still being able to see the surfaces in the GUI.)

By default the engine result files for the individual PES points are not saved on disk, as this can easily lead to huge amounts of data to be stored. This behavior can be changed with the *PESScan%CalcPropertiesAtPESPoints* keyword:

PESScan

CalcPropertiesAtPESPoints

Type Bool

Default value No

Description Whether to perform an additional calculation with properties on all the sampled points of the PES. If this option is enabled AMS will produce a separate engine output file for every sampled PES point.

Note that this performs a full single point calculation on every sampled PES point, including the calculation of any *PES point properties* (page 141) selected in *Properties* block.

4.5.1 Troubleshooting

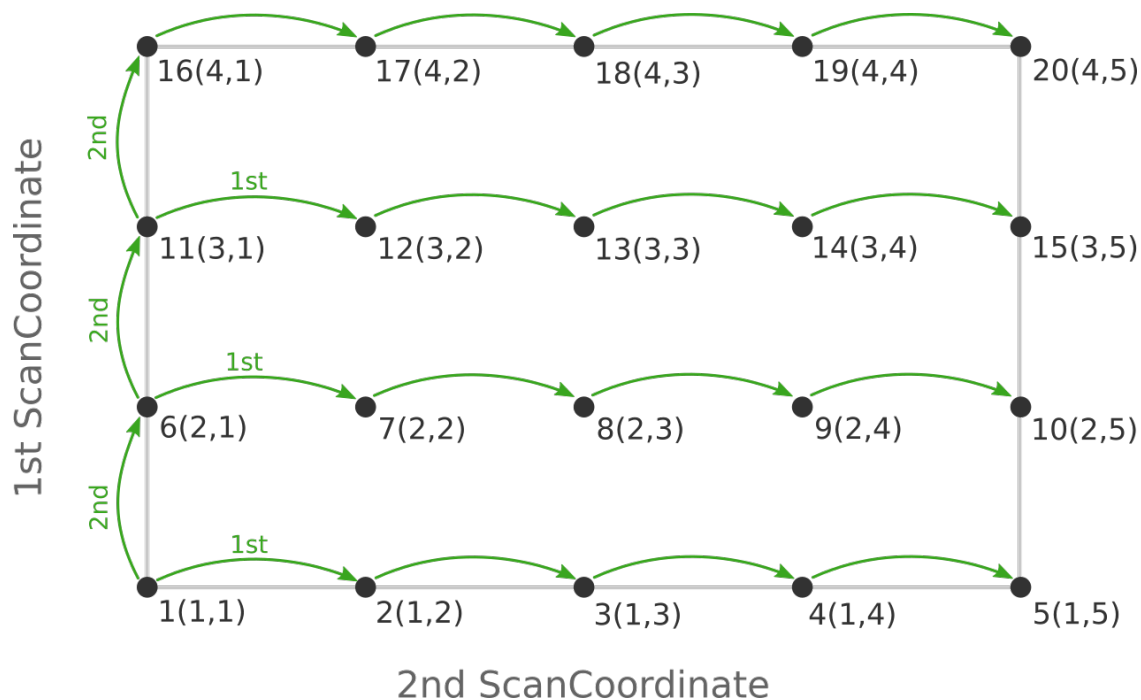
Technically all PES scan calculations are conducted as a series of geometry optimizations with constraints for the scanned coordinates, where the value of the constraint varies slowly through the scanned range. In this way every sampled point on the potential energy surface corresponds to a particular set of constraints. As with any geometry optimization, it can happen that an optimization towards a particular point on the potential energy surface does not converge. This is the most common problem encountered during PES scan calculations.

Since PES scans are implemented as a series of geometry optimizations, they are influenced by the settings used for the geometry optimizer, e.g. its convergence thresholds and the maximum number of steps before an optimization is considered to have failed. The optimizer is configured in the `GeometryOptimization` block, see the page on [geometry optimization](#) (page 43) in the AMS manual. Note that PES scans always use the *Quasi-Newton* (page 49) optimizer.

While tweaking the geometry optimizer's settings can sometimes help with convergence problems, these problems can also be easily caused by errors in the user input.

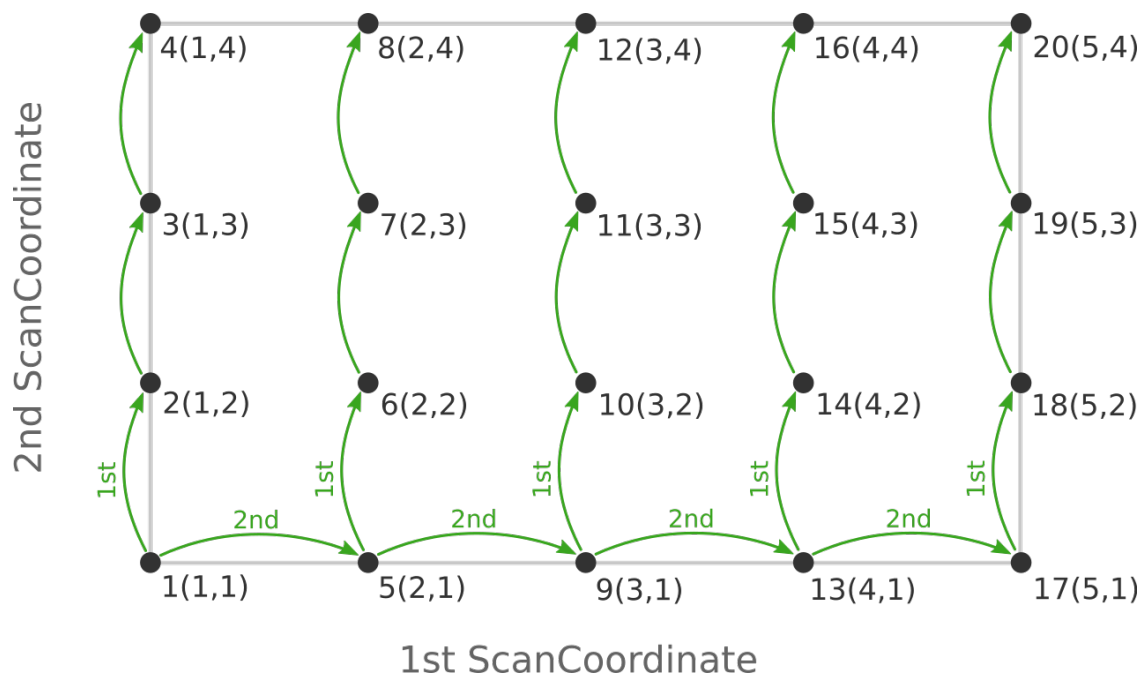
A very common problem is that the geometry in the input, i.e. the `System` block, is incompatible with the starting values of the scanned coordinates. This would for example be the case if one wants to scan a dihedral angle from 0 to 90 degrees, but the actual angle on the input geometry is close to 90 degrees. In this case it would be better to flip the scanned range from 90 to 0 degrees, so that the input geometry already close to the first sampled point on the PES. Otherwise the optimization for the first point has to cross a very long distance on the PES, making convergence much harder. AMS automatically detects this and prints a warning. We generally advise preparing the input geometry for a PES scan by first running a geometry optimization with constraints set to lower bound of the scanned coordinate intervals.

For multidimensional PES scans the order in which the PES points are visited depends on the order in which the scanned coordinates are specified, i.e. the order of the `ScanCoordinate` blocks in the input. Generally, the order in which the PES points are visited is such that the coordinate which was specified in the first `ScanCoordinate` block varies **slowest**. This is illustrated in the following figure:



Here the scan starts at point 1 (1, 1) at the bottom left corner of the PES and first moves along the entire range of the 2nd scan coordinate, before taking a step along the 1st coordinate to point 6 (2, 1). The same PES points could be visited in a different order (and under different names) if the order of the two `ScanCoordinate` blocks is reversed

in the AMS input:



Depending on the shape of the scanned potential energy surface a particular order of visiting the PES points might be easier or harder for the optimizer, and convergence problems can sometimes be fixed by simply changing the order of the scanned coordinates. In the example above, it might be that scanning along the “vertical” direction is “harder” than scanning along the “horizontal” direction. In this case one should use the scan order from the first picture, which has only three “vertical” steps (whereas the other scan order has 15).

Note that AMS has a little safe-guard built in to help with PES scan convergence issues: If the optimization towards a particular PES point did not succeed in the initial attempt, AMS will later try again, but starting from a different (converged) point close to unconverged one. This “PES gap filling” happens at the very end of the calculation, after the initial scan has been completed. This gap filling step is enabled by default, but can be controlled with the `PESScan%FillUnconvergedGaps` keyword:

PESScan

FillUnconvergedGaps

Type Bool

Default value Yes

Description After the initial pass over the PES, restart the unconverged points from converged neighboring points.

4.6 Nudged Elastic Band (NEB)

The Nudged Elastic Band (NEB) method¹ can be used to find a reaction path and the transition state between a reactant and a product state.

At the beginning of a NEB calculation, the geometry of the initial and final systems are optimized to minimize their energy (unless the `OptimizeEnds` (page 65) option is set to False).

¹ G. Henkelman, B.P. Uberuaga and H. Jonsson, *A climbing image nudged elastic band method for finding saddle points and minimum energy paths*, *Journal of Chemical Physics* 113, 9901 (2000) (<https://doi.org/10.1063/1.1329672>)

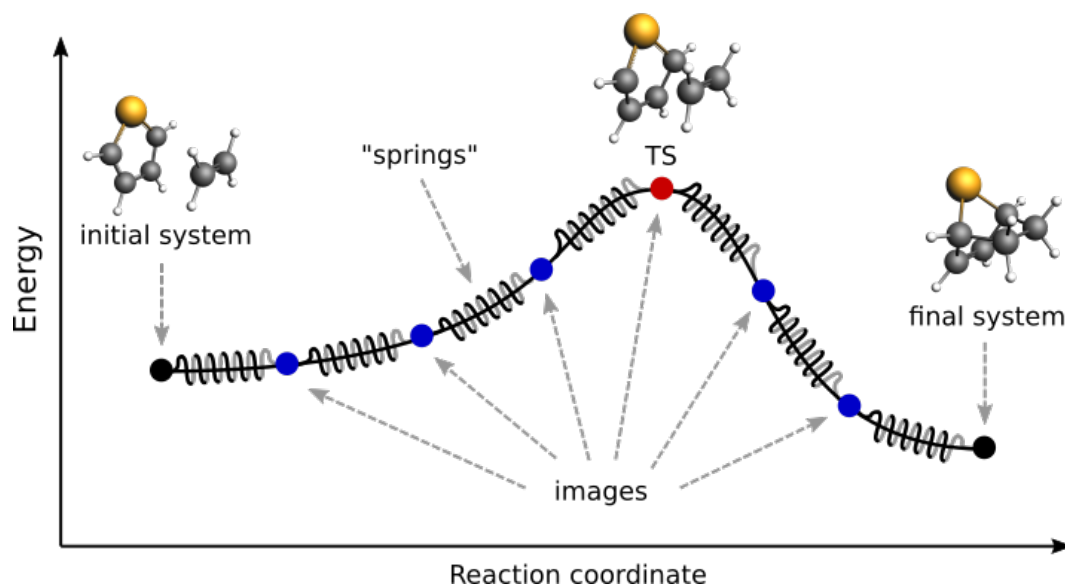


Fig. 4.1: Pictorial representation of a reaction path computed with NEB

Then, a rough approximation of the reaction path is build: a set of *images* is created by performing a linear interpolation between the **initial** and **final** systems. Optionally, an **intermediate** system can be provided, in which case the interpolation is performed between the initial and intermediate systems, and then between the intermediate and final systems.

Finally, a reaction path is found by performing a simultaneous optimization of all the images. In the NEB method the images are not independent from each other. The force on each image depend on its neighboring images: at each step the forces parallel to the reaction path are eliminated and a so-called spring force is added that tries to keep each image in the middle between its neighbors. This does not let images slide to the initial or final reaction state and ensures that they are evenly distributed along the reaction path.

During the NEB path optimization, a climbing image algorithm is used to drive the highest-energy image in the path to the transition state (unless the *Climbing* (page 65) option is set to False).

Be aware that NEB is a computationally expensive method, typically involving hundreds if not thousands of energy and gradients evaluations.

See also:

- [Nudged Elastic Band \(NEB\) Examples](#) (page 269)
- [GUI Nudged Elastic Band tutorial](#)

4.6.1 Input

A NEB calculation in AMS is triggered by setting the `Task` to NEB:

```
Task NEB
```

The NEB method requires two or three input systems. The first, unnamed system is used as the **initial system** and the system called `final` is used as a **final system**. These two systems are mandatory. This is an example of system-definitions for a HCN isomerization reaction:


```

Task NEB

# This is the initial system:
System
  Atoms
    C  0.0000  0.0000  0.0000
    N  1.1800  0.0000  0.0000
    H  2.1960  0.0000  0.0000
  End
End

# This is the final system (note the header 'final' in the next line):
System final
  Atoms
    C  0.0000  0.0000  0.0000
    N  1.1630  0.0000  0.0000
    H -1.0780  0.0000  0.0000
  End
End

```

Optionally, a third system, called *intermediate*, can be used to provide a better approximation for the transition state. The intermediate system will be placed in the middle of the chain. When providing three input systems it may be a good idea to optimize the ends in advance and set *OptimizeEnds* (page 65) to False to prevent creating an unbalanced reaction path.

Note that not only the atomic coordinates, but also the lattice parameters and the charge (if non-zero) must be set for all input systems.

Important: The order in which atoms are specified in the `System%Atoms` blocks **must be the same for the initial and final systems** (if you provide an intermediate system, you must use a consistent atom-ordering for that too). The order of the atoms must be consistent because the images-interpolation algorithm maps the n-th atom of the initial system to the n-th atom of the final system.

All NEB-specific options are specified in the NEB input block:

```

NEB
  Climbing Yes/No
  ClimbingThreshold float
  Images integer
  InterpolateInternal Yes/No
  InterpolateShortest Yes/No
  Iterations integer
  Jacobian float
  MapAtomsToCell Yes/No
  OldTangent Yes/No
  OptimizeEnds Yes/No
  OptimizeLattice Yes/No
  Parallel
    nCoresPerGroup integer
    nGroups integer
    nNodesPerGroup integer
  End
  ReOptimizeEnds Yes/No
  Restart string
  Skewness float
  Spring float

```

(continues on next page)

End

All keys of the NEB block have reasonable defaults or are optional. Thus, in principle, the NEB block can be omitted altogether. These are the main options:

NEB**Images**

Type Integer

Default value 8

GUI name Number of images

Description Number of NEB images (not counting the chain ends). Using more images will result in a smoother reaction path and can help with convergence problems, but it will also increase the computation time.

Iterations

Type Integer

GUI name Maximum number of iterations

Description Maximum number of NEB iterations. The default value depends on the number of degrees of freedom (number of images, atoms, periodic dimensions).

Spring

Type Float

Default value 1.0

Unit Hartree/Bohr²

GUI name Spring value

Description Spring force constant in atomic units.

Skewness

Type Float

Default value 1.0

GUI name Skewness

Description Degree of how much images are shifted towards or away from the TS, which may help tackle problems with a long reaction path (for example involving a loose adsorption complex) without needing too many images. A value greater than 1 will make sure that images are concentrated near the transition state. The optimal value depends on the path length, the number of images (larger [Skewness] may be needed for a longer path and fewer images). Technically [Skewness] is equal to the ratio between the optimized distances to the lower and the higher neighbor image on the path.

Climbing

Type Bool

Default value Yes

GUI name Climb highest image to TS

Description Use the climbing image algorithm to drive the highest image to the transition state.

ClimbingThreshold**Type** Float**Default value** 0.0**Unit** Hartree/Bohr**GUI name** CI force threshold

Description Climbing image force threshold. If ClimbingThreshold > 0 and the max perpendicular force component is above the threshold then no climbing is performed at this step. This entry can be used to get a better approximation for the reaction path before starting the search for the transition state. A typical value is 0.01 Hartree/Bohr.

InterpolateInternal**Type** Bool**Default value** Yes**GUI name** Interpolate in Internal coordinates

Description The initial NEB image geometries are calculated by interpolating between the initial and the final state. By default, for non-periodic systems the interpolation is performed in Internal coordinates but the user can choose to do it in the Cartesian ones. For periodic systems the interpolation is always done in Cartesian coordinates.

InterpolateShortest**Type** Bool**Default value** Yes**GUI name** Interpolate across cell boundary

Description Allow interpolation across periodic cell boundaries. Set to false if an atom is intended to move more than half across the simulation box during reaction.

OptimizeEnds**Type** Bool**Default value** Yes**GUI name** Optimize reactants/products

Description Start the NEB with optimization of the reactant and product geometries.

Restart**Type** String**GUI name** Restart from

Description Provide an ams.rkf file from a previous NEB calculation to restart from. It can be an unfinished NEB calculation or one performed with different engine parameters.

ReOptimizeEnds**Type** Bool**Default value** No**GUI name** Re-optimize reactants/products

Description Re-optimize reactant and product geometries upon restart.

The following keys are related to solid-state NEB (SS-NEB):

NEB**OptimizeLattice****Type** Bool**Default value** No**GUI name** Optimize lattice**Description** Turn on the solid-state NEB (SS-NEB).**Jacobian****Type** Float**GUI name** Jacobian value**Description** Scaling factor used to convert the lattice strain to a NEB coordinate value. Default value: $\sqrt{N} \cdot (V/N)^{1/d}$, where V - lattice volume (area for 2D, length for 1D), N - number of atoms, and d - number of periodic dimensions.**MapAtomsToCell****Type** Bool**Default value** Yes**GUI name** Map atoms to cell**Description** Translate atoms to the [-0.5,0.5] cell before every step. This option cannot be disabled for SS-NEB.

At each iteration, the images may be computed in parallel. The parallel execution is normally configured completely automatically, but users can override the automatic parallelization using the keys in the `Parallel` block.

NEB**Parallel****Type** Block**Description** Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.**nCoresPerGroup****Type** Integer**GUI name** Cores per group**Description** Number of cores in each working group.**nGroups****Type** Integer**GUI name** Number of groups**Description** Total number of processor groups. This is the number of tasks that will be executed in parallel.**nNodesPerGroup****Type** Integer**GUI name** Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

The following keys modify other aspects of the NEB and should, in principle, be left to their defaults:

NEB

OldTangent

Type Bool

Default value No

GUI name Use old tangent

Description Turn on the old central difference tangent.

4.6.2 Frozen atom constraints

It is possible to perform NEB with part of the system frozen, using any of the following keys of the Constraints block or a combination thereof:

Constraints

Atom

Type Integer

Recurring True

Description Fix the position of an atom. Just one integer referring to the index of the atom in the [System%Atoms] block.

AtomList

Type Integer List

Recurring True

Description Fix positions of the specified atoms. A list of integers referring to indices of atoms in the [System%Atoms] block.

FixedRegion

Type String

Recurring True

Description Fix positions of all atoms in a region.

Note: the frozen atom constraints will be enforced both during the geometry optimizations of the initial and final systems and during the NEB optimization.

4.6.3 Optimizations and convergence criteria

The NEB path is optimized using a limited-memory BFGS (l-BFGS) method where the system being optimized is a union of all NEB images with their respective molecular and spring forces.

The NEB convergence thresholds are defined in the *GeometryOptimization%Convergence block* (page 44). NEB is considered converged when the following criteria are satisfied:

- the change in the highest image energy must be less than [GeometryOptimization%Convergence%Energy]

- the max atomic force component for the highest image must be less than [GeometryOptimization%Convergence%Gradients]
- the max atomic force component for all other images must be less than ten times the [GeometryOptimization%Convergence%Gradients] value.

If the optimization of the initial NEB end point fails to converge, you can try using the *FIRE optimization method* (page 48).

4.6.4 Output

Results are printed to the text output and stored in the binary result file ‘ams.rkf’. In the ‘ams.rkf’ file, NEB calculation results are stored in the History section just like in a normal geometry optimization. The NEB section of the RKF file contains additional, NEB-specific, information.

The NEB reaction path can be visualized using the **AMSmovie GUI module**.

4.6.5 Troubleshooting

- In case the geometry optimization of the initial and final systems fails: try using the *FIRE optimization method* (page 48)
- In case the optimization of the NEB path does not converge:
 - make sure that the order in which the atoms are defined is consistent between the initial and final systems (see the *important* note in the *NEB input section* (page 64))
 - try increasing the number of *NEB images* (page 65)
 - try tweaking the *TrustRadius* (page 55) or *TrialStep* (page 55) options (see *Limited-memory BFGS* (page 55))
 - try specifying an **intermediate** system

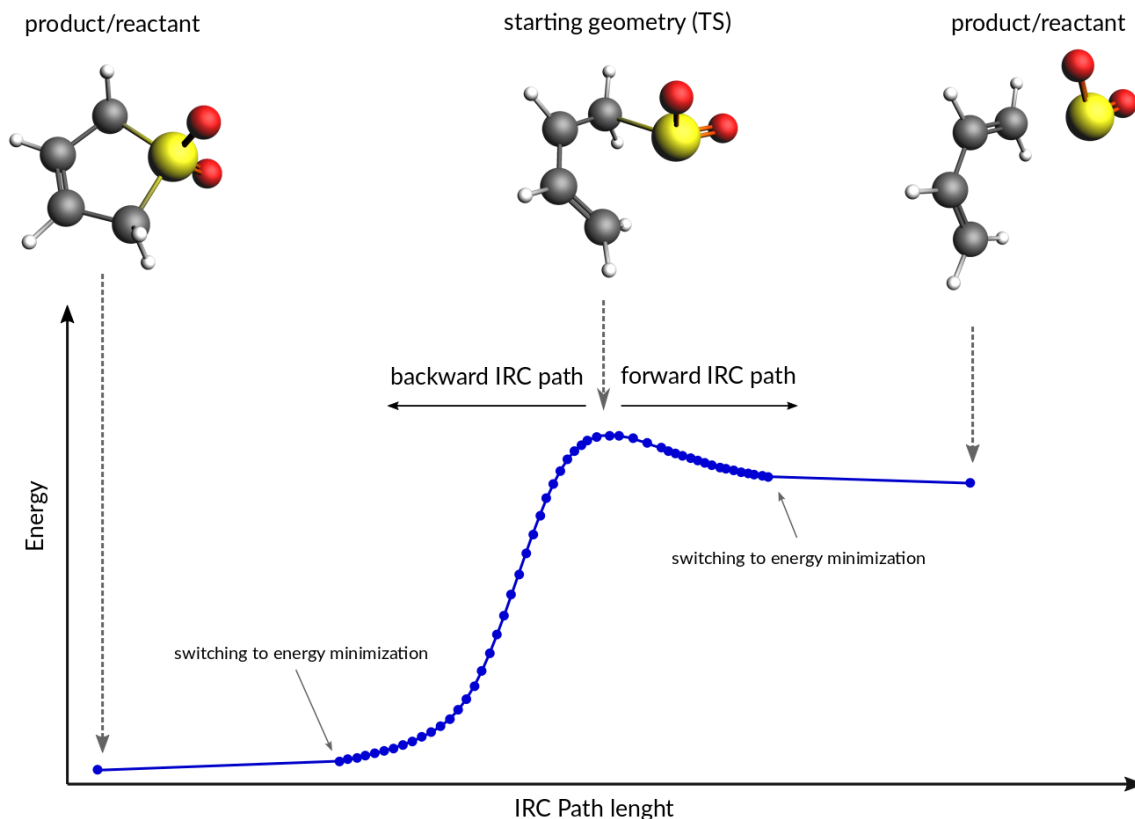
4.7 Intrinsic Reaction Coordinate (IRC)

The path of a chemical reaction can be traced from the transition state (TS) to the products and/or reactants using the Intrinsic Reaction Coordinate (IRC) method^{1,2}. **The method assumes that the starting geometry is a fair approximation of the TS.** A minimum energy profile (MEP) is defined as the steepest-descent path on the potential energy surface from the transition state down towards a local minimum. An IRC path is defined similarly but in the mass-weighted coordinates³, which means that instead of the steepest descent direction it follows that of the maximum instantaneous acceleration. This makes IRC somewhat related to the Molecular Dynamics method. The energy profile is obtained as well as the length and curvature properties of the path, providing the basic quantities for an analysis of the reaction path.

¹ L. Deng, T. Ziegler and L. Fan, *A combined density functional and intrinsic reaction coordinate study on the ground state energy surface of H₂ CO*, Journal of Chemical Physics 99, 3823 (1993) (<https://doi.org/10.1063/1.466129>)

² L. Deng and T. Ziegler, *The determination of Intrinsic Reaction Coordinates by density functional theory*, International Journal of Quantum Chemistry 52, 731 (1994) (<https://doi.org/10.1002/qua.560520406>)

³ C. Gonzalez and H.B. Schlegel, *Reaction Path Following In Mass-Weighted Internal Coordinates* J. Phys. Chem. 94, 5523-5527 (1990) (<https://doi.org/10.1021/j100377a021>)



See also:

[Examples](#) (page 241) and the [PES exploration for hydrohalogenation tutorial](#)

4.7.1 Method details

Calculation of an IRC path consists of two nested loops, the so-called outer and inner loops. The outer loop runs over IRC points and the inner loop is over geometry optimization steps for the given IRC point. The first IRC point starts from the transition state geometry, which is a saddle point, in one of the two possible downhill directions. Each IRC point after that starts from the optimized geometry of the previous point. At the start of every step, the pivot point is determined, which is a point at the $\text{Step}/2$ distance in the direction opposite to the gradient. When working in the mass-weighted coordinates, this direction corresponds to the acceleration of the corresponding atom. The final point of the given IRC step corresponds to the energy minimum point at the same distance ($\text{Step}/2$) from the pivot point further downhill. More precisely, the coordinates of the target point are optimized during the inner loop to minimize projection of the gradient on the hypersphere of radius $\text{Step}/2$ around the pivot point. The angle between the (pivot-start) and (pivot-final) vectors determines the curvature of the reaction path. If this angle becomes smaller than 90 degrees then the IRC scan is considered to have reached vicinity of an endpoint and the program switches to energy minimization (options for this energy minimization can be specified in the [Geometry Optimization](#) (page 43) block.). If the angle is between 90 and 120 degrees then the current IRC step is canceled and a new one is started from the same starting point with half the initial Step parameter. In all other cases the optimized geometry becomes a starting point for the next IRC step. By default, when the forward path is completed the backward one is started from the same TS geometry. When both forward and backward paths are complete, a summary of the whole reaction path is printed to the output.

4.7.2 Input

The IRC scan in AMS is triggered by setting the Task to IRC:

```
Task IRC
```

All IRC-related options are specified in the IRC input block:

```
IRC
  Convergence
    Gradients float
    Step float
  End
  CoordinateType [Cartesian | Delocalized]
  Direction [Both | Forward | Backward]
  InitialHessian
    File string
    Type [Calculate | FromFile]
  End
  KeepConvergedResults Yes/No
  MaxIRCSteps integer
  MaxIterations integer
  MaxPoints integer
  MinEnergyProfile Yes/No
  MinPathLength float
  Restart
    File string
    RedoBackward integer
    RedoForward integer
  End
  Step float
End
```

All keys of the IRC block have reasonable defaults or are optional. Thus, in principle, the IRC block can be omitted altogether. These are some of the main options:

IRC

Direction

Type Multiple Choice

Default value Both

Options [Both, Forward, Backward]

Description Select direction of the IRC path. The difference between the Forward and the Backward directions is determined by the sign of the largest component of the vibrational normal mode corresponding to the reaction coordinate at the transition state geometry. The Forward path correspond to the positive sign of the component. If Both is selected then first the Forward path is computed followed by the Backward one.

Step

Type Float

Default value 0.2

GUI name Step size

Description IRC step size in mass-weighted coordinates, $\sqrt{\text{amu}} \cdot \text{bohr}$. One may have to increase this value when heavy atoms are involved in the reaction, or decrease it if the reactant or products are very close to the transition state.

InitialHessian

Type Block

Description Options for initial Hessian at the transition state. The first eigenvalue of the initial Hessian defines direction of the first forward or backward step. This block is ignored when restarting from a previous IRC calculation because the initial Hessian found in the restart file is used.

File

Type String

GUI name File

Description If 'Type' is set to 'FromFile' then in this key you should specify the RKF file containing the initial Hessian (or the ams results dir. containing it). This can be used to load a Hessian calculated previously with the 'Properties%Hessian' keyword. If you want to also use this file for the initial geometry then also specify it in a 'LoadSystem' block.

Type

Type Multiple Choice

Default value Calculate

Options [Calculate, FromFile]

GUI name Initial Hessian

Description Calculate the exact Hessian for the input geometry or load it from the results of a previous calculation.

The following keys set limits on the number of steps for the inner and outer IRC loops and, related to that, the geometry optimization criteria. Note that tighter criteria may require a greater MaxIterations limit. Please also note that the outer loop limits are valid for each half of the path (forward and backward) separately. That is, if all settings are left at their defaults then up to 200 IRC points may be calculated, each of them may require up to 300 energy evaluations.

IRC

MaxIRCSteps

Type Integer

GUI name Maximum IRC steps

Description Soft limit on the number of IRC points to compute in each direction. After the specified number of IRC steps the program will switch to energy minimization and complete the path. This option should be used when you are interested only in the reaction path area near the transition state. Note that even if the soft limit has been hit and the calculation has completed, the IRC can still be restarted with a 'RedoBackward' or 'RedoForward' option.

MaxPoints

Type Integer

Default value 100

GUI name Maximum points

Description Hard limit on the number of IRC points to compute in each direction. After the specified number of IRC steps the program will stop with the current direction and switch to the next one. If both 'MaxPoints' and 'MaxIRCSteps' are set to the same value then 'MaxPoints' takes precedence, therefore this option should be used to set a limit on the number of IRC steps if you intend to use the results later for a restart.

MaxIterations

Type Integer

Default value 300

GUI name Maximum iterations

Description The maximum number of geometry iterations allowed to converge the inner IRC loop. If optimization does not converge within the specified number of steps, the calculation is aborted.

Convergence

Type Block

Description Convergence at each given point is monitored for two items: the Cartesian gradient and the calculated step size. Convergence criteria can be specified separately for each of these items. The same criteria are used both in the inner IRC loop and when performing energy minimization at the path ends.

Gradients

Type Float

Default value 0.001

Unit Hartree/Angstrom

GUI name Gradient convergence

Description Convergence criterion for the max component of the residual energy gradient.

Step

Type Float

Default value 0.001

Unit Angstrom

GUI name Step convergence

Description Convergence criterion for the max component of the step in the optimization coordinates.

MinPathLength

Type Float

Default value 0.1

Unit Angstrom

Description Minimum length of the path required before switching to energy minimization. Use this to overcome a small kink or a shoulder on the path.

The following keys modify other aspects of the IRC scan:

IRC

CoordinateType

Type Multiple Choice

Default value Cartesian

Options [Cartesian, Delocalized]

GUI name Coordinates used for optimization

Description Select the type of coordinates in which to perform the optimization. Note that the Delocalized option should be considered experimental.

MinEnergyProfile

Type Bool

Default value No

GUI name Minimum energy profile

Description Calculate minimum energy profile (i.e. no mass-weighting) instead of the IRC.

KeepConvergedResults

Type Bool

Default value Yes

Description Keep the binary RKF result file for every converged IRC point. These files may contain more information than the main ams.rkf result file.

It is possible to restart an IRC calculation that crashed, has been killed or exceeded the MaxPoints limit, or to recompute the path starting from a certain point, using the Restart key:

IRC

Restart

Type Block

Description Restart options. Upon restart, the information about the IRC input parameters and the initial system (atomic coordinates, lattice, charge, etc.) is read from the restart file. The IRC input parameters can be modified from input. Except for 'MaxPoints' and 'Direction' all parameters not specified in the input will use their values from the restart file. The 'MaxPoints' and 'Direction' will be reset to their respective default values if not specified in the input. By default, the IRC calculation will continue from the point where it left off. However, the 'RedoForward' and/or 'RedoBackward' option can be used to enforce recalculation of a part of the reaction path, for example, using a different 'Step' value.

File

Type String

GUI name Restart

Description Name of an RKF restart file generated by a previous IRC calculation. Do not use this key to provide an RKF file generated by a TransitionStateSearch or a SinglePoint calculation, use the 'LoadSystem' block instead.

RedoBackward

Type Integer

Default value 0

Description IRC step number to start recalculating the backward path from. By default, if the backward path has not been completed then start after the last completed step. If the

backward path has been completed and the 'RedoBackward' is omitted then no point on the backward path will be recomputed.

RedoForward

Type Integer

Default value 0

Description IRC step number to start recalculating the forward path from. By default, if the forward path has not been completed then start after the last completed step. If the forward path has been completed and the 'RedoForward' is omitted then no point on the forward path will be recomputed.

4.7.3 Output

A summary of reaction path is printed to the output file at the end of the IRC calculation.

The IRC reaction path can be visualized using the AMSmovie GUI module.

Results of an IRC calculation are also stored in the History section of the 'ams.rkf' file, just like in a normal geometry optimization. In addition to the standard KF variables such as "Coords" and "Energy", the following IRC-specific variables are also created:

- *IRCDirection* - IRC direction to which this point belongs: 1 - forward, 2 - backward.
- *IRCIteration* - the IRC (a.k.a. the outer loop) iteration number.
- *OptIteration* - the geometry optimization (a.k.a. the inner loop) iteration number (0 means the results correspond to the converged geometry at this IRC step).
- *IRCGradMax* - value of the max component of the IRC gradient that determines convergence of the inner loop.
- *IRCGradRms* - the RMS value of the IRC gradient that determines convergence of the inner loop. Both the *ircGradRms* and the *ircGradMax* are given in the mass-weighted atomic units for IRC steps and in the atomic units for the final minimization loop.
- *ArcLength* - length, in Angstrom, of the arc that connects the initial and the final point of this IRC step. The corresponding pivot point is located near the the middle point of the arc.
- *Angle* - value of the angle (in degrees) between lines connecting the pivot point with the initial and final points. A value of 180 degrees means the path is passing straight through the pivot point, while a smaller value means the path makes a bend at this point.
- *PathLength* - sum of the *ArcLength* values from the transition state up to this point, in Angstrom.
- *Converged* - a Fortran logical value containing the convergence status of the given geometry.

The IRC section of the RKF file contains all the data needed for a successful restart procedure.

4.8 Excited state optimizations

It depends on the engine if it is possible to do an electronically excited state optimization, like an electronically excited state *geometry optimization* (page 43), *transition state search* (page 57), *linear transit*, *PES scan* (page 60), or *IRC* (page 70). Required is that the engine should be able to calculate the nuclear gradient for a particular electronically excited state. ADF and DFTB are engines that can do such calculation. If the excited state gradient is available one can also calculate an *IR spectrum* (page 153) of the excited state, and one could calculate a *vibrationally resolved electronic spectrum* (page 188).

One should look in the documentation of the engine how to set up the calculation of the excited state gradient.

4.9 Molecular dynamics

Molecular dynamics (MD) can be used to simulate the evolution of a system in time.

See also:

- *Examples* (page 241)
- *AMS GUI Tutorial*
- *MD trajectory analysis tool* (page 228)

To perform a MD simulation, first select the corresponding Task:

```
Task MolecularDynamics
```

All aspects of the simulation can then be configured using the `MolecularDynamics` block.

```
MolecularDynamics
  AddMolecules
    AtomTemperature float
    ContactDistance float
    Coords float_list
    CoordsBox float_list
    CoordsSigma float_list
    DeviationAngle float
    Energy float
    EnergySigma float
    FractionalCoords float_list
    FractionalCoordsBox float_list
    FractionalCoordsSigma float_list
    Frequency integer
    MinDistance float
    NumAttempts integer
    Rotate Yes/No
    StartStep integer
    StopStep integer
    System string
    Temperature float
    TemperatureSigma float
    Velocity float
    VelocityDirection float_list
    VelocitySigma float
  End
  Barostat
    BulkModulus float
    ConstantVolume Yes/No
    Duration integer_list
    Equal [None | XYZ | XY | YZ | XZ]
    Pressure float_list
    Scale [XYZ | Shape | X | Y | Z | XY | YZ | XZ]
    Tau float
    Type [None | Berendsen | MTK]
  End
  BondBoost
    Chain
      AtomNames string
      MaxDistances float_list
      MinDistances float_list
```

(continues on next page)

(continued from previous page)

```

End
DistanceRestraint string
NSteps integer
NumInstances integer
End
CRESTMTD
AddEnergy Yes/No
GaussianScaling
  ScaleGaussians Yes/No
  ScalingSlope float
End
Height float
NGaussiansMax integer
NSteps integer
RestartFile string
Width float
End
CVHD
Bias
  DampingTemp float
  Delta float
  Height float
End
ColVarBB
  at1
    region string
    symbol string
  End
  at2
    region string
    symbol string
  End
  cutoff float
  p integer
  rmax float
  rmin float
  End
  Frequency integer
  StartStep integer
  StopStep integer
  WaitSteps integer
End
CalcPressure Yes/No
Checkpoint
  Frequency integer
  WriteProperties Yes/No
End
Deformation
  LatticeVelocity # Non-standard block. See details.
  ...
  End
  LengthRate float_list
  LengthVelocity float_list
  Period float
  ScaleAtoms Yes/No
  StartStep integer
  StopStep integer

```

(continues on next page)

(continued from previous page)

```

StrainRate # Non-standard block. See details.
    ...
End
TargetLattice # Non-standard block. See details.
    ...
End
TargetLength float_list
Type [Linear | Exponential | Sine | Cosine]
End
Gravity
    Acceleration float
End
HeatExchange
    HeatingRate float
    Method [Simple | HEX | eHEX]
    Sink
        AtomList integer_list
        Box
            Amax float
            Amin float
            Bmax float
            Bmin float
            Cmax float
            Cmin float
        End
        Region string
    End
    Source
        AtomList integer_list
        Box
            Amax float
            Amin float
            Bmax float
            Bmin float
            Cmax float
            Cmin float
        End
        Region string
    End
    StartStep integer
    StopStep integer
End
InitialVelocities
    File string
    Temperature float
    Type [Zero | Random | FromFile | Input]
    Values # Non-standard block. See details.
    ...
End
End
NSteps integer
Plumed
    Input # Non-standard block. See details.
    ...
End
Parallel
    nCoresPerGroup integer

```

(continues on next page)

(continued from previous page)

```
        nGroups integer
        nNodesPerGroup integer
    End
End
Preserve
    AngularMomentum Yes/No
    CenterOfMass Yes/No
    Momentum Yes/No
End
Print
    System Yes/No
    Velocities Yes/No
End
Remap
    Type [None | Atoms]
End
RemoveMolecules
    Formula string
    Frequency integer
    SafeBox
        Amax float
        Amin float
        Bmax float
        Bmin float
        Cmax float
        Cmin float
        FractionalCoordsBox float_list
    End
    SinkBox
        Amax float
        Amin float
        Bmax float
        Bmin float
        Cmax float
        Cmin float
        FractionalCoordsBox float_list
    End
    StartStep integer
    StopStep integer
End
ReplicaExchange
    AllowWrongResults Yes/No
    EWMALength integer
    SwapFrequency integer
    TemperatureFactors float_list
    Temperatures float_list
    nReplicas integer
End
Restart string
Thermostat
    BerendsenApply [Local | Global]
    ChainLength integer
    Duration integer_list
    Region string
    Tau float
    Temperature float_list
    Type [None | Berendsen | NHC]
```

(continues on next page)

(continued from previous page)

```

End
TimeStep float
Trajectory
  PrintFreq integer
  SamplingFreq integer
  TProfileGridPoints integer
  WriteBonds Yes/No
  WriteCharges Yes/No
  WriteGradients Yes/No
  WriteMolecules Yes/No
  WriteVelocities Yes/No
End
fbMC
  Frequency integer
  MassRoot float
  NSteps integer
  PrintFreq integer
  StartStep integer
  StepLength float
  StopStep integer
  Temperature float
End
End

```

4.9.1 General

The time evolution of the system is simulated by numerically integrating the equations of motion. A velocity Verlet integrator is used with a time step set by the `TimeStep` key. The MD driver will perform `NSteps` timesteps in total.

Because the overall computational cost depends on `NSteps` but not on `TimeStep`, it is desirable to set the timestep as large as possible to maximize the sampled timescale with a given computational budget. However, numerical integration errors grow rapidly as the timestep increases. These errors will cause a loss of energy conservation, crashes, and other artifacts. It is thus important to set the `TimeStep` value carefully, as its optimal value strongly depends on the studied system and simulated conditions.

As a rule of thumb, reasonable timesteps for systems not undergoing chemical reactions are 10-20 times lower than the period of the fastest vibration mode. Systems containing hydrogen atoms at room temperature can thus be accurately simulated using a 1 fs timestep. Longer timesteps can be safely used for systems containing only heavy atoms (vibration periods scale with the square root of the atomic mass). Conversely, the timestep needs to be made shorter for high-temperature simulations. The same also applies to simulations of chemical reactions, which are usually accompanied by significant transient local heating. The default timestep of 0.25 fs should work for most of these cases.

MolecularDynamics

NSteps

Type Integer

Default value 1000

GUI name Number of steps

Description The number of steps to be taken in the MD simulation.

TimeStep

Type Float

Default value 0.25

Unit Femtoseconds

Description The time difference per step.

During a long simulation, numerical integration errors will cause some system-wide quantities to drift from their exact values. For example, the system may develop a nonzero net linear velocity, causing an overall translation or flow. Non-periodic (molecular) and 1D-periodic systems may also develop nonzero angular momentum (overall rotation) and a Brownian motion of their center of mass through space. These problems are corrected by periodically removing any accumulated drift. This feature can be controlled using the `Preserve` key.

MolecularDynamics

Preserve

Type Block

Description Periodically remove numerical drift accumulated during the simulation to preserve different whole-system parameters.

AngularMomentum

Type Bool

Default value Yes

GUI name : Angular momentum

Description Remove overall angular momentum of the system. This option is ignored for 2D and 3D-periodic systems.

CenterOfMass

Type Bool

Default value No

GUI name : Center of mass

Description Translate the system to keep its center of mass at the coordinate origin. This option is not very useful for 3D-periodic systems.

Momentum

Type Bool

Default value Yes

GUI name Preserve: Total momentum

Description Remove overall (linear) momentum of the system.

4.9.2 Constrained molecular dynamics

It is possible to keep part of the system frozen in place during MD. This is achieved using the `Atom` or `AtomList` keys of the `Constraints` top-level input block.

Constraints

Atom

Type Integer

Recurring True

Description Fix the position of an atom. Just one integer referring to the index of the atom in the [System%Atoms] block.

AtomList

Type Integer List

Recurring True

Description Fix positions of the specified atoms. A list of integers referring to indices of atoms in the [System%Atoms] block.

Note: During simulations with a changing simulation box (NpT, NpH), the absolute Cartesian coordinates of the frozen atoms cannot be kept fixed. In this case, their fractional cell coordinates are maintained at the original values.

4.9.3 (Re-)Starting a simulation

The state of a system at the beginning of a simulation is defined by the positions and momenta of all atoms. The positions can be set in the input or loaded from a file as described under *Geometry, System definition* (page 29). Initial velocities are then supplied using the `InitialVelocities` block.

Probably the most common way to start up a simulation is to draw the initial velocities from a Maxwell-Boltzmann distribution by setting `Type=Random` and `Temperature` to a suitable value. Alternatively, velocities can be loaded from an `ams.rkf` file produced by an earlier simulation using `Type=FromFile` and `File`. This is the recommended way to start a production simulation from the results of a short preparation/equilibration run.

Velocities of all atoms in units of Å/fs can also be explicitly defined in the `Values` block after setting `Type=Input`. This is mainly useful to repeat or extend simulations done by other programs. For example, velocities can be extracted from the `vels` or `molodyn.vel` files used by the standalone ReaxFF program. A simple AWK script is supplied in `scripting/standalone/reaxff-ams/vels2ams.awk` to help with the conversion.

MolecularDynamics

InitialVelocities

Type Block

Description Sets the frequency for printing to stdout and storing the molecular configuration on the `.rkf` file.

File

Type String

Description AMS RKF file containing the initial velocities.

Temperature

Type Float

Unit Kelvin

GUI name Initial temperature

Description Sets the temperature for the Maxwell-Boltzmann distribution when the type of the initial velocities is set to random, in which case specifying this key is mandatory. AMSinput will use the first temperature of the first thermostat as default.

Type

Type Multiple Choice

Default value Random

Options [Zero, Random, FromFile, Input]

GUI name Initial velocities

Description Specifies the initial velocities to assign to the atoms. Three methods to assign velocities are available. Zero: All atom are at rest at the beginning of the calculation. Random: Initial atom velocities follow a Maxwell-Boltzmann distribution for the temperature given by the [MolecularDynamics%InitialVelocities%Temperature] keyword. FromFile: Load the velocities from a previous ams result file. Input: Atom's velocities are set to the values specified in the [MolecularDynamics%InitialVelocities%Values] block, which can be accessed via the Expert AMS panel in AMSinput.

Values

Type Non-standard block

Description This block specifies the velocity of each atom, in Angstrom/fs, when [MolecularDynamics%InitialVelocities%Type] is set to Input. Each row must contain three floating point values (corresponding to the x,y,z component of the velocity vector) and a number of rows equal to the number of atoms must be present, given in the same order as the [System%Atoms] block.

The MD module also supports exact restarts of interrupted simulations by pointing the `Restart` key to an `ams.rkf` file. This will restore the entire state of the MD module from the last available checkpoint (if the previous simulation was interrupted) or from the final state (if the previous simulation ended after `NSteps`). An earlier trajectory can thus be seamlessly extended by increasing `NSteps` and using `Restart`.

Note: `Restart` should be combined with `LoadSystem` from the same `ams.rkf` to restore the atomic positions.

Warning: The `Restart` feature is only intended for exact restarts, so the rest of the `MolecularDynamics` settings should be the same as in the original run. Only `NSteps` and engine settings (contents of the `Engine` block) can always be changed safely across restarts.

Although some MD settings (such as the trajectory sampling options) can in practice be changed without problems, changing others (such as thermostat or barostat settings) will cause the restart to fail or produce physically incorrect results. It is thus strongly recommended to only use `Restart` for exact continuation and `InitialVelocities Type=FromFile` together with `LoadSystem` otherwise.

MolecularDynamics

Restart

Type String

GUI name Restart from

Description The path to the `ams.rkf` file from which to restart the simulation.

4.9.4 Thermostats and barostats

By default, the MD simulation samples the microcanonical (NVE) ensemble. Although this is useful to check energy conservation and other basic physical properties, it does not directly map to common experimental conditions. The canonical (NVT) ensemble can be sampled instead by applying a `Thermostat`, which serves as a simulated heat bath around the system, keeping its average temperature at a set value.

AMS offers two thermostats with drastically different properties, mode of operation, and applicability, selected using the `Type` key:

Berendsen The Berendsen friction thermostat drives the system to a particular target temperature by rescaling the velocities of all atoms in each step. This ensures rapid (exponential) convergence of the temperature with a time constant τ . However, this thermostat produces an incorrect velocity distribution and should thus be avoided in all situations where correct energy fluctuations are important. Additionally, using a too short time constant τ tends to cause incorrect equipartition of energy between different degrees of freedom in the system, leading to the “flying ice cube” phenomenon. The time constant τ should thus be set as large as possible to limit these artifacts while still providing sufficient temperature control. Common values of τ for condensed-phase systems lie between 100 fs (strong damping, rapid convergence) and 10 ps (weak coupling with minimal artifacts).

This thermostat is mainly useful for systems far from equilibrium, for example during the initial preparation and equilibration phase of a simulation. The NHC thermostat should be preferred where possible.

NHC This enables a chain of coupled Nosé-Hoover thermostats. This method introduces artificial degrees of freedom representing the heat bath and ensures correct sampling of the canonical ensemble. The combined total energy of the system and the heat bath is conserved and shown in the GUI as `Conserved Energy`. Checking this quantity for drift and artifacts thus offers a valuable test of the correctness of the simulation. This thermostat exhibits oscillatory relaxation with a period of τ . It is thus not well suited for systems starting far from equilibrium, because the oscillations may take long to settle. The time constant τ should be at least comparable to the period of some natural oscillation of the system to ensure efficient energy transfer. It is commonly on the order of hundreds of femtoseconds, although higher values may be used if weak coupling is desired.

Multiple independent thermostats can be used to separately control different non-overlapping regions of the system at the same time. This is done by first defining appropriate *Regions* (page 34) in the `System` block and then specifying the `Thermostat` block multiple times with the `Region` key of each thermostat set to an appropriate region expression.

MolecularDynamics

Thermostat

Type Block

Recurring True

Description This block allows to specify the use of a thermostat during the simulation. Depending on the selected thermostat type, different additional options may be needed to characterize the specific thermostat’ behavior.

BerendsenApply

Type Multiple Choice

Default value Global

Options [Local, Global]

GUI name Apply Berendsen

Description Select how to apply the scaling correction for the Berendsen thermostat: - per-atom-velocity (Local) - on the molecular system as a whole (Global).

ChainLength

Type Integer

Default value 10

GUI name NHC chain length

Description Number of individual thermostats forming the NHC thermostat

Duration**Type** Integer List**GUI name** Duration(s)**Description** Specifies how many steps should a transition from a particular temperature to the next one in sequence take.**Region****Type** String**Default value** ***Description** The identifier of the region to thermostat. The default '*' applies the thermostat to the entire system. The value can be a plain region name, or a region expression, e.g. '*-myregion' to thermostat all atoms that are not in myregion, or 'regionA+regionB' to thermostat the union of the 'regionA' and 'regionB'. Note that if multiple thermostats are used, their regions may not overlap.**Tau****Type** Float**Unit** Femtoseconds**GUI name** Damping constant**Description** The time constant of the thermostat.**Temperature****Type** Float List**Unit** Kelvin**GUI name** Temperature(s)**Description** The target temperature of the thermostat. You can specify multiple temperatures (separated by spaces). In that case the Duration field specifies how many steps to use for the transition from one T to the next T (using a linear ramp). For NHC thermostat, the temperature may not be zero.**Type****Type** Multiple Choice**Default value** None**Options** [None, Berendsen, NHC]**GUI name** Thermostat**Description** Selects the type of the thermostat.

Just like using a `Thermostat` to control the temperature of the system, a `Barostat` can be applied to keep the pressure constant by adjusting the volume. This enables sampling the isenthalpic-isobaric (NpH) ensemble by using only a barostat or the isothermal-isobaric (NpT) ensemble by combining a barostat and a thermostat. Unlike thermostats, a barostat always applies to the entire system and there can thus be at most one barostat defined.

AMS offers two barostats with similar properties to the related thermostats:

Berendsen The Berendsen friction-like isobaric ensemble method rescales the system in each step to drive the pressure towards a target value. Similarly to the Berendsen thermostat, the relaxation is exponential with a time constant `Tau`. Similar considerations for the choice of `Tau` apply as in the case of the thermostat, but the

value of τ for the barostat is usually at least several times higher than the corresponding τ used for the thermostat. This barostat does not have any conserved quantity.

MTK This enables the Martyna-Tobias-Klein extended Lagrangian barostat, which generates a true isobaric ensemble by integrating the cell parameters as additional degrees of freedom. This barostat is derived from the Andersen-Hoover isotropic barostat and the Parrinello-Rahman-Hoover anisotropic barostat. Like the NHC thermostat, it exhibits oscillatory relaxation unsuitable for systems far from equilibrium. This barostat must always be combined with a NHC thermostat. One instance of such thermostat coupled to the atoms as usual, while a second instance is created internally and coupled to the cell degrees of freedom.

MolecularDynamics

Barostat

Type Block

Description This block allows to specify the use of a barostat during the simulation.

BulkModulus

Type Float

Default value 2200000000.0

Unit Pascal

Description An estimate of the bulk modulus (inverse compressibility) of the system for the Berendsen barostat. This is only used to make τ correspond to the true observed relaxation time constant. Values are commonly on the order of 10-100 GPa ($1e10$ to $1e11$) for solids and 1 GPa ($1e9$) for liquids ($2.2e9$ for water). Use $1e9$ to match the behavior of standalone ReaxFF.

ConstantVolume

Type Bool

Default value No

Description Keep the volume constant while allowing the box shape to change. This is currently supported only by the MTK barostat.

Duration

Type Integer List

Description Specifies how many steps should a transition from a particular pressure to the next one in sequence take.

Equal

Type Multiple Choice

Default value None

Options [None, XYZ, XY, YZ, XZ]

Description Enforce equal scaling of the selected set of dimensions. They will be barostatted as one dimension according to the average pressure over the components.

Pressure

Type Float List

Unit Pascal

Description Specifies the target pressure. You can specify multiple pressures (separated by spaces). In that case the Duration field specifies how many steps to use for the transition from one p to the next p (using a linear ramp).

Scale

Type Multiple Choice

Default value XYZ

Options [XYZ, Shape, X, Y, Z, XY, YZ, XZ]

Description Dimensions that should be scaled by the barostat to maintain pressure. Selecting Shape means that all three dimensions and also all the cell angles are allowed to change.

Tau

Type Float

Unit Femtoseconds

GUI name Damping constant

Description Specifies the time constant of the barostat.

Type

Type Multiple Choice

Default value None

Options [None, Berendsen, MTK]

GUI name Barostat

Description Selects the type of the barostat.

Temperature and pressure regimes

Arbitrary temperature and pressure regimes can be generated by setting Temperature or Pressure to a list of values, corresponding to the successive set points. This needs to be accompanied by a Duration key specifying the length of each regime segment in steps:

```
Thermostat
  Temperature 0      300      300      500      500      300
  Duration    100     200      100     200      100
End
```

Note that there is always N-1 Duration values for N Temperature values. The target temperature of the thermostat in this example will evolve as follows:

1. Increase linearly from 0 to 300 K over 100 steps.
2. Stay constant at 300 K for 200 steps.
3. Increase linearly from 300 to 500 K over 100 steps.
4. Stay constant at 500 K for 200 steps.
5. Decrease linearly from 500 to 300 K over 100 steps.
6. Stay constant at 300 K for the rest of the simulation.

4.9.5 Trajectory sampling and output

A basic principle of the numerical integration of motion in MD is that the changes in the state of the system between successive time steps are small. This means that storing the results of every step is not useful, because all the data is strongly correlated. Instead, a snapshot of the system is taken every `N` steps, where `N` is set low enough to still capture the fastest motion of interest but high enough to avoid wasting space due to correlations. The resulting sequence of snapshots is then commonly called the trajectory.

AMS writes the trajectory to the `History` and `MDHistory` sections of `ams.rkf`, according to the settings in the `Trajectory` block. A snapshot of the system and various MD variables is stored every `SamplingFreq` timesteps. By default, this frequency is also used to print basic thermodynamic parameters of the simulation to the output and log file. Set `PrintFreq` to override this.

Frequently sampling a long trajectory can generate large volumes of data. If the space usage becomes a concern, one can selectively disable storing some parts of the trajectory to save space using the `Write*` keys. Note however that this will make it impossible to use some analysis methods on the resulting trajectory:

- `WriteBonds` is necessary for reaction network analysis (`ChemTraYzer`). Disabling `WriteBonds` also makes `AMSmovie` show only guessed bonds instead of those calculated by the engine.
- `WriteMolecules` is required by the Molecule Fractions panel in `AMSmovie`.
- `WriteVelocities` is required to calculate the velocity autocorrelation functions needed for diffusivity and IR spectra.

The trajectory itself contains only the data needed for subsequent analysis of the dynamics of the system. However, much more data is usually generated on every integration step. This includes, for example, the internal data used by an engine when evaluating the energies and forces. This information is normally discarded after each step, because it is often very large. However, a `Checkpoint` containing the complete internal state of the MD driver together with a result file generated by the engine is stored every `Frequency` steps. An interrupted simulation can then be restarted from this checkpoint using the `Restart` keyword. Additionally, the engine result files called `MDStep*.rkf` can also be used to extract various engine-specific details about the system, such as the orbitals for QM engines.

MolecularDynamics

Trajectory

Type Block

Description Sets the frequency for printing to stdout and storing the molecular configuration on the `.rkf` file.

PrintFreq

Type Integer

GUI name Printing frequency

Description Print current thermodynamic properties to the output every `N` steps. By default this is done every `SamplingFreq` steps.

SamplingFreq

Type Integer

Default value 100

GUI name Sample frequency

Description Write the the molecular geometry (and possibly other properties) to the `.rkf` file once every `N` steps.

TProfileGridPoints

Type Integer

Default value 0

Description Number of points in the temperature profile. If TProfileGridPoints > 0, a temperature profile along each of the three unit cell axes will be written to the .rkf file. By default, no profile is generated.

WriteBonds

Type Bool

Default value Yes

Description Write detected bonds to the .rkf file.

WriteCharges

Type Bool

Default value Yes

Description Write current atomic point charges (if available) to the .rkf file. Disable this to reduce trajectory size if you do not need to analyze charges.

WriteGradients

Type Bool

Default value No

Description Write gradients (negative of the atomic forces) to the .rkf file.

WriteMolecules

Type Bool

Default value Yes

Description Write the results of molecule analysis to the .rkf file.

WriteVelocities

Type Bool

Default value Yes

Description Write velocities to the .rkf file. Disable this to reduce trajectory size if you do not need to analyze the velocities.

Checkpoint

Type Block

Description Sets the frequency for storing the entire MD state necessary for restarting the calculation.

Frequency

Type Integer

Default value 1000

GUI name Checkpoint frequency

Description Write the MD state and engine-specific data to the respective .rkf files once every N steps.

WriteProperties

Type Bool

Default value No

Description Write the properties from the properties section to the ChecoPoint file once every N steps.

CalcPressure

Type Bool

Default value No

Description Calculate the pressure in periodic systems. This may be computationally expensive for some engines that require numerical differentiation. Some other engines can calculate the pressure for negligible additional cost and will always do so, even if this option is disabled.

Print

Type Block

Description This block controls the printing of additional information to stdout.

System

Type Bool

Default value No

Description Print the chemical system before and after the simulation.

Velocities

Type Bool

Default value No

Description Print the atomic velocities before and after the simulation.

4.9.6 Lattice deformations (volume regimes)

The `Deformation` block can be used to gradually deform the periodic lattice of the system during a MD simulation. This block can be repeated to define multiple deformations, which will be applied on every MD step in the order in which they are listed in the input.

See also:

Example demonstrating various lattice deformations (page 287)

Warning: While a `Deformation` and a `Barostat` can be used at the same time, remember to set the `Scale` parameter of the barostat so that no dimension is simultaneously being deformed and barostatted.

Each `Deformation` block will be active on MD steps between `StartStep` and `StopStep`.

The time dependence of the lattice parameters is defined by the `Type` key:

Linear This deformation type adds the same constant amount to the selected lattice parameters on every simulation step. When used with `TargetLattice`, `LatticeVelocity`, or `StrainRate`, the lattice matrix H evolves as $H(t) = H_0 + \Delta H \cdot t$, where H_0 is the lattice at `StartStep`.

Exponential When used with `StrainRate`, this type strains the lattice by the given strain matrix `\epsilon` on every step, so that the lattice matrix H evolves as $H(t) = H_0(1 + \epsilon)^t$. When used with `LengthRate`, the length of each lattice vector evolves as $l(t) = l_0(1 + r)^t$.

Sine This is a periodic deformation going from the starting value of the selected lattice parameters to a set target, and then with the same amplitude to the opposite direction from the starting lattice.

Cosine This periodic deformation oscillates between the starting lattice and a defined target.

The period of the oscillation for the **Sine** and **Cosine** types must be set using the **Period** key.

The extent of the deformation is defined by setting one of the six mutually exclusive input keys. These belong to two groups, depending on whether they operate on the lattice matrix as a whole, or just on the lengths of the individual lattice vectors:

TargetLattice, Velocity, and Rate These input keys expect a “lattice-like” matrix of numbers, consisting of up to three rows containing up to three values each. Each row contains the components of a single lattice vector and corresponds to a row of the *Lattice block* (page 30). For systems with 1D or 2D periodicity the matrix may be padded to 3x3 with zeros.

TargetLength, LengthVelocity, and LengthRate These input keys expect a list of up to three values, defining the desired length of each lattice vector or the (absolute or relative) rate of its change.

MolecularDynamics

Deformation

Type Block

Recurring True

Description Deform the periodic lattice of the system during the simulation.

LatticeVelocity

Type Non-standard block

Description Velocity of individual lattice vector components in Angstrom/fs. The format is identical to the *System%Lattice* block. For Type **Sine** and **Cosine**, this defines the maximum velocity (at the inflection point).

LengthRate

Type Float List

Default value [0.0, 0.0, 0.0]

Description Relative rate of change of each lattice vector per step.

LengthVelocity

Type Float List

Default value [0.0, 0.0, 0.0]

Unit Angstrom/fs

Description Change the length of each lattice vector with this velocity. With **Type=Exponential**, **LengthVelocity** is divided by the current lattice vector lengths on **Start-Step** to determine a **LengthRate**, which is then applied on all subsequent steps. For Type **Sine** and **Cosine**, this defines the maximum velocity (at the inflection point).

Period

Type Float

Unit Femtoseconds

Description Period of oscillation for Type **Sine** and **Cosine**.

ScaleAtoms

Type Bool

Default value Yes

Description Scale the atomic positions together with the lattice vectors. Disable this to deform only the lattice, keeping the coordinates of atoms unchanged.

StartStep

Type Integer

Default value 1

Description First step at which the deformation will be applied.

StopStep

Type Integer

Default value 0

Description Last step at which the deformation will be applied. If unset or zero, nSteps will be used instead.

StrainRate

Type Non-standard block

Description Strain rate matrix to be applied on every step. The format is identical to the System%Lattice block.

TargetLattice

Type Non-standard block

Description Target lattice vectors to be achieved by StopStep. The format is identical to the System%Lattice block.

TargetLength

Type Float List

Default value [0.0, 0.0, 0.0]

Unit Angstrom

Description Target lengths of each lattice vector to be achieved by StopStep. The number of values should equal the periodicity of the system. If a value is zero, the corresponding lattice vector will not be modified.

Type

Type Multiple Choice

Default value Linear

Options [Linear, Exponential, Sine, Cosine]

Description Function defining the time dependence of the deformed lattice parameters. Linear increments the lattice parameters by the same absolute amount every timestep. Exponential multiplies the lattice parameters by the same factor every timestep. Only StrainRate, LengthRate, and LengthVelocity are supported for Type=Exponential. Sine deforms the system from the starting lattice to TargetLattice/TargetLength and then by the same amount to the opposite direction, while Cosine deforms the system from the starting lattice to the target and back.

Example: Transition from the initial lattice to a 10 Å cube over 1000 steps:

```

MolecularDynamics
  Deformation
    StopStep 1000
    TargetLattice
      10.0  0.0  0.0
      0.0 10.0  0.0
      0.0  0.0 10.0
    End
  End
End

```

Example: Oscillate the length of the c lattice vector between the initial value and 20 \AA , leaving the a and b vectors unchanged:

```

MolecularDynamics
  Deformation
    Type Cosine
    Period 100
    TargetLength 0 0 20
  End
End

```

Example: Stretch the box in the “z” direction by a true exponential strain of 10 ppm per timestep while barostatting the remaining dimensions:

```

MolecularDynamics
  Barostat Type=MTK Pressure=1e5 Tau=1000 Scale=XY
  Deformation
    Type Exponential
    StrainRate
      0.0  0.0  0.0
      0.0  0.0  0.0
      0.0  0.0 1e-5
    End
  End
End

```

4.9.7 Molecule Gun: adding molecules during simulation

The molecule gun allows you to “shoot” (add with velocity) a molecule into the simulation box.

See also:

The [GUI tutorial](#) on the molecule gun.

Molecules can be continuously added to the simulation or only once. The initial position can be pre-set or be random within the simulation box or a part thereof. It can be defined either in the Cartesian or fractional coordinates. The initial velocity can be specified either directly (in Angstrom per femtosecond) or as translational temperature or kinetic energy. Possible applications of the molecule gun include e.g. the simulation of enforced collisions or deposition processes on surfaces.

MolecularDynamics

AddMolecules

Type Block

Recurring True

GUI name Add molecules

Description This block controls adding molecules to the system (a.k.a. the Molecule Gun). Multiple occurrences of this block are possible. By default, molecules are added at random positions in the simulation box with velocity matching the current system temperature. The initial position can be modified using one of the following keywords: Coords, CoordsBox, FractionalCoords, FractionalCoordsBox. The Coords and FractionalCoords keys can optionally be accompanied by CoordsSigma or FractionalCoordsSigma, respectively.

AtomTemperature

Type Float

Default value 0.0

Unit Kelvin

Description Add random velocity corresponding to the specified temperature to individual atoms of the molecule. This only affects rotational and internal degrees of freedom, not the net translational velocity of the inserted molecule as set by the other options.

ContactDistance

Type Float

Default value 0.0

Unit Angstrom

Description Translate the bullet along the velocity vector until it comes within ContactDistance of any other atom.

Coords

Type Float List

Unit Angstrom

Description Place molecules at or around the specified Cartesian coordinates. This setting takes precedence over other ways to specify initial coordinates of the molecule: [CoordsBox], [FractionalCoords], and [FractionalCoordsBox].

CoordsBox

Type Float List

Unit Angstrom

Description Place molecules at random locations inside the specified box in Cartesian coordinates. Coordinates of the box corners are specified as: Xmin, Xmax, Ymin, Ymax, Zmin, Zmax. This setting is ignored if Coords is used. In AMSinput, if this field is not empty it will be used instead of the default Coords.

CoordsSigma

Type Float List

Unit Angstrom

Description Sigma values (one per Cartesian axis) for a Gauss distribution of the initial coordinates. Can only be used together with Coords.

DeviationAngle

Type Float

Default value 0.0

Unit Degree

Description Randomly tilt the shooting direction up to this angle away from the VelocityDirection vector.

Energy

Type Float

Unit Hartree

Description Initial kinetic energy of the molecule in the shooting direction.

EnergySigma

Type Float

Default value 0.0

Unit Hartree

Description Sigma value for the Gauss distribution of the initial kinetic energy around the specified value. Should only be used together with Energy.

FractionalCoords

Type Float List

Description Place molecules at or around the specified fractional coordinates in the main system's lattice. For non-periodic dimensions a Cartesian value in Angstrom is expected. This setting is ignored if [Coords] or [CoordsBox] is used.

FractionalCoordsBox

Type Float List

Description Place molecules at random locations inside the box specified as fractional coordinates in the main system's lattice. Coordinates of the box corners are specified as: Xmin, Xmax, Ymin, Ymax, Zmin, Zmax. For non-periodic dimensions the Cartesian value in Angstrom is expected. This setting is ignored if [Coords], [CoordsBox], or [FractionalCoords] is used.

FractionalCoordsSigma

Type Float List

Description Sigma values (one per axis) for a Gauss distribution of the initial coordinates. For non-periodic dimensions the Cartesian value in Angstrom is expected. Can only be used together with FractionalCoords.

Frequency

Type Integer

Default value 0

Description A molecule is added every [Frequency] steps after the StartStep. There is never a molecule added at step 0.

MinDistance

Type Float

Default value 0.0

Unit Angstrom

Description Keep the minimal distance to other atoms of the system when adding the molecule.

NumAttempts

Type Integer

Default value 10

Description Try adding the molecule up to the specified number of times or until the MinDistance constraint is satisfied. If all attempts fail a message will be printed and the simulation will continue normally.

Rotate

Type Bool

Default value No

Description Rotate the molecule randomly before adding it to the system.

StartStep

Type Integer

Default value 0

Description Step number when the first molecule should be added. After that, molecules are added every Frequency steps. For example, if StartStep=99 and Frequency=100 then a molecule will be added at steps 99, 199, 299, etc. . . . No molecule will be added at step 0, so if StartStep=0 the first molecule is added at the step number equal to [Frequency].

StopStep

Type Integer

Description Do not add this molecule after the specified step.

System

Type String

Description String ID of the [System] that will be added with this 'gun'. The lattice specified with this System is ignored and the main system's lattice is used instead. AMSinput adds the system at the coordinates of the System (thus setting Coords to the center of the System).

Temperature

Type Float

Unit Kelvin

Description Initial energy of the molecule in the shooting direction will correspond to the given temperature.

TemperatureSigma

Type Float

Default value 0.0

Unit Kelvin

Description Sigma value for the Gauss distribution of the initial temperature the specified value. Should only be used together with Temperature.

Velocity

Type Float

Unit Angstrom/fs

Description Initial velocity of the molecule in the shooting direction.

VelocityDirection

Type Float List

Description Velocity direction vector for aimed shooting. It will be random if not specified. In AMSinput add one or two atoms (which may be dummies). One atom: use vector from center of the system to add to that atom. Two atoms: use vector from the first to the second atom.

VelocitySigma

Type Float

Default value 0.0

Unit Angstrom/fs

Description Sigma value for the Gauss distribution of the initial velocity around the specified value. Should only be used together with Velocity.

4.9.8 Removing molecules during simulation

Molecules *detected* (page 208) by the AMS driver can also be removed from the system. This feature can, for example, be used to remove reaction products.

MolecularDynamics

RemoveMolecules

Type Block

Recurring True

GUI name Remove molecules

Description This block controls removal of molecules from the system. Multiple occurrences of this block are possible.

Formula

Type String

Description Molecular formula of the molecules that should be removed from the system. The order of elements in the formula is very important and the correct order is: C, H, all other elements in the strictly alphabetic order. Element names are case-sensitive, spaces in the formula are not allowed. Digit '1' must be omitted. Valid formula examples: C2H6O, H2O, O2S. Invalid formula examples: C2H5OH, H2O1, OH, SO2. Invalid formulas are silently ignored. Use * to remove any molecule, which must be combined with SinkBox or SafeBox.

Frequency

Type Integer

Default value 0

Description The specified molecules are removed every so many steps after the StartStep. There is never a molecule removed at step 0.

SafeBox**Type** Block**Description** Part of the simulation box where molecules may not be removed. Only one of the SinkBox or SafeBox blocks may be present. If this block is present the molecule will not be removed if any of its atoms is within the box. For a periodic dimension it is given as a fraction of the simulation box (the full 0 to 1 range by default). For a non-periodic dimension it represents absolute Cartesian coordinates in Angstrom.**Amax****Type** Float**Description** Coordinate of the upper bound along the first axis.**Amin****Type** Float**Description** Coordinate of the lower bound along the first axis.**Bmax****Type** Float**Description** Coordinate of the upper bound along the second axis.**Bmin****Type** Float**Description** Coordinate of the lower bound along the second axis.**Cmax****Type** Float**Description** Coordinate of the upper bound along the third axis.**Cmin****Type** Float**Description** Coordinate of the lower bound along the third axis.**FractionalCoordsBox****Type** Float List**GUI name** Safe box**Description** Do not remove molecules that are (partly) inside the safe box. Borders of the safe box specified as: Amin, Amax, Bmin, Bmax, Cmin, Cmax. For periodic dimensions fractional coordinates between 0 and 1 and for non-periodic dimensions Cartesian values in Angstrom are expected.**SinkBox****Type** Block**Description** Part of the simulation box where matching molecules will be removed. By default, molecules matching the formula will be removed regardless of their location. If this block is present then such a molecule will only be removed if any of its atoms is within the box. For a periodic dimension it is given as a fraction of the simulation box (the full 0 to 1 range by default). For a non-periodic dimension it represents absolute Cartesian coordinates in Angstrom.

Amax

Type Float

Description Coordinate of the upper bound along the first axis.

Amin

Type Float

Description Coordinate of the lower bound along the first axis.

Bmax

Type Float

Description Coordinate of the upper bound along the second axis.

Bmin

Type Float

Description Coordinate of the lower bound along the second axis.

Cmax

Type Float

Description Coordinate of the upper bound along the third axis.

Cmin

Type Float

Description Coordinate of the lower bound along the third axis.

FractionalCoordsBox

Type Float List

GUI name Sink box

Description Remove molecules that are (partly) inside the sink box. Borders of the sink box specified as: Amin, Amax, Bmin, Bmax, Cmin, Cmax. For periodic dimensions fractional coordinates between 0 and 1 and for non-periodic dimensions Cartesian values in Angstrom are expected.

StartStep

Type Integer

Default value 0

Description Step number when molecules are removed for the first time. After that, molecules are removed every [Frequency] steps. For example, if StartStep=99 and Frequency=100 then molecules will be removed at steps 99, 199, 299, etc... No molecule will be removed at step 0, so if StartStep=0 the first molecules are removed at the step number equal to [Frequency].

StopStep

Type Integer

Description Do not remove the specified molecules after this step.

Warning: When there is a `Molecules%AdsorptionSupportRegion` defined, the molecule formulas depend on whether the molecule is adsorbed or not.

4.9.9 Accelerated dynamics

The PLUMED library support in AMS

PLUMED (<http://www.plumed.org/>) is a plugin that works with various MD programs and is also available in AMS. It can be used for on-the-fly analysis of the dynamics, or to perform a wide variety of free energy methods. The interface with the plugin is really simple: you just need to specify the PLUMED input in the `MolecularDynamics%Plumed%Input` block and it will be passed to the library “as is”. At each MD step, the current state of the system will be passed to the plugin to be updated according to the PLUMED input.

MolecularDynamics

Plumed

Type Block

Description Input for PLUMED. The parallel option is still experimental.

Input

Type Non-standard block

Description Input for PLUMED. Contents of this block is passed to PLUMED as is.

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

Metadynamics for Conformer-Rotamer Ensemble Sampling (CREST-MTD)

This is a very specific implementation of metadynamics that is only meant for exploration of conformer space, as used in the Conformer-Rotamer Ensemble Sampling (CREST) approach. It is an RMSD-based metadynamics that places a new 1-dimensional Gaussian on the potential energy surface every `NSteps` steps. The Gaussian is a function of the RMSD from the current geometry, which will always be zero at the moment of placement. All Gaussians have the same maximum height (`Height`) and the same width (`Width`). There is an upper limit of `NGaussiansMax` to the number of Gaussians present in the system, and when this is exceeded the oldest Gaussian is removed. By default the Gaussians are gradually introduced, using a scaling function that increases from 0 to 1 with the simulation time. The keyword `ScalingSlope` in the inputblock `GaussianScaling` determines the slope of the scaling function with respect to time. The default value of 0.03 steps^{-1} yields a scaling factor of nearly 1 after 100 steps. The keyword `ScaleGaussians` in the inputblock `GaussianScaling` determines whether the Gaussians are scaled at all.

MolecularDynamics

CRESTMTD

Type Block

GUI name CREST_MTD

Description Input for CREST metadynamics simulation.

AddEnergy

Type Bool

Default value No

Description Add the bias energy to the potential energy (to match the gradients)

GaussianScaling

Type Block

Description Options for gradual introduction of the Gaussians

ScaleGaussians

Type Bool

Default value Yes

Description Introduce the Gaussians gradually, using a scaling function

ScalingSlope

Type Float

Default value 0.03

Description Slope of the scaling function for the Gaussians with respect to time

Height

Type Float

Unit Hartree

Description The height of the Gaussians added

NGaussiansMax

Type Integer

Description Maximum number of Gaussians stored

NSteps

Type Integer

Description Interval of Gaussian placement

RestartFile

Type String

Description Filename for file from which to read data on Gaussians placed previously.

Width

Type Float

Unit Bohr

Description The width of the Gaussians added in terms of the RMSD

By default the gradients of the Gaussians with respect to the atom coordinates are added to the state gradients, but the value of the Gaussians is not added to the energy, which is common in metadynamics. For testing purposes it can be useful to add the Gaussian value to the energy, and this can be done with the keyword `AddEnergy`. Irrespective of this choice, the energy value of the Gaussian at each geometry is printed in the `ams.rkf` file in the section `CrestMTDHistory`. It is also possible to use Gaussians from an earlier CREST-MTD simulation, using the keyword `RestartFile`.

Collective Variable-driven HyperDynamics (CVHD)

The Collective Variable-driven HyperDynamics is a molecular dynamics acceleration method that allows observation of rare events by filling energy minima with a bias potential. In this sense it is similar to metadynamics. The difference of the hyperdynamics is that it ensures that the bias disappears in the transition state region. This difference allows hyperdynamics to calculate the rate of slow processes, for example the ignition phase of combustion.

See also:

The [GUI tutorial](#) on CVHD.

The CVHD implementation in AMS follows the algorithm described in [K.M. Bal, E.C. Neyts, JCTC, 11 \(2015\) \(https://doi.org/10.1021/acs.jctc.5b00597\)](#)

The `StartStep`, `Frequency`, `StopStep`, and `WaitSteps` keys define when and how often the bias potential is added, and when it is removed. The `Bias` block defines parameters of the bias potential peaks and the `ColVarBB` block describes parameters of the bond-breaking collective variable.

MolecularDynamics

CVHD

Type Block

Recurring True

GUI name CVHD

Description Input for the Collective Variable-driven HyperDynamics (CVHD).

Bias

Type Block

Description The bias is built from a series of Gaussian peaks deposited on the collective variable axis every [Frequency] steps during MD. Each peak is characterized by its (possibly damped) height and the RMS width (standard deviation).

DampingTemp

Type Float

Default value 0.0

Unit Kelvin

GUI name Bias damping T

Description During well-tempered hyperdynamics the height of the added bias is scaled down with an $\exp(-E/kT)$ factor [PhysRevLett 100, 020603 (2008)], where E is the current value of the bias at the given CV value and T is the damping temperature DampingTemp. If DampingTemp is zero then no damping is applied.

Delta

Type Float

Description Standard deviation parameter of the Gaussian bias peak.

Height

Type Float

Unit Hartree

Description Height of the Gaussian bias peak.

ColVarBB

Type Block

Recurring True

GUI name Collective Variable

Description Description of a bond-breaking collective variable (CV) as described in [Bal & Neyts, JCTC, 11 (2015)]. A collective variable may consist of multiple ColVar blocks.

at1

Type Block

Description Specifies the first bonded atom in the collective variable.

region

Type String

Default value *

Description Restrict the selection of bonded atoms to a specific region. If this is not set, atoms anywhere in the system will be selected.

symbol

Type String

Description Atom type name of the first atom of the bond. The name must be as it appears in the System block. That is, if the atom name contains an extension (e.g C.1) then the full name including the extension must be used here.

at2

Type Block

Description Specifies the second bonded atom in the collective variable.

region

Type String

Default value *

Description Restrict the selection of bonded atoms to a specific region. If this is not set, atoms anywhere in the system will be selected.

symbol

Type String

Description Atom type name of the second atom of the bond. The value is allowed to be the same as [at1], in which case bonds between atoms of the same type will be included.

cutoff

Type Float

Default value 0.3

GUI name Bond order cutoff

Description Bond order cutoff. Bonds with BO below this value are ignored when creating the initial bond list for the CV. The bond list does not change during lifetime of the variable even if some bond orders drop below the cutoff.

p

Type Integer

Default value 6

GUI name Exponent p

Description Exponent value p used to calculate the p-norm for this CV.

rmax

Type Float

Unit Angstrom

GUI name R max

Description Max bond distance parameter Rmax used for calculating the CV. It should be close to the transition-state distance for the corresponding bond.

rmin

Type Float

Unit Angstrom

GUI name R min

Description Min bond distance parameter Rmin used for calculating the CV. It should be close to equilibrium distance for the corresponding bond.

Frequency

Type Integer

Description Frequency of adding a new bias peak, in steps. New bias is deposited every [Frequency] steps after [StartStep] if the following conditions are satisfied: the current CV value is less than 0.9 (to avoid creating barriers at the transition state), the step number is greater than or equal to [StartStep], and the step number is less than or equal to [StopStep].

StartStep

Type Integer

Description If this key is specified, the first bias will be deposited at this step. Otherwise, the first bias peak is added at the step number equal to the Frequency parameter. The bias is never deposited at step 0.

StopStep

Type Integer

Description No bias will be deposited after the specified step. The already deposited bias will continue to be applied until the reaction event occurs. After that no new CVHD will be started. By default, the CVHD runs for the whole duration of the MD calculation.

WaitSteps

Type Integer

Description If the CV value becomes equal to 1 and remains at this value for this many steps then the reaction event is considered having taken place. After this, the collective variable will be reset and the bias will be removed.

During a CVHD calculation, the following variables are saved to the MDHistory section of the RKF file, in addition to other MD properties:

- *BiasEnergy* - value the bias energy at the current MD step, in Hartree.
- *MaxBiasEnergy* - max BiasEnergy since the last sampling step.
- *BoostFactor* - the boost factor at the given MD step. The boost factor is calculated at each MD step as $boost = e^{E_{bias}/kT}$, where T is the MD ensemble temperature.
- *MaxBoostFactor* - max BoostFactor value since the last sampling step.
- *HyperTime* - boosted MD time, in femtoseconds, which is a sum of the hyper-time steps calculated from the current boost factor and the MD time step as $\Delta t_{boost} = boost * \Delta t$. In hyperdynamics, the hyper-time value is directly related to the rate of the process boosted by the corresponding collective variable.

Temperature Replica Exchange

Sampling of rare events can be accelerated using the Replica Exchange Molecular Dynamics (REMD) method, also known as Parallel Tempering. This method runs multiple replicas (copies) of the simulated system in parallel, each in a different ensemble. In the case of Temperature REMD, these ensembles are all NVT or NpT, each at a different temperature. Periodically, Monte Carlo swaps are attempted between neighboring ensembles. If the current configuration of replica A has a sufficient Boltzmann probability in the ensemble of replica B (and vice versa), the two configurations will be swapped. This causes high-energy configuration to migrate into the high-temperature replicas while low-energy configurations eventually end up in the coldest ensemble. This facilitates the crossing of energy barriers in the high-temperature ensembles while keeping the coldest replica at a given temperature of interest. Because each replica always samples an unbiased ensemble, any property can be calculated using standard MD analysis methods without special preparation.

The method is controlled using the ReplicaExchange block:

MolecularDynamics

ReplicaExchange

Type Block

Description This block is used for (temperature) Replica Exchange MD (Parallel Tempering) simulations.

AllowWrongResults

Type Bool

Default value No

Description Allow combining Replica Exchange with other features when the combination is known to produce physically incorrect results.

EWMALength

Type Integer

Default value 10

Description Length of the exponentially weighted moving average used to smooth swap probabilities for monitoring. This value is equal to the inverse of the EWMA mixing factor.

SwapFrequency

Type Integer

Default value 100

Description Attempt an exchange every N steps.

TemperatureFactors

Type Float List

Description This is the ratio of the temperatures of two successive replicas. The first value sets the temperature of the second replica with respect to the first replica, the second value sets the temperature of the third replica with respect to the second one, and so on. If there are fewer values than nReplicas, the last value of TemperatureFactor is used for all the remaining replicas.

Temperatures

Type Float List

Description List of temperatures for all replicas except for the first one. This is mutually exclusive with TemperatureFactors. Exactly nReplicas-1 temperature values need to be specified, in increasing order. The temperature of the first replica is given by [Thermostat%Temperature].

nReplicas

Type Integer

Default value 1

GUI name Number of replicas

Description Number of replicas to run in parallel.

The number of replicas set by nReplicas must never exceed the total number of processors used for the simulation. If possible, the total number of processors should be an integer multiple of nReplicas to ensure good load balancing.

The temperature of the base (coldest) replica is determined by the Thermostat input block, just like in an ordinary MD simulation. There are two ways to set the temperatures of the remaining replicas, either using Temperatures or TemperatureFactors. The latter is typically more convenient, as it makes it easy to set up the optimal geometric progression of temperatures. In the simplest case, it is enough to supply just a single value in TemperatureFactors, setting the common ratio of temperatures of any two adjacent replicas.

SwapFrequency should be set as low as practical for maximum efficiency. The value of this parameter isn't critical because it doesn't affect the validity of the results. However, setting it too high will decrease overall acceleration by missing some opportunities to exchange. Conversely, using a value that is too low will increase the communication

overhead and lead to useless back-and-forth swaps between adjacent replicas. Ideally, `SwapFrequency` should be comparable to the correlation time of the system to ensure that individual exchange attempts are uncorrelated.

The trajectory of each replica is written to a separate RKF file: *ams.rkf* for the base replica and *replicaX.rkf* for the other replicas. One can easily switch between these files in the GUI using *File* → *Related Files*. In addition to data present in any MD trajectory, these files also contain an extra section *ReplicaExchangeHistory* with the following data items written every `SwapFrequency` steps:

- *AvgSwapProbability* – Average swap acceptance for each pair of replicas, smoothed using an exponentially weighted moving average with a mixing factor equal to the inverse of `EWMALength`.
- *{Min,Max,Mean,StdDev}PotentialEnergy* – Statistics of the potential energy for each ensemble over the last `SwapFrequency` steps.
- *SystemInEnsemble i* – Identifies the system (continuous trajectory) currently running in ensemble (replica) *i*.
- *EnsembleOfSystem i* – Inverse mapping of *SystemInEnsemble*, giving the current replica number in which the system number *i* runs.
- *TemperatureOfSystem i* – Equivalent to *EnsembleOfSystem* using temperatures instead of integer numbers to identify ensembles.

These data items can be plotted using the MD *Replica Exchange* menu in AMSmovie. For example, plotting *TemperatureOfSystem* or *EnsembleOfSystem* is useful to visualize the migration of each system through the space of ensembles, where each curve represents one continuous trajectory. Plotting potential energy statistics or average acceptances facilitates tuning the number of replicas and their temperatures to achieve efficient acceleration. The replica exchange method can only work when the potential energy distributions of adjacent ensembles have a sufficient overlap. This can be easily seen by comparing *MaxPotentialEnergy* of ensemble *i* with *MinPotentialEnergy* of ensemble *i+1*. The optimal degree of overlap is such that leads to approximately 20 % of swap attempts getting accepted. The acceptance of swaps can be monitored by plotting *AvgSwapProbability* and the corresponding `TemperatureFactors` can then be adjusted to keep it near the optimal value.

Bond Boost Method

The bond boost method implemented here is described in³. In this method, the distances between atoms that are relevant to the reaction of interest are calculated to determine the orientation of the reactant molecules. If a suitable initial configuration is recognized, an additional restraint energy (possibly consisting of more than one term) is added to the system that is intended to stretch or compress bonds at a pre-defined rate such that this additional energy can help achieve the energy to cross the reaction barrier. A single term of the restraint energy is depends on the restraints type and its parameters, see *restraint definitions* (page 223) for details. If more than one suitable configuration is found then the one with the smallest sum of distances is used to create the restraints.

See also:

The [GUI tutorial](#) on the Bond Boost Method.

MolecularDynamics

BondBoost

Type Block

Recurring True

Description Forced reaction (bond boost) definitions. Multiple BondBoost blocks may be specified, which will be treated independently.

Chain

³ A. Vashisth, C. Ashraf, W. Zhang, C.E. Bakis, and A.C.T. van Duin, *Accelerated ReaxFF simulations for describing the reactive cross-linking of polymers*, J. Phys. Chem. A, 122, 6633-6642 (2018) (<https://doi.org/10.1021/acs.jpca.8b03826>)

Type Block

Description Specifications of a chain of atoms. When a chain is detected the distance restraints will be activated. No other chain of this type will be detected while any restraints for this chain is active.

AtomNames**Type** String

Description Atom names specifying the chain. An atom name can optionally be followed by '@' and a region name, in this case only atoms of this type from the given region will be matched. A leading '@' followed by a number indicates that this position in the chain must be occupied by the atom found earlier at the specified position in the chain. For example "O H N C @1" indicates that the last atom in the chain of the five atoms must be the first oxygen, thus defining a 4-membered ring. This is the only way to define a ring because implicit rings will not be detected. For example, "O H N C O" does not include rings.

MaxDistances**Type** Float List**Unit** Angstrom

Description Maximum distances for each pair of atoms in the chain. The number of distances must be one less than the number of AtomNames.

MinDistances**Type** Float List**Unit** Angstrom

Description Minimum distances for each pair of atoms in the chain. The number of distances must be one less than the number of AtomNames.

DistanceRestraint**Type** String**Recurring** True

Description Specify two atom indices followed by the distance in Angstrom, the ForceConstant (in a.u.) and, optionally, the profile type and F(Inf) (in a.u.). This restraint will try to keep the distance between the two specified atoms at the given value. For periodic systems this restraint follows the minimum image convention. Each index indicates position of the corresponding atom in the AtomNames key. Currently recognized restraint profile types: Harmonic (default), Hyperbolic, Erf.

NSteps**Type** Integer**GUI name** Boost lifetime

Description Number of steps the restraints will remain active until removed. Atoms participating in one reaction are not available for the given number of steps.

NumInstances**Type** Integer**Default value** 1**GUI name** Number of instances

Description Number of reactions of this type taking place simultaneously.

For example:

```
MolecularDynamics
  BondBoost
    NSteps 10000
    Chain
      AtomNames      N.1  C.t  O    H    @1
      MinDistances   3.0  1.2  1.5  0.8
      MaxDistances   3.8  3.0  2.5  1.5
    End
#   Restrains      i  j  R_O  FC
DistanceRestraint  1  2  1.5  0.05  Hyperbolic  0.7
DistanceRestraint  2  3  3.5  0.03  Hyperbolic  0.5
DistanceRestraint  3  4  1.0  0.03  Hyperbolic  0.5
  End
End
```

The `AtomNames`, `MinDistances`, and `MaxDistances` keys constitute the atom chain definitions for the initial configuration. Thus, the example above defines a chain of atoms N.1-C.t-O-H-N.1 with R(N.1-C) in the (3.0,3.8) range, R(C.t-O) in the (1.2,3.0) range, etc.. In this example, the last atoms in the chain is required to be the same as the first one, thus defining a ring. The specified restraints will push pairs of atoms C-N and O-H close together, which will hopefully let them form a bond, and pull atoms C.t and O away from each other, thus breaking the C-O bond. The restraint type is set to `Hyperbolic` to avoid very large forces that would otherwise result from a harmonic potential at a large deviation.

Note: When detecting coordinates, the program uses the full atom name and not just the element name. An atom name consists of the element name optionally followed by a period and a suffix, just like N.1 and C.t in the example above. Using extended names for some atoms one may allow only a subset of bonds to be boosted.

Force bias Monte Carlo (fbMC)

Configurational sampling and structural relaxation can be accelerated by the Force bias Monte Carlo method. Unlike molecular dynamics, this approach does not use the equations of motion or atomic velocities. Instead, all atoms are randomly displaced in each fbMC steps. AMS uses an uniform-acceptance force bias Monte Carlo flavour of fbMC called tfMC, as published by Mees et al.⁴ Unlike other well-known Monte Carlo methods such as the Metropolis algorithm, fbMC does not generate completely random moves and then test whether these should be accepted or rejected. Instead, the atomic displacements are generated so that atoms preferably move in the direction of the forces acting on them. This special biasing enables fbMC to accept every move, sampling the correct canonical ensemble with high efficiency.

In AMS the fbMC method is implemented as an add-on to molecular dynamics, enabling mixed MC-MD simulations. After every `Frequency` MD steps, the MD procedure is stopped and fbMC takes over, executing `NSteps` Monte Carlo steps. Once the fbMC procedure completes, the MD simulation continues from the new geometry generated by fbMC while reusing atomic velocities from before the start of fbMC.

The fbMC module requires three main input settings. Apart from the `NSteps` and `Frequency` defining the mix of molecular dynamics and Monte Carlo, `Temperature` needs to be set as desired to sample a particular canonical (NVT) ensemble. Although the fbMC procedure always runs in constant temperature mode, the rest of the MD simulation can in principle use any ensemble (with or without a thermostat or barostat).

⁴ M.J. Mees, G. Pourtois, E. C. Neyts, B. J. Thijsse, and André Stesmans, *Uniform-acceptance force-bias Monte Carlo method with time scale to study solid-state diffusion*, *Phys. Rev. B* 85, 134301 (2012) (<https://doi.org/10.1103/PhysRevB.85.134301>)

Note: To run a pure fbMC simulation with no molecular dynamics component, leave `Frequency` set to 1 (the default), set the `MD TimeStep` and `InitialVelocities` to zero, and do not use other MD features such as a thermostat or a barostat.

Note: AMS currently cannot write sampled geometries to the trajectory during the fbMC procedure. To see the evolution of the system during a long fbMC run, split it into multiple shorter fbMC passes to let the MD driver sample the current state between them. To do this, set the `NSteps` for fbMC to the desired trajectory sampling interval, keep `Frequency` set to 1 and set `Trajectory SamplingFreq` to 1.

The amount of acceleration provided by fbMC depends on the `StepLength` parameter, which defines the maximum displacement of the lightest atom in the system. The default value of 0.1 Å is relatively conservative. Increasing `StepLength` leads to faster evolution of the system, but setting it too high can lead to inaccurate sampling or cause crashes by generating geometries that are too distorted.

An estimate of the timescale associated with fbMC (tfMC) steps will be printed to the output.

MolecularDynamics

fbMC

Type Block

Recurring True

GUI name fbMC

Description This block sets up force bias Monte Carlo interleaved with the molecular dynamics simulation.

Frequency

Type Integer

Default value 1

Description Run the fbMC procedure every `Frequency` MD steps.

MassRoot

Type Float

Default value 2.0

Description Inverse of the exponent used to mass-weight fbMC steps.

NSteps

Type Integer

GUI name Number of steps

Description Number of fbMC steps to perform on every invocation of the procedure.

PrintFreq

Type Integer

GUI name Printing frequency

Description Print current thermodynamic properties to the output every `N` fbMC steps. This defaults to the `PrintFreq` set in the `Trajectory` block. Setting this to zero disables printing fbMC steps.

StartStep**Type** Integer**Default value** 1**Description** First step at which the fbMC procedure may run.**StepLength****Type** Float**Default value** 0.1**Unit** Angstrom**Description** Maximum allowed displacement of the lightest atom in the system in each Cartesian coordinate in one fbMC step.**StopStep****Type** Integer**Default value** 0**Description** Last step at which the fbMC procedure may run. If unset or zero, there is no limit.**Temperature****Type** Float**Unit** Kelvin**Description** Temperature used for fbMC.

4.9.10 Non-equilibrium MD (NEMD)

T-NEMD for thermoconductivity: heat exchange

There are different methods to study thermal conductivity using non-equilibrium molecular dynamics (NEMD). A common feature of these methods is that they require the system to be divided into three or more zones, each with its own thermostat and other properties. One method maintains the temperature of the heat source and the heat sink zones at the given temperature using two different thermostats and measures the amount of heat transferred. This method does not require any special features besides a standard thermostat and a possibility to calculate the amount of heat accumulated by the thermostat per unit of time. The accumulated thermostat energies are available in the MDHistory section of ams.rkf file, in variables called 'XXXXstat#Energy', where XXXX is a 4-letter abbreviation of the thermo-/barostat ('BerT' for a Berendset thermostat, 'NHCT' for an NHC thermostat, 'NHTB' for an MTK barostat, etc.) and '#' is a 1-digit index of the thermo-/barostat.

In the other method, the heat flow is constant and the induced temperature gradient is measured. This method is implemented in AMS in three variants: a simple heat exchange, HEX¹ and eHEX². In the simple heat exchange method the atom velocities are scaled up (or down) by a factor corresponding to the amount of heat deposited to the heat source (or withdrawn from the heat sink) without paying attention to the conservation of the total momentum of the heat source (or sink). In the HEX method the velocities are scaled in such a way that the total momentum is conserved. This, however, introduces a small but measurable drift in the total energy. The eHEX algorithm improves upon the HEX by adding a third-order time-integration correction to remove the drift.

¹ T. Ikeshoji and B. Hafskjold, *Non-equilibrium molecular dynamics calculation of heat conduction in liquid and through liquid-gas interface* Molecular Physics 81, 251-261 (1994) (<https://doi.org/10.1080/00268979400100171>)

² P. Wirnsberger, D. Frenkel, and C. Dellago, *An enhanced version of the heat exchange algorithm with excellent energy conservation properties* Journal of Chemical Physics 143, 124104 (2015) (<http://dx.doi.org/10.1063/1.4931597>)

This method is controlled by the HeatExchange sub-block of the MolecularDynamics block:

MolecularDynamics

HeatExchange

Type Block

Recurring True

GUI name Heat exchange

Description Input for the heat-exchange non-equilibrium MD (T-NEMD).

HeatingRate

Type Float

Unit Hartree/fs

Description Rate at which the energy is added to the Source and removed from the Sink. A heating rate of 1 Hartree/fs equals to about 0.00436 Watt of power being transferred through the system.

Method

Type Multiple Choice

Default value Simple

Options [Simple, HEX, eHEX]

Description Heat exchange method used. Simple: kinetic energy of the atoms of the source and sink regions is modified irrespective of that of the center of mass (CoM) of the region (recommended for solids). HEX: kinetic energy of the atoms of these regions is modified keeping that of the corresponding CoM constant. eHEX: an enhanced version of HEX that conserves the total energy better (recommended for gases and liquids).

Sink

Type Block

Description Defines the heat sink region (where the heat will be removed).

AtomList

Type Integer List

GUI name Sink region

Description The atoms that are part of the sink. This key is ignored if the [Box] block or [Region] key is present.

Box

Type Block

Description Part of the simulation box (in fractional cell coordinates) defining the heat sink. If this block is specified, then by default, the whole box in each of the three dimensions is used, which usually does not make much sense. Normally, you will want to set the bounds along one of the axes.

Amax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the first axis.

Amin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the first axis.

Bmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the second axis.

Bmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the second axis.

Cmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the third axis.

Cmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the third axis.

Region

Type String

GUI name Sink region

Description The region that is the sink. This key is ignored if the [Box] block is present.

Source

Type Block

Description Defines the heat source region (where the heat will be added).

AtomList

Type Integer List

GUI name Source region

Description The atoms that are part of the source. This key is ignored if the [Box] block or [Region] key is present.

Box

Type Block

Description Part of the simulation box (in fractional cell coordinates) defining the heat source. If this block is specified, then by default, the whole box in each of the three dimensions is used, which usually does not make much sense. Normally, you will want to set the bounds along one of the axes. This block is mutually exclusive with the FirstAtom/LastAtom setting.

Amax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the first axis.

Amin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the first axis.

Bmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the second axis.

Bmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the second axis.

Cmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the third axis.

Cmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the third axis.

Region

Type String

GUI name Source region

Description The region that is the source. This key is ignored if the [Box] block is present.

StartStep

Type Integer

Default value 0

Description Index of the MD step at which the heat exchange will start.

StopStep**Type** Integer**Description** Index of the MD step at which the heat exchange will stop.

One should be careful when choosing a value for the HeatingRate because a too large value may lead to pyrolysis of the heat source or to an abnormal termination when all the kinetic energy of the heat sink has been drained. The optimal value depends on the size of the system, its heat conductivity and the desired temperature gradient value. The thermal conductivity k can be calculated by dividing the heat flow rate W by the temperature gradient $grad(T)$ and by the flow cross-section area S : $k = W / (S \cdot grad(T))$. See the *trajectory sampling* (page 89) section above on how to obtain the temperature profile from which the $grad(T)$ can be computed.

4.10 Grand Canonical Monte Carlo (GCMC)

Tip: Take a look at the [GCMC tutorial](#) and learn how to setup a GCMC calculation.

4.10.1 General info

About Monte Carlo / the Grand Canonical Ensemble

It is best to read a bit about Monte Carlo and ensembles before working with the GCMC code. Almost every book or review text on molecular simulations will do, for example: Frenkel D, Smit B. Understanding molecular simulation: from algorithms to applications. Academic Press; 2002. 672 p.

Wikipedia also has some pages of interest:

- http://en.wikipedia.org/wiki/Monte_Carlo_method
- http://en.wikipedia.org/wiki/Grand_canonical_ensemble

It is important to note that this method heavily relies on random numbers, and simulations are thus non-repeatable in detail, but should converge to the same answer.

About the AMS GCMC code

The GCMC code was originally developed for standalone Reaxff by Thomas Senftle, working as a Graduate Student at Penn State University under the supervision of Dr. Adri van Duin¹². The original version was a wrapper code that called an external executable to perform the reaxff minimization step and energy calculation, and relied on file modification and parsing to steer the reaxff code and get the results back.

The code was later rewritten by Hans van Schoot (SCM), in close collaboration with Thomas Senftle, to integrate it directly into the ADF-ReaxFF code. The current version is an AMS re-implementation so the method can be used with almost any engine supported by AMS (support for 3D periodic boundary conditions by the engine is currently a requirement).

4.10.2 Method Details

Overview

¹ T.P. Senftle, R.J. Meyer, M.J. Janik, A.C.T. van Duin, *Development of a ReaxFF potential for Pd/O and application to palladium oxide formation*, J. Chem. Phys. 139, 044109 (2013) (<https://doi.org/10.1063/1.4815820>)

² T.P. Senftle, A.C.T. van Duin, M.J. Janik, *Determining in situ phases of a nanoparticle catalyst via grand canonical Monte Carlo simulations with the ReaxFF potential*, Catalysis Communications 52, 72–77 (<https://doi.org/10.1016/j.catcom.2013.12.001>)

The GCMC method will perform a number of Grand Canonical Monte Carlo trial moves (set by the `Iterations` option of the GCMC input block), and accept or reject them based on the energy produced by the geometry optimization of the trial geometry for the given engine. The Monte Carlo algorithm will always accept a step if it results in a decrease of the energy, and accept steps that go up in energy with a probability. This section will give some details about how the method works.

MC Moves (Insert/Delete/Displace/ChangeVolume)

The GCMC method currently supports 4 types of MC Moves: Insert, Delete, Displace (sometimes also called Move), and ChangeVolume. The first three MC moves are always available and the ChangeVolume becomes available only ChangeVolume option is set to True. The first three move types change coordinates of atoms in the system, while the ChangeVolume move changes the lattice only.

On every MC iteration, the method first selects one of the molecules defined by the `Molecule` input blocks at random and then selects an applicable MC move type. If there are no molecules of this type in the system then no Delete or Displace is attempted. If the selected molecule has the `NoAddRemove` option set then the Insert and Delete moves will not be attempted. If no MC move is possible with the selected molecule type then another one is selected or a VolumeChange is attempted, if allowed. If no moves with any of the provided molecules is possible (i.e. if all molecules have `NoAddRemove` set to True, there is nothing to displace and the volume is fixed) then the program will stop.

The Insert and Displace MC move will rotate the molecule randomly and put it at a random position, and then check if the minimum interatomic distance between the molecule and the rest of the system is within the `MinDistance` and `MaxDistance` boundaries. If the condition is not satisfied, a new set of coordinates is generated and the check is performed again. This is repeated up to `NumAttempts` times before stopping with an error.

The volume change is controlled by the `VolumeChangeMax` keyword. This sets the volume change limit, and it should be a value between 0 and 1. The new volume will be calculated as: $V_{\text{new}} = \exp(\text{random}(-1:1) * \text{VolumeChangeMax}) * V_{\text{old}}$.

Calculating the chemical potential

The chemical potential of the molecule (or atom) reservoir is used when calculating the Boltzmann accept/reject criteria after a MC move is executed. This value can be derived from first principles using statistical mechanics, or equivalently, it can be determined from thermochemical tables available in literature sources.

For example, the proper chemical potential for a GCMC simulation in which single oxygen atoms are exchanged with a reservoir of O2 gas, should be equal to 1/2 the chemical potential of O2 at the temperature and pressure of the reservoir¹:

$$\mu_O^{MC}(T, P) = \frac{1}{2}\mu_{O_2}^{MC}(T, P) = \frac{1}{2} \left[\mu_{O_2}^{ref}(T, P_{ref}) + kT \ln \left(\frac{P}{P_{ref}} \right) - E_{O_2}^{diss} \right]$$

where the reference chemical potential $\mu_{O_2}^{ref}(T, P_{ref})$ is the experimentally determined chemical potential of O2 at T and P_{ref} , $kT \ln \left(\frac{P}{P_{ref}} \right)$ is the pressure correction to the free energy, and $E_{O_2}^{diss}$ is the dissociation energy of the O2 molecule.

Calculating energies

Because the GCMC simulation adds and deletes atoms or molecules during the runtime, it cannot directly compare the AMS energies for the MC acceptance criteria: inserting a molecule will usually lower the total energy of the system, causing the MC to always accept it, and always reject a deletion. To compensate this, the GCMC method calculates a “corrected” MC energy to compare the trial energy with, consisting of the previously accepted AMS energy and a correction depending on the move:

$$E_{old}^{MC} = E_{old}^{AMS} + \mu^{MC} \text{ for an Insert move;}$$

$$E_{old}^{MC} = E_{old}^{AMS} - \mu^{MC} \text{ for a Delete move;}$$

$$E_{old}^{MC} = E_{old}^{AMS} \text{ for a Displace move;}$$

$$E_{old}^{MC} = E_{old}^{AMS} - P(V_{\text{new}} - V_{\text{old}}) + N_{\text{inserted}} \ln \left(\frac{V_{\text{new}}^{avail}}{V_{\text{old}}^{avail}} \right) kT \text{ for a ChangeVolume move.}$$

Here, μ^{MC} is the chemical potential of the inserted/deleted molecule, P is the pressure, V is the volume, and N_{inserted} is the total number of MC molecules. The “new” and “old” subscripts refer to the current and the last accepted values. The V_{avail} values are calculated from the MC-available volume as described below.

Calculating volumes

The available volume can be calculated in a few different ways, depending on the `VolumeOption` setting:

- *Free*: volume = total volume - occupied volume - specified vacuum volume (`NonAccessibleVolume`)
- *Total*: volume = total cell volume
- *Accessible*: volume = specified accessible volume (`AccessibleVolume`)
- *FreeAccessible*: volume = specified accessible volume (`AccessibleVolume`) - occupied volume

Here, the occupied volume is calculated as a sum of volumes of atoms that do not belong to the MC part of the system (i.e. that were not inserted during calculation and are not `Removables`). The volume of an atom is calculated using the average of the covalent and the Van der Waals radii of the atom defined in the `atominfo` module used throughout AMS.

The `AccessibleVolume` and `NonAccessibleVolume` keywords can be used to get a more accurate available volume.

Acceptance criteria

An MC move is always accepted if the AMS energy is lower than the corrected MC energy of the last accepted MC move, or if the energy increase is small enough. If the new energy is higher, the code generates a random number between 0 and 1, and accepts the move if the random number is greater than:

```
prob = preFactor * exp(-Beta*deltaE)
```

The prefactor is calculated (for insert and delete moves) using the deBroglie wavelength of the inserted molecules, the number of inserted molecules and the available MC volume of the system.

4.10.3 Input

The GCMC functionality in AMS is triggered using the following Task key:

```
Task GCMC
```

```
GCMC
  AccessibleVolume float
  Box
    Amax float
    Amin float
    Bmax float
    Bmin float
    Cmax float
    Cmin float
  End
  Ensemble [Mu-VT | Mu-PT]
  Iterations integer
  MapAtomsToOriginalCell Yes/No
  MaxDistance float
  MinDistance float
  Molecule
    ChemicalPotential float
    NoAddRemove Yes/No
```

(continues on next page)

(continued from previous page)

```

    SystemName string
End
NonAccessibleVolume float
NumAttempts integer
Pressure float
Removables # Non-standard block. See details.
    ...
End
Restart string
Temperature float
UseGCPrefactor Yes/No
VolumeChangeMax float
VolumeOption [Free | Total | Accessible | FreeAccessible]
End

```

The following keys are common for all GCMC calculations and should always be specified. The ChemicalPotential value should correspond to the μ^{MC} expression above, and not to the experimental chemical potential μ^{ref} , which means it should include the (engine-dependent) free molecule's energy.

GCMC**Molecule****Type** Block**Recurring** True**GUI name** Molecules

Description This block defines the molecule (or atom) that can be inserted/moved/deleted with the MC method. The coordinates should form a reasonable structure. The MC code uses these coordinates during the insertion step by giving them a random rotation, followed by a random translation to generate a random position of the molecule inside the box. Currently, there is no check to make sure all atoms of the molecule stay inside the simulation box. The program does check that the MaxDistance/MinDistance conditions are satisfied.

ChemicalPotential**Type** Float**Unit** Hartree

Description Chemical potential of the molecule (or atom) reservoir. It is used when calculating the Boltzmann accept/reject criteria after a MC move is executed. This value can be derived from first principles using statistical mechanics, or equivalently, it can be determined from thermochemical tables available in literature sources. For example, the proper chemical potential for a GCMC simulation in which single oxygen atoms are exchanged with a reservoir of O2 gas, should equal 1/2 the chemical potential of O2 at the temperature and pressure of the reservoir: $\text{cmpot} = \mu_{\text{O}}(T,P) = 1/2 * \mu_{\text{O}_2}(T,P) = 1/2 * [\mu_{\text{ref}}(T,P_{\text{ref}}) + kT * \log(P/P_{\text{ref}}) - E_{\text{diss}}]$ where the reference chemical potential $[\mu_{\text{ref}}(T,P_{\text{ref}})]$ is the experimentally determined chemical potential of O2 at T and Pref; $kT * \log(P/P_{\text{ref}})$ is the pressure correction to the free energy, and E_{diss} is the dissociation energy of the O2 molecule.

NoAddRemove**Type** Bool**Default value** No**GUI name** Fix molecule

Description Set to True to tell the GCMC code to keep the number of molecules/atoms of this type fixed. It will thus disable Insert/Delete moves on this type, meaning it can only do a displacement move, or volume change move (for an NPT ensemble).

SystemName

Type String

GUI name Molecule

Description String ID of a named [System] to be inserted. The lattice specified with this System, if any, is ignored and the main system's lattice is used instead.

Iterations

Type Integer

GUI name Number of GCMC iterations

Description Number of GCMC moves.

Temperature

Type Float

Default value 300.0

Unit Kelvin

Description Temperature of the simulation. Increase the temperature to improve the chance of accepting steps that result in a higher energy.

The following keys are related to Insert and Displace moves.

GCMC

NumAttempts

Type Integer

Default value 1000

GUI name Max tries

Description Try inserting/moving the selected molecule up to the specified number of times or until all constraints are satisfied. If all attempts fail a message will be printed and the simulation will stop. If the MaxDistance-MinDistance interval is small this number may have to be large.

MinDistance

Type Float

Default value 0.3

Unit Angstrom

GUI name Add molecules not closer than

Description Keep the minimal distance to other atoms of the system when adding the molecule.

MaxDistance

Type Float

Default value 3.0

Unit Angstrom

GUI name Add molecules within

Description The max distance to other atoms of the system when adding the molecule.

The following keys influence computation of the acceptance probability and of the MC energy correction.

GCMC

UseGCPreFactor

Type Bool

Default value Yes

GUI name Use GC prefactor

Description Use the GC pre-exponential factor for probability.

VolumeOption

Type Multiple Choice

Default value Free

Options [Free, Total, Accessible, FreeAccessible]

GUI name Volume method

Description Specifies the method to calculate the volume used to calculate the GC pre-exponential factor and the energy correction in the Mu-PT ensemble: Free: $V = \text{totalVolume} - \text{occupiedVolume} - \text{NonAccessibleVolume}$; Total: $V = \text{totalVolume}$; Accessible: $V = \text{AccessibleVolume}$; FreeAccessible: $V = \text{AccessibleVolume} - \text{occupiedVolume}$. The AccessibleVolume and NonAccessibleVolume are specified in the input, the occupiedVolume is calculated as a sum of atomic volumes.

AccessibleVolume

Type Float

Default value 0.0

Description Volume available to GCMC, in cubic Angstroms. AccessibleVolume should be specified for “Accessible” and “FreeAccessible” [VolumeOption].

NonAccessibleVolume

Type Float

Default value 0.0

GUI name Non-accessible volume

Description Volume not available to GCMC, in cubic Angstroms. NonAccessibleVolume may be specified for the “Free” [VolumeOption] to reduce the accessible volume.

The following keys apply to the ensemble choice and options for the Mu-PT ensemble.

GCMC

Ensemble

Type Multiple Choice

Default value Mu-VT

Options [Mu-VT, Mu-PT]

Description Select the MC ensemble: Mu-VT for fixed volume or Mu-PT for variable volume. When the Mu-PT ensemble is selected the [Pressure] and [VolumeChangeMax] should also be specified.

VolumeChangeMax

Type Float

Default value 0.05

Description Fractional value by which logarithm of the volume is allowed to change at each step. The new volume is then calculated as $V_{\text{new}} = \exp(\text{random}(-1:1) * \text{VolumeChangeMax}) * V_{\text{old}}$

Pressure

Type Float

Default value 0.0

Unit Pascal

Description Pressure used to calculate the energy correction in the Mu-PT ensemble. Set it to zero for incompressible solid systems unless at very high pressures.

The GCMC code can insert multiple atom/molecule types in a single simulation, so it needs to keep track of what atom belongs to which insert. This information is automatically stored and updated when insertion/deletion/moving of atoms or molecules during the simulation, but is by default unknown for the atoms of the starting geometry. The GCMC code will therefore by default not modify the atoms in the original input in the MC trial moves. The `Restart` key and the `Removables` block are two ways to provide information about Deletable/Movable atoms/molecules in the input structure. If the `Restart` key is present the `Removables` block will be ignored.

GCMC**Restart**

Type String

Description Name of an RKF restart file. Upon restart, the information about the GCMC input parameters, the initial system (atomic coordinates, lattice, charge, etc.) and the MC molecules (both already inserted and to be inserted) are read from the restart file. The global GCMC input parameters and the MC Molecules can be modified from input. Any parameter not specified in the input will use its value from the restart file (i.e. not the default value). Molecules found in the restart file do not have to be present as named Systems in the input, however if there is a System present that matches the name of a molecule from restart then the System's geometry will replace that found in the restart file. It is also possible to specify new Molecules in the input, which will be added to the pool of the MC molecules from restart.

Removables

Type Non-standard block

Description The Removables can be used to specify a list of molecules that can be removed or moved during this GCMC calculation. Molecules are specified one per line in the format following format: `MoleculeName atom1 atom2 ...`. The `MoleculeName` must match a name specified in one of the [Molecule] blocks. The atom indices refer to the whole input System and the number of atoms must match that in the specified Molecule. A suitable Removables block is written to the standard output after each accepted MC move. If you do so then you should also replace the initial atomic coordinates with the ones found in the same file. If a [Restart] key is present then the Removables block is ignored.

An example of the Removables block:

```
Removables
  Oatom 41
  O2 44 45
  Oatom 42
  Oatom 43
End
```

This example specifies that 5 atoms belong to 4 different GCMC molecules of two different types, `Oatom` and `O2`. Thus in addition to the main input `System` there should be at least two additional `Systems` defined, one called “Oatom” (containing one atom) and the other “O2” (containing two atoms). The first one was inserted three times (atoms 41, 42, and 43) and the second one was inserted once.

Finally there are more technical keywords:

GCMC

MapAtomsToOriginalCell

Type Bool

Default value Yes

Description Keeps the atom (mostly) in the original cell by mapping them back before the geometry optimizations.

Note that the `GeometryOptimization` block is also read by the GCMC task, and the settings used for the individual optimizations. The documentation for these keywords can be found in the [Geometry Optimization](#) (page 43) section of this manual.

4.10.4 Output

In addition to the standard KF variables in the “History” section on `ams.rkf` such as “Coords” and “Energy”, the following GCMC-specific variables are also created for each accepted MC step:

- *MCMove* - integer index of the MC move.
- *MCMoveType* - string containing the type of the MC move.
- *MCMolecule* - string containing the name of the inserted/displaced/removed molecule.
- *Converged* - a Fortran logical value containing the convergence status of the given geometry.

Results of a GCMC calculation are stored in the GCMC section of the RKF file, in a number of variables. The following variables contain a summary of the MC statistics up to and including the latest step:

- *NIterMCtried* - the latest iteration number.
- *NIterMCaccept* - the number of accepted MC moves.
- *NIterMCreject* - the number of rejected MC moves.
- *NMCacceptAdd* - the number of accepted MC molecule insertions.
- *NMCacceptRemove* - the number of accepted MC molecule removals.
- *NMCacceptMove* - the number of accepted MC molecule moves.
- *NMCacceptVolume* - the number of accepted volume changes.
- *NMCrejectAdd* - the number of rejected MC molecule insertions.
- *NMCrejectRemove* - the number of rejected MC molecule removals.
- *NMCrejectMove* - the number of rejected MC molecule moves.

- *NMCrejectVolume* - the number of rejected volume changes.

The following variables (actually arrays of the size *Iterations*) in the GCMC section contain the detailed information about all MC moves in the current simulation. Only the first *NIterMCtried* elements of each array contain valid data.

- *HistoryAccepted* - MC move status value (1 - accepted, 0 - rejected, -1 - not done yet).
- *HistoryAMSEnergy* - the AMS energy (the E^{AMS} above).
- *HistoryMCEnergy* - the corrected MC energy ($E^{MC} = E^{AMS} - \Sigma\mu_i^{MC}$, where $\Sigma\mu_i^{MC}$ is the total chemical potential of all inserted molecules).
- *HistoryVolume* - the simulation box volume.
- *HistoryMoveType* - the MC move type index (0 - insert, 1 - delete, 2 - displace, 3 - change volume). The name of the move type with index *i* can be found in the *MoveType(i)* variable.
- *HistoryMoleculeType* - the inserted/deleted/displaced molecule type index. The name of the molecule type with index *i* can be found in the *MoleculeName(i)* variable.
- *HistoryMoleculeIndex* - the inserted/deleted/displaced molecule index within its type.

4.11 Automated PES Exploration

In chemistry and materials science, two types of Potential Energy Surface (PES) critical points are of particular interest: local minima and (first-order) saddle points.

The *PESExploration* task consists of a set of algorithms (which we will refer to as *jobs*) that will automatically explore the PES of a given system, looking for local minima and saddle points.

The available PES exploration jobs are:

***Process Search* (page 132)** A composite method for finding escape mechanisms from a state. This will find both local minima and saddle points.

***Basin Hopping* (page 135)** A Monte Carlo method for finding local minima.

***Saddle Search* (page 133)** A single-ended method for finding nearby saddle points.

***Landscape Refinement* (page 137)** Given a pre-calculated Energy Landscape, re-optimize local minima and saddle points using a different computational engine/settings.

***Binding Sites* (page 138)** Given a pre-calculated Energy Landscape, compute the binding sites.

The AMS driver links to the client program of the [EON software package](http://theory.cm.utexas.edu/eon/index.html) (<http://theory.cm.utexas.edu/eon/index.html>)¹ and uses its implementation of the dimer method. EON is developed by the University of Texas at Austin and the University of Iceland. See also the *Required citations* (page 319) section.

See also:

- PES Exploration GUI tutorials:
 - [Automated reaction pathway discovery for hydrohalogenation](#)
 - [Cluster Growth: Cobalt Clusters](#)
 - [Water dissociation on an oxide surface](#)

¹ Samuel T Chill, Matthew Welborn, Rye Terrell, Liang Zhang, Jean-Claude Berthet, Andreas Pedersen, Hannes Jónsson and Graeme Henkelman *EON: software for long time simulations of atomic scale systems*, *Modelling Simul. Mater. Sci. Eng.* 22 055002 (2014) (<https://doi.org/10.1088/0965-0393/22/5/055002>)

- Examples:
 - Example: *Basin Hopping for Ar 13 cluster* (page 282)
 - Example: *PES Exploration, Process Search for alanine with PLAMS* (page 283)
 - Example: *PES Exploration, Binding Sites for O on Pt 111* (page 283)

4.11.1 Overview

While many details of a PES exploration calculation depend on the specific *job* (page 126) selected, some aspects are common to all PES explorations jobs. Here we give a brief overview of a typical PES exploration, using a Process Search job as example.

A PES exploration calculation generally consists of multiple *expeditions*, each with several *explorers*.

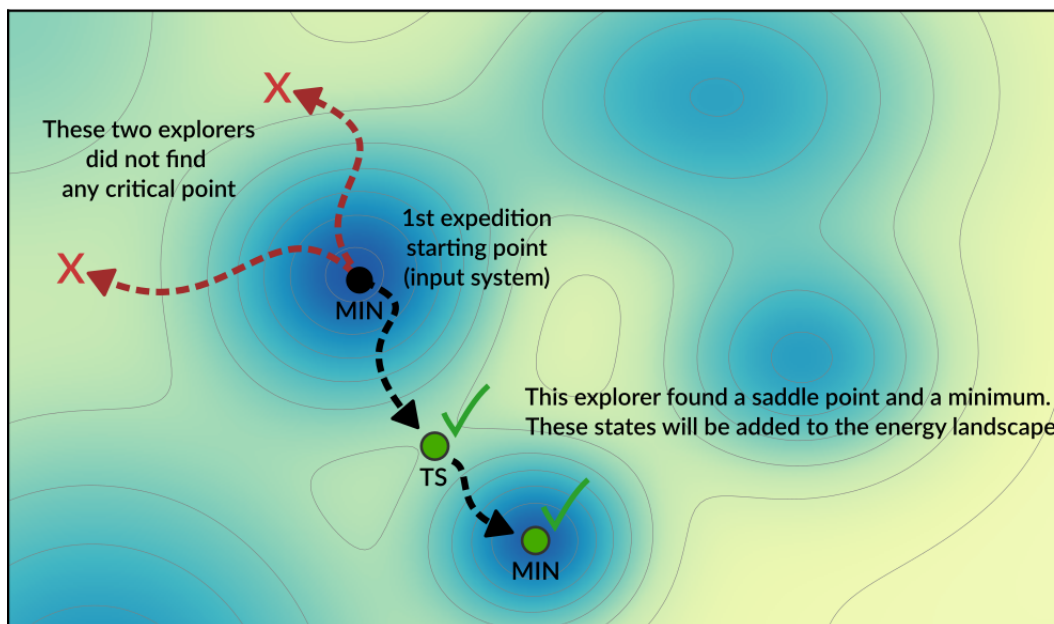
Explorers are given an initial structure, and their goal is to find a nearby critical point (a local minimum, a saddle point or both). The explorer moves around the PES by modifying the atomic positions of the system and by using the specified *Engine* (page 209) to compute energy and gradients.

An *expedition* is a collection of *explorers* all starting from the same point on the PES.

Before setting off for the first expedition, the input structure is optimized and added to the *Energy Landscape*, which is the database of all interesting points found during the exploration (see the section *Results: the “Energy Landscape”* (page 128) for more details).

Starting from this initial structure, a number of PES explorers (3 in the diagram below) will set off in random directions exploring the potential energy surface and looking for nearby critical points.

1st expedition with 3 explorers

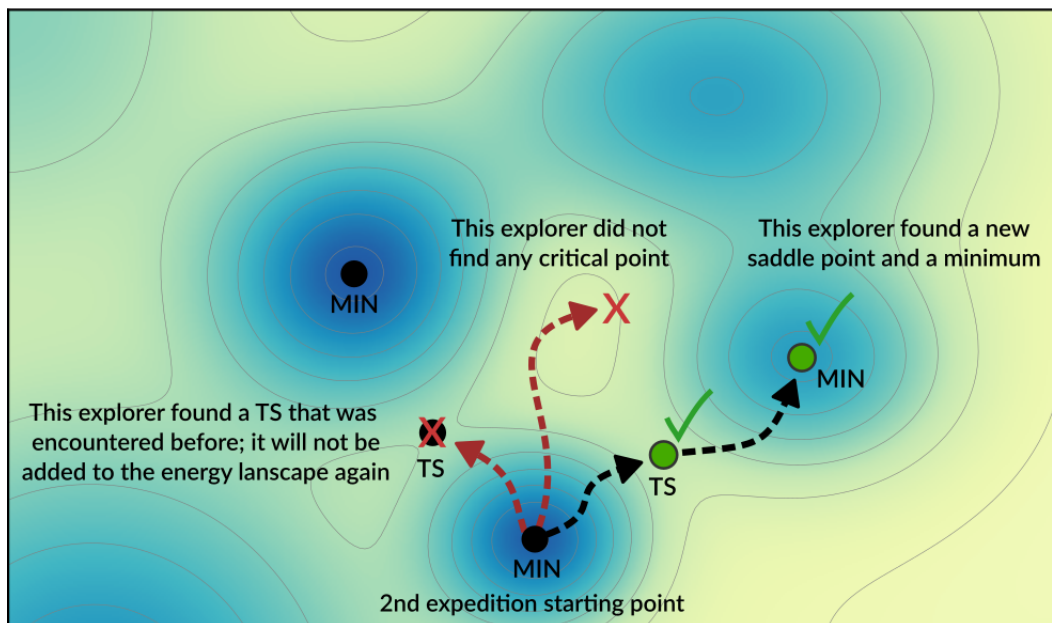


Explorers have “stop conditions” (e.g. a maximum number of steps or a maximum energy above the starting point) so in general not all explorers will successfully find a critical point. In this case, only one of the explorers found relevant critical points: a saddle point and a local minimum. These newly found critical points are added to the Energy Landscape.

After the first expedition is over, the program will start with the second expedition. The starting point for the next expedition will be a randomly chosen local minimum from the list of minima present in the Energy Landscape (the

possible starting points for an exploration are called *seed states*. See also the *DynamicSeedStates* (page 126) option. In this example, the starting point of the second expedition is the new local minimum found during the first expedition. A new set of explorers will set off in random directions from this point.

2nd expedition with 3 explorers



In the diagram above, one of the explorer found a saddle point that was already found in a previous exploration. This structure will not be added to Energy Landscape since it was already *seen* before (see the *Structure comparison* (page 130) section for more details). The newly found critical points are added to the Energy Landscape.

Usually, many expeditions and/or many explorers are needed to map the PES, but you should keep in mind that the computation time of the calculation will roughly be proportional to the product $\text{NumExpeditions} \times \text{NumExplorers}$.

By having many explorers you will have a higher chance of comprehensively mapping the PES near the starting point of each expedition. By having many expeditions, you will have a higher chance of traveling further away from the initial structure.

It should be emphasized that the PES exploration task is stochastic in nature, as it uses random numbers to perform initial-displacements. This means that if you run the same calculation twice you might find different critical points.

4.11.2 Job selection and main options

To use one of the PES Exploration procedures you should set the *Task* (page 8) to *PESExploration* and specify one of the jobs in the *PESExploration%Job* key:

```
Task PESExploration

PESExploration
  Job [ProcessSearch | BasinHopping | SaddleSearch | LandscapeRefinement | ↵
↵BindingSites]
End
```

The input options for the various jobs are described in the corresponding sections: *Process Search* (page 132), *Basin Hopping* (page 135), *Saddle Search* (page 133), *Landscape Refinement* (page 137), *Binding Sites* (page 138).

It is then important pick an appropriate number of expeditions and explorers. Having many expeditions and explorers will result in a more comprehensive PES exploration, but since the computation time will roughly be proportional to the product `NumExpeditions` x `NumExplorers` you'll need to find an appropriate balance.

```
PESExploration
  NumExpeditions integer
  NumExplorers integer
  DynamicSeedStates Yes/No
End
```

PESExploration

Type Block

Description Configures details of the automated PES exploration methods.

NumExpeditions

Type Integer

Default value 1

Description Sets the number of subsequent expeditions our job will consist of. Larger values result in a more comprehensive exploration of the potential energy surface, but will take more computational time.

NumExplorers

Type Integer

Default value 1

Description Sets the number of independent PES explorers dispatched as part of each expedition. Larger values will result in a more comprehensive exploration of the potential energy surface, but will take more computational time. By default an appropriate number of explorers are executed in parallel.

DynamicSeedStates

Type Bool

Default value Yes

Description Whether subsequent expeditions may start from states discovered by previous expeditions. This should lead to a more comprehensive exploration of the potential energy surface. Disabling this will focus the PES exploration around the initial seed states.

The following miscellaneous input option generally apply to most PES exploration calculations:

```
PESExploration
  WriteHistory [None | Converged | All]
  Temperature float
  FiniteDifference float
End
```

PESExploration

Type Block

Description Configures details of the automated PES exploration methods.

WriteHistory

Type Multiple Choice

Default value Converged

Options [None, Converged, All]

Description When to write the molecular geometry (and possibly other properties) to the history on the `ams.rkf` file. The default is to only write the converged geometries to the history. Changed to write either all frames or not frames at all to the history. Note that for parallel calculations, only the first group of processes writes to `ams.rkf`.

Temperature

Type Float

Default value 300.0

Unit Kelvin

Description The temperature that the job will run at. This may be used in different ways depending on the job, e.g. acceptance probabilities for Monte-Carlo based jobs, thermostating for dynamics based jobs, kinetic prefactors for jobs that find transition states. Some jobs may not use this temperature at all.

FiniteDifference

Type Float

Default value 0.01

Unit Angstrom

Description The finite difference distance to use for Dimer, Hessian, Lanczos, and optimization methods.

4.11.3 Results: the “Energy Landscape”

The results of a PES Exploration are the structures and energies of the critical points found. There are multiple ways for you to inspect the results:

- The energy landscape can be visualized using the `AMSMovie` GUI module (In `AMSMovie`: File → Open and select the `ams.rkf` file of your calculation). See the GUI tutorials (e.g. [Automated reaction pathway discovery for hydrohalogenation](#)) for more details.
- The results are printed to the text output under the header `Final Energy Landscape`, see *below* (page 128).
- The results are stored on the `ams.rkf` binary results file in the section `EnergyLandscape`
- In `PLAMS`, you can use the `get_energy_landscape` method of the `AMSResults` object to conveniently extract the results.

Results on the text output

These are printed at the end of the text output under the header `Final Energy Landscape`. Here is an output example for a `ProcessSearch` job for the simple HCN molecule (computed with the DFTB engine):

----- Final Energy Landscape -----					
Id	Energy (a.u.)	RE (eV)	RE (kcal/mol)	Counts	Crit. point
(continues on next page)					

(continued from previous page)

```

-----
      1      -5.782789      0.00000      0.0000      4      Min
      2      -5.748792      0.92510      21.3334      4      Min
      3      -5.689025      2.55143      58.8375      4      TS
->1 <--> 2

Number of configurations      3
Number of local minima      2
Number of transition states      1
Energy range (a.u.)      0.093763
Energy range (eV)      2.55143
Energy range (kcal/mol)      58.8375

Configurations
-----
3
Id 1 Energy(hartree)      -5.78278852 isTS=False
H      -5.05604362312999      1.04394551246415      0.36836337958238
C      -4.06812264817497      0.95005534859126      -0.01533183100739
N      -3.00601476684712      0.84907755770632      -0.42810754191928
3
Id 2 Energy(hartree)      -5.74879169 isTS=False
H      -4.01886309319822      1.09046314142766      1.01610923624838
C      -4.06910389782095      0.79751242545583      -1.12016730790896
N      -4.04221404713293      0.95510285187825      0.02898207831626
3
Id 3 Energy(hartree)      -5.68902502 isTS=True
H      -4.51488871821561      1.04261603660424      0.52096081259126
C      -4.32990923520194      0.88203060069718      -0.58636688911847
N      -3.28538308473452      0.91843178146029      -0.00966991681712

```

From this you can see that 3 critical points were found: 2 minima (Id 1 and Id 2) and 1 transition state (Id 3) connecting the first two states (indicated by 1 <--> 2).

Under `Counts` you can see how many times each state was encountered during the exploration (often the same state is found multiple times) See the *Structure comparison* (page 130) section for more details on how states are compared.

RE is the “relative energy” with respect to the lowest-energy state found.

Under the header `Configurations` you will find the XYZ geometries (in Angstrom) corresponding to the various critical points.

4.11.4 Continue a PES exploration from a previous calculation

You can load an Energy Landscape obtained from PES exploration calculation and use it as starting point for a new PES exploration. In this way you can extend your Energy Landscape, and potentially use different PES exploration algorithms.

To load a previously computed energy landscape, use the `PESExploration%LoadEnergyLandscape%Path` option (note: you should still provide an input system in the *System* (page 29) block even if you are loading a previous Energy Landscape. The input system will be optimized and added to the energy landscape as a minimum).

It is often convenient to load only some of the states from a previous calculation; this can be done via the `Remove` or `KeepOnly` input options. Specify the seed states with the `SeedStates` option.

These are all the input options related to the loading an Energy Landscape:

```

PESExploration
  LoadEnergyLandscape
    KeepOnly integer_list
    Path string
    Remove integer_list
    SeedStates integer_list
  End
End

```

PESExploration**LoadEnergyLandscape****Type** Block**Description** Options related to the loading of an Energy Landscape from a previous calculation.**KeepOnly****Type** Integer List**GUI name** List of states to keep**Description** Upon loading the Energy Landscape, only keep the states specified here. The states should be specified via a list of integers referring to the indices of the states you want to keep.**Path****Type** String**GUI name** Load energy landscape from**Description** The path to load an energy landscape from. Accepts either AMS result folders, or .con files in the native EON format (only available through the text input file).**Remove****Type** Integer List**GUI name** List of states to remove**Description** Upon loading the Energy Landscape, remove (i.e. do not load) the states specified here. The states should be specified via a list of integers referring to the indices of the states you want to remove (i.e. the states you don't want to load).**SeedStates****Type** Integer List**GUI name** List of seed states**Description** By default when you start a new PES Exploration from a loaded Energy Landscape, expeditions can start from any of the loaded minima. By using this input option, you can instruct the program to only use some of the states as 'expedition starting point'. The states that serve as 'expedition starting points' should be specified via a list of integers referring to the indices of the states.

4.11.5 Structure comparison

During a PES exploration some critical points will most likely be encountered more than once.

Whenever a critical point is found, the program will ask itself: is this a new structure, or have I already encountered this before? If the structure was not seen before (i.e. it's a novel configuration), it will be added to the list of structures found during the exploration. If the structure was already found before, the program will simply increase a counter keeping track of how many times a specific structure was found during the PES exploration.

When deciding if two structures are the same or not, the program will compare both the geometry and the energy of the two configurations. Several parameters and thresholds can be tweaked in the `PESExploration%StructureComparison` block to tell AMS under which conditions two systems should be considered equal.

```
PESExploration
  StructureComparison
    CheckRotation Yes/No
    DistanceDifference float
    EnergyDifference float
    IndistinguishableAtoms Yes/No
    NeighborCutoff float
    RemoveTranslation Yes/No
  End
End
```

PESExploration

StructureComparison

Type Block

Description Settings for structure comparison.

CheckRotation

Type Bool

Description Rotates the system optimally before comparing structures. The default is to do this only for molecular systems when there are no fixed atom constraints.

DistanceDifference

Type Float

Default value 0.1

Unit Angstrom

Description If the distance between two mapped atoms is larger than this threshold, the two configurations are considered different structures.

EnergyDifference

Type Float

Default value 0.01

Unit eV

Description If the energy difference between two configurations is larger than this threshold, the two configurations are considered to be different structures.

IndistinguishableAtoms

Type Bool

Default value Yes

Description If yes, the order of the atoms does not affect the structural comparison. Atoms of the same element are then indistinguishable.

NeighborCutoff

Type Float

Default value 3.3

Unit Angstrom

Description Atoms within this distance of each other are considered neighbors.

RemoveTranslation

Type Bool

Description Translates the system optimally before comparing structures. The default is to do this only when there are no fixed atom constraints.

4.11.6 Process Search job

Overview

Process Search is a composite method for finding escape mechanisms from a state. Each PES explorer will perform the following steps:

- Starting from a minimum, a *SaddleSearch* (page 133) calculation is performed to find a nearby saddle point
- If a saddle point is found, two images are placed on either side of the saddle along the min mode.
- The two images are then minimized; one of the images is expected to fall back into the original minimum, while the other image is expected to end up in the product state on the other side of the saddle. If this is case, the product, TS and reactant states are added to the energy landscape.

The Process Search procedure will therefore not only find minima and saddle points, but it will also keep track of which minima each saddle point is connecting.

See also:

- GUI tutorials [Automated reaction pathway discovery for hydrohalogenation](#) and [Water dissociation on an oxide surface](#).

Input options

To perform a Process Search calculation you should specify:

```
Task PESExploration
PESExploration
  Job ProcessSearch

  ProcessSearch
    # Options for the Process Search procedure. See description below.
  End

  SaddleSearch
    # Options for the Saddle Search part of the Process Search procedure.
  End
End
```

The options for the saddle search part of a Process Search job are described in the *SaddleSearch* (page 133) section.

Options specific to the Process Search procedure can be specified in the PESExploration%ProcessSearch block:

```
PESExploration
  ProcessSearch
    MinimizationOffset float
  End
End
```

PESExploration

ProcessSearch

Type Block

Description Input options specific to the process search procedure.

MinimizationOffset

Type Float

Default value 0.2

Description After a saddle is found, images are placed on either side of the saddle along the mode and minimized to ensure that the saddle is connected to the original minimum and to locate the product state. MinimizationOffset is the distance those images are displaced from the saddle.

4.11.7 Saddle Search job

Overview

Saddle Search is a single-ended method for finding nearby saddle points.

In a Saddle Search calculation, each PES explorer starts by performing a random displacement from the initial geometry (or seed state). Then, starting from this point, the explorer searches for a nearby saddle point using a minimum-mode following method (the available min-mode following methods are dimer³⁴⁵ and Lanczos⁶).

If you already know which atoms are likely to be involved in the reactions, you can use the DisplaceAtomsInRegion option to nudge the Saddle Search procedure towards the right part of the PES (*regions* (page 34) can be specified in the *system input block* (page 29), or via the “Model → Regions” panel in AMSinput).

Input options

To perform a Saddle Search calculation you should specify:

³ G. Henkelman and H. Jónsson, *A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives*, J. Chem. Phys. 111, 7010-7022 (1999) (<http://dx.doi.org/10.1063/1.480097>)

⁴ A. Heyden, A.T. Bell, and F.J. Keil *Efficient methods for finding transition states in chemical reactions: Comparison of improved dimer method and partitioned rational function optimization method*, J. Chem. Phys. 123, 224101 (2005) (<http://dx.doi.org/10.1063/1.2104507>)

⁵ J. Kästner and P. Sherwood *Superlinearly converging dimer method for transition state search*, J. Chem. Phys. 128, 014106 (2008) (<http://dx.doi.org/10.1063/1.2815812>)

⁶ R. Malek and N. Mousseau, *Dynamics of Lennard-Jones clusters: A characterization of the activation-relaxation technique*, Phys. Rev. E 62, 7723 (2000) (<http://dx.doi.org/10.1103/PhysRevE.62.7723>)

```

Task PESExploration

PESExploration
  Job SaddleSearch

  SaddleSearch
    # Options for the Saddle Search procedure. See description below.
  End
End

```

Options for the Saddle Search procedure can be specified in the `PESExploration%SaddleSearch` block.

Note: these options also affect the Saddle Search part of *Process Search Jobs* (page 132).

```

PESExploration
  SaddleSearch
    DisplaceAtomsInRegion string
    DisplaceMagnitude float
    MaxEnergy float
    MaxIterations integer
    MinModeMethod [dimer | lanczos]
  End
End

```

PESExploration

SaddleSearch

Type Block

Description Configuration for the Saddle Search procedure (used in SaddleSearch and ProcessSearch Jobs).

DisplaceAtomsInRegion

Type String

Default value

Description A string corresponding to the name of a region. When performing the initial random displacement, only displace atoms in the specified region. This can help the Saddle Search procedure to start off in a desired region of the PES.

DisplaceMagnitude

Type Float

Default value 0.1

Description The standard deviation of the Gaussian displacement in each degree of freedom for the selected atoms.

MaxEnergy

Type Float

Default value 20.0

Unit eV

Description The energy (relative to the starting point of the saddle search) at which a saddle search explorer considers the search bad and terminates it.

MaxIterations

Type Integer

Default value 400

Description Maximum number of iterations for each saddle search run.

MinModeMethod

Type Multiple Choice

Default value dimer

Options [dimer, lanczos]

Description The minimum-mode following method to use.

4.11.8 Basin Hopping job

Overview

Basin hopping² is an iterative Monte Carlo method for PES exploration and global minimization.

An “iteration” in basin hopping consists of the following steps:

1. The atomic coordinates of the system are randomly perturbed (the details of this random perturbation can be configured via the keywords `Displacement`, `DisplacementDistribution`, `SingleAtomDisplace`, `SwapProbability`. See *below* (page 136) for more details).
2. A local optimization of the randomly perturbed structure is performed (see also the *Optimizer* (page 137) section)
3. The new optimized structure is accepted or rejected based of its energy (if the energy of the newly optimized structure lower than the lowest energy so far, the new structure is readily accepted. Otherwise it is accepted with probability $e^{(-\Delta E/kT)}$ where ΔE is the difference between the energy at the previous iteration and the current energy, T is temperature specified in the `PESExploration%Temperature` keyword, see *Job selection and main options* (page 126)).

By iteratively performing these steps, the basin hopping procedure explores the highly-dimensional configuration space of the system, looking for the global minimum and finding local minima along the way. The number of iteration can be specified via the `Steps` keywords (see *below* (page 136) for more details).

For a more comprehensive PES exploration, AMS can perform several expeditions (potentially with several explorers for each expedition) in a single calculation. See the `NumExpeditions` and `NumExplorers` keywords in the *Job selection and main options* (page 126) section for more details.

See also:

- *Example: Basin Hopping for Ar 13 cluster* (page 282)
- GUI tutorials *Cluster Growth: Cobalt Clusters and Water dissociation on an oxide surface*.

Input options

To perform a Basin Hopping calculation you should specify:

² David J. Wales and Jonathan P. K. Doye *Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms*, *J. Phys. Chem. A* 1997, 101, 28, 5111–5116 (<https://doi.org/10.1021/jp970984n>)

```

Task PESExploration

PESExploration
  Job BasinHopping

  Temperature float

  BasinHopping
    # Options for the Basin Hopping procedure. See description below.
  End
End

```

The PESExploration%Temperature keyword is described in the *Job selection and main options* (page 126) section.

The following input keywords are specific to Basin Hopping procedure:

```

PESExploration
  BasinHopping
    DisplaceAtomsInRegion string
    Displacement float
    PushApartDistance float
    Steps integer
  End
End

```

PESExploration

BasinHopping

Type Block

Description Configures the details of the Basin Hopping subtask.

DisplaceAtomsInRegion

Type String

Default value

Description If you specify a region name here, only the atoms belonging to this region will be displaced during the basin hopping procedure. For more details on regions, see the documentation on the System definition.

Displacement

Type Float

Default value 0.5

Unit Angstrom

Description Displacement in each degree of freedom.

PushApartDistance

Type Float

Default value 0.4

Unit Angstrom

Description Push atoms apart until no atoms are closer than this distance. This criterion is enforced for the initial structure and all those generated by random displacements.

Steps**Type** Integer**Default value** 20**Description** Number of displace & optimize Monte-Carlo steps to take.

4.11.9 Landscape Refinement

The `LandscapeRefinement` Job can be used to re-optimize the critical points (local minima and saddle points) found in a previous PES exploration calculation using a different engine or different engine settings.

Since PES exploration calculations can be computationally demanding, a possible strategy is to first perform a PES exploration using either a fast engine or computationally cheap settings for the engine of choice, and then to *refine* the energy landscape obtained using a more accurate (and computationally more expensive) method.

See also:

Check the [tutorial](#) on automated reaction pathway discovery for hydrohalogenation. There the landscape refinement is used to go from the DFTB level of theory to DFT.

The `LandscapeRefinement` job needs a previously computed Energy Landscape. See the section [Continue a PES exploration from a previous calculation](#) (page 129) for more details.

To perform a Landscape Refinement calculation you should specify:

```
Task LandscapeRefinement
PESExploration
  Job LandscapeRefinement
  LoadEnergyLandscape
    Path path/to/previous/calculation/ams.results
  End
End
```

Warning: If you perform a LandscapeRefinement of an Energy Landscapes obtained with *ProcessSearch* (page 132) job, the connections between TS and minima are **NOT** recomputed.

Lets say, for example, that after a ProcessSearch calculation using DFTB we found a TS connecting minima with Id 3 and 4. After a LandscapeRefinement using a different engine (for example ADF) it is no longer assured that the refined TS will still connect the same two minima.

When you visualize a refined energy landscape using AMSmovie, be aware that some of the connections might be incorrect.

4.11.10 Optimizer

Geometry optimizations are performed for most PES Explorations job types. In the `PESExploration%Optimizer` block you may configure some of the parameters for these geometry optimizations:

```
PESExploration
  Optimizer
    ConvergedForce float
```

(continues on next page)

(continued from previous page)

```
MaxIterations integer
Method [CG | QM | LBFGS | FIRE | SD]
End
End
```

PESExploration

Optimizer

Type Block

Description Configures the details of the geometry optimizers used by the PES explorers.

ConvergedForce

Type Float

Default value 0.005

Unit eV/Angstrom

Description Convergence threshold for nuclear gradients.

MaxIterations

Type Integer

Default value 400

Description Maximum number of iterations allowed for optimizations.

Method

Type Multiple Choice

Default value CG

Options [CG, QM, LBFGS, FIRE, SD]

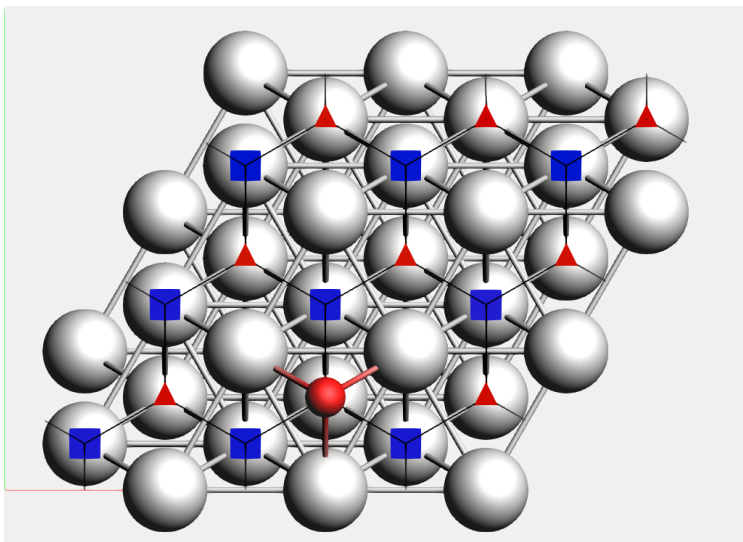
Description Select the method for geometry optimizations.

4.11.11 Binding Sites

Binding sites can be determined from an Energy Landscape.

When computing binding sites you will first need to define a **reference region**, which typically will be a surface or a cluster (*regions* (page 34) can be specified in the *system input block* (page 29), or via the “Model → Regions” panel in AMSinput). AMS will then go through all the local minima of Energy Landscape and mark as ‘binding sites’ the positions where an atom of an adsorbed molecule is neighboring atoms in the reference region.

For instance, in *Example: PES Exploration, Binding Sites for O on Pt 111* (page 283), the platinum surface is the reference region, while the oxygen atom is the adsorbate. In the picture below, you can see the oxygen binding sites on the platinum 111 surface (the two different types of binding sites are marked by a blue square and a red triangle respectively).



Lines connecting the the binding sites will be drawn if 1) there is a transition state connecting two local minima associated to these binding-sites (notice that may there are multiple local minima associated with the same binding site), and 2) there is at least an atom (from the adsorbate region) that changes its position from the first binding site to the second one mediated by the same transition state described above. In this process, AMS will align as much as possible all local minima and transition states to the input's file structure but ignoring the atoms in the region adsorbate trying to establish a common reference frame (see also the `StatesAlignment` input block below).

See also:

- *Example: PES Exploration, Binding Sites for O on Pt 111* (page 283)
- *Cluster Growth: Cobalt Clusters*

Input options

There are two distinct ways of triggering the computation of binding sites. You can either:

compute the binding sites at the end of a *Process Search* (page 132), *Basin Hopping* (page 135) or *Landscape Refinement* (page 137) calculation by setting the `PESExploration%BindingSites%Calculate` option to `yes`:

```
PESExploration
  Job [ProcessSearch | BasinHopping | LandscapeRefinement]

  BindingSites
    Calculate Yes
  End

  StatesAlignment
    ReferenceRegion reference_region_name
  End
End
```

or

compute the binding sites by setting the `PESExploration%Job` to `BindingSites` and load a previously computed Energy Landscape (see *Example: PES Exploration, Binding Sites for O on Pt 111* (page 283)):

```
PESExploration
  Job BindingSites
```

(continues on next page)

(continued from previous page)

```

LoadEnergyLandscape
  Path path/to/previous/calculation/ams.results
End

StatesAlignment
  ReferenceRegion reference_region_name
End
End

```

The following input options are related to the calculation of binding sites:

```

PESExploration
  BindingSites
    AllowDisconnected Yes/No
    Calculate Yes/No
    DistanceDifference float
  End
End

```

PESExploration**BindingSites****Type** Block**Description** Options related to the calculation of binding sites.**AllowDisconnected****Type** Bool**Default value** No**Description** Allow disconnected binding sites.**Calculate****Type** Bool**Default value** No**Description** Calculate binding sites at the end of a job. Not needed for Binding Sites job.**DistanceDifference****Type** Float**Default value** -1.0**Unit** Angstrom**Description** If the distance between two mapped binding-sites is larger than this threshold, the binding-sites are considered different. If not specified, its value will set equal to [PESExploration%StructureComparison%DistanceDifference]

The following input options are related to the definition of the reference region and alignment thereof:

```

PESExploration
  StatesAlignment
    DistanceDifference float
    ReferenceRegion string

```

(continues on next page)

(continued from previous page)

End End

PESExploration**StatesAlignment****Type** Block**Description** Configures details of how the energy landscape configurations are aligned respect to the main chemical system [System].**DistanceDifference****Type** Float**Default value** -1.0**Unit** Angstrom**Description** If the distance between two mapped atoms is larger than this threshold, the configuration is considered not aligned. If not specified, its value will set equal to [PESExploration%StructureComparison%DistanceDifference]**ReferenceRegion****Type** String**Default value****Description** Defines the region that is considered as the reference for alignments. Atoms outside this region are ignored in the alignments. TODO: Description of which PES exploration jobs actually use this ...

4.11.12 References

GRADIENTS, HESSIAN, STRESS TENSOR, ELASTICITY

No matter what application the AMS driver is used for, in one way or another it always explores the potential energy surface (PES) of the system. One can furthermore ask AMS to calculate additional properties of the PES in the points that are visited. These are mostly (but not exclusively) derivatives of the energy, e.g. we can ask AMS to calculate the gradients or the Hessian in the visited points. In general all these PES point properties are requested through the `Properties` block in the AMS input:

```
Properties
  Gradients Yes/No
  StressTensor Yes/No
  Hessian Yes/No
  SelectedRegionForHessian string
  PESPointCharacter Yes/No
  ElasticTensor Yes/No
End
```

This properties described on this page in the AMS manual are all related to derivatives of the energy.

Note that because these properties are tied to a particular point on the potential energy surface, they are found on the *engine output files* (page 17). Note also that the properties are not always calculated in every PES point that the AMS driver visits during a calculation. By default they are only calculated in *special* PES points, where the definition of special depends on the *task* (page 41) AMS is performing: For a *geometry optimization* (page 43) properties would for example only be calculated at the final, converged geometry. This behavior can often be modified by keywords special to the particular running task.

5.1 Nuclear gradients

The first derivative with respect to the nuclear coordinates can be requested as follows:

```
Properties
  Gradients Yes/No
End
```

Properties

Gradients

Type Bool

Default value No

GUI name Nuclear gradients

Description Calculate the nuclear gradients.

Note that these are gradients, not forces, the difference being the sign. The gradients are printed in the output and written to the *engine result file* (page 17) belonging to the particular point on the PES in the `AMSResults%Gradients` variable as a $3 \times n_{\text{atoms}}$ array in atomic units (Hartree/Bohr).

5.2 Hessian

The calculation of the second derivative of the total energy with respect to the nuclear coordinates is enabled by:

```
Properties
  Hessian Yes/No
  SelectedRegionForHessian string
End
```

Properties

Hessian

Type Bool

Default value No

Description Whether or not to calculate the Hessian.

SelectedRegionForHessian

Type String

GUI name Hessian only for

Description Compute the Hessian matrix elements only for the atoms in a particular region. If not specified, the Hessian will be computed for all atoms.

The Hessian is not printed to the text output, but is saved in the engine result file as variable `AMSResults%Hessian`. Note that this is just the plain partial second derivatives (no mass-weighting) of the total energy. The $3 \times N_{\text{atoms}}$ columns/rows of the matrix are grouped by atom: the first three rows/columns correspond to the first atom, etc..

Note that the AMS driver also supports the *Mobile Block Hessian* (page 156) (MBH) method, which allows treating parts of the system as rigid blocks.

Often one is not interested in the Hessian matrix itself, but just in using it for the calculation of IR frequencies or to characterize a PES point (as e.g. a local minimum or a saddle point). For these application, see the following pages in the manual:

- *IR frequencies* (page 153)
- *PES point character* (page 144)

5.3 PES point character

A PES point can according to the slope and curvature of the PES at that point be classified in the following categories:

- A local minimum on the PES with vanishing nuclear gradients and no negative frequencies.
- A transition state with vanishing nuclear gradients and exactly one negative frequency, i.e. a first order saddle point on the PES.
- A higher order saddle point, i.e. a stationary point on the PES with vanishing nuclear gradients but more than one imaginary frequency.
- A non-stationary point on the PES. Here the gradients are non-zero.

This classification can easily be done if both the gradients and the normal modes have already been calculated. However, calculating the full Hessian needed for the entire set of normal modes is very expensive and undesirable if one only wants to know the character of a PES point. The AMS driver can quickly, and without calculating the full Hessian, characterize a PES point into one of the above categories. This can be used to confirm the success of e.g. a *transition state search* (page 57) or *geometry optimization* (page 43). A PES point can be characterized by requesting `PESPointCharacter` as a property:

```
Properties
  PESPointCharacter Yes/No
End
```

Properties

PESPointCharacter

Type Bool

Default value No

GUI name Characterize PES point

Description Determine whether the sampled PES point is a minimum or saddle point. Note that for large systems this does not entail the calculation of the full Hessian and can therefore be used to quickly confirm the success of a geometry optimization or transition state search.

This will calculate the few lowest normal modes using an iterative diagonalization of the Hessian¹ based on a Davidson algorithm implemented in the PRIMME library². The procedure has been optimized for finding a small number of low-lying eigenvalues in as few matrix-vector multiplications (and thus single point calculations) as possible. This is facilitated by performing the iterative method using a pre-conditioning matrix based on an approximation of the Hessian. The approximate Hessian is obtained from the full Hessian at a lower level of theory. This calculation also provides the initial guesses for the desired normal modes. What the lower level of theory is depends on the main engine used in the calculation: DFTB with the GFN1-xTB model is used as the lower level of theory for relatively slow engines, e.g. DFT based engines. For semi-empirical engines like DFTB or MOPAC, the lower level of theory is currently UFF. It is currently not possible to change the engine used to obtain the preconditioning Hessian and the approximate modes.

- Note that the iterative calculation of the normal modes is skipped when ...
 1. ... the nuclear gradients are so large that the PES point is considered non-stationary. The calculation of the modes is then just not necessary for classifying it.
 2. ... the full normal modes or Hessian have also been requested. The iterative calculation is then not necessary, as all modes are already known.
 3. ... the molecule is very small. (For small systems the iterative calculation of the few lowest normal modes is not faster than the full calculation of all modes, so all modes are calculated instead.)
- The classification as a stationary or non-stationary point uses the gradient convergence criterion from the geometry optimizer as the tolerance, see *geometry optimization* (page 43). This makes sure that the criterion for what is considered converged/stationary is always in sync between the optimizer and the PES point characterization.
- For periodic systems the PES point characterization does not take the lattice degrees of freedom into account. A PES point where the nuclear gradients are small enough would for example be classified as a stationary point, even if the system is under stress.

Details of the iterative procedure can be configured in the `PESPointCharacter` block:

¹ P. Deglmann and F. Furche, *Efficient characterization of stationary points on potential energy surfaces*, J. Chem. Phys. 117, 9535 (2002) (<https://doi.org/10.1063/1.1523393>)

² A. Stathopoulos and J. R. McCombs, *PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description*, ACM Transactions on Mathematical Software, Vol. 37, No. 2, (2010), 21:1–21:30. (<https://doi.org/10.1145/1731022.1731031>)

```

PESPointCharacter
  Displacement float
  NegativeFrequenciesTolerance float
  NumberOfModes integer
  Tolerance float
End

```

PESPointCharacter**Type** Block**Description** Options for the characterization of PES points.**Displacement****Type** Float**Default value** 0.04

Description Controls the size of the displacements used for numerical differentiation: The displaced geometries are calculated by taking the original coordinates and adding the mass-weighted mode times the reduced mass of the mode times the value of this keyword.

NegativeFrequenciesTolerance**Type** Float**Default value** -10.0**Unit** cm-1

Description The threshold in frequency below which a mode is considered imaginary, i.e. indicating a transition state. This is a small negative number, as very small negative frequencies are normally due to numerical noise on an essentially flat PES and do not indicate true transition states.

NumberOfModes**Type** Integer**Default value** 2

Description The number of (lowest) eigenvalues that should be checked.

Tolerance**Type** Float**Default value** 0.016

Description Convergence tolerance for residual in iterative Davidson diagonalization.

- Note that the residual tolerance that can be achieved is limited by the numerical differentiation that is performed. The default values should apply in most cases, but if convergence becomes a problem one may choose to increase the tolerance or to increase the step size (slightly). Note that the default residual tolerance is lower than for the other mode selective methods. This is because PRIMME uses a different convergence criteria than mode tracking/refinement. The higher value used as a default will therefore not result in decreased levels of accuracy. The method will bail if the number of iterations exceeds the number of normal modes as at this point still achieving convergence becomes unlikely, in part due to the next point.
- In order to avoid producing the known and irrelevant rigid modes, the method searches for normal modes orthogonal to six (or five) rigid modes. Imperfections due to the numerical differentiation may mean that the translational and rotational rigid modes are not exact eigenmodes of the Hessian that is constructed. As a result, some part of the lowest vibrational normal mode may lie in the span of the theoretical rigid modes and

therefore be inaccessible to the Davidson method. This places a lower bound on the residual tolerance that can be achieved, which is directly related to the numerical differentiation accuracy. The take-away: do not set the tolerance too low, the default usually suffices.

- Behind the scenes, the method actually computes a few more modes than requested. In the case of multiplicities, eigenvalues may still converge out of order. These additional eigenvalues essentially guarantee that the obtained modes are indeed the lowest ones.

5.4 Thermodynamics, gas phase Gibbs free energy

At the end of a completed IR Frequencies (normal modes) calculation, a survey is given of thermodynamic properties: entropy, internal energy, constant volume heat capacity, enthalpy and Gibbs free energy, see:

- *IR frequencies* (page 153)
 - *Thermodynamics* (page 173)
 - *Gibbs free energy change for a gas phase reaction* (page 176)

5.5 Stress tensor

For periodic systems (chains, slabs, bulk) one can also request the clamped-ion stress tensor (note: the clamped-ion stress is only part of the *true* stress tensor):

```
Properties
  StressTensor Yes/No
End
```

Properties

StressTensor

Type Bool

Default value No

GUI name Stress tensor

Description Calculate the stress tensor.

The clamped-ion stress tensor σ_α (Voigt notation) is computed via numerical differentiation of the energy E WRT a strain deformations ϵ_α keeping the atomic fractional coordinates constant:

$$\sigma_\alpha = \frac{1}{V_0} \left. \frac{\partial E}{\partial \epsilon_\alpha} \right|_{\text{constant atomic fractional coordinates}}$$

where V_0 is the volume of the unit cell (for 2D periodic system V_0 is the area of the unit cell, and for 1D periodic system V_0 is the length of the unit cell).

The clamped-ion stress tensor (in Cartesian notation) is written to the engine result file in `AMSResults%StressTensor`.

5.6 Elastic tensor

The elastic tensor $c_{\alpha,\beta}$ (Voigt notation) is computed via second order numerical differentiation of the energy E WRT strain deformations ϵ_α and ϵ_β :

$$c_{\alpha,\beta} = \frac{1}{V_0} \frac{\partial^2 E}{\partial \epsilon_\alpha \partial \epsilon_\beta}$$

where V_0 is the volume of the unit cell (for 2D periodic system V_0 is the area of the unit cell, and for 1D periodic system V_0 is the length of the unit cell).

For each strain deformation $\epsilon_\alpha \epsilon_\beta$, the atomic positions will be optimized. The elastic tensor can be computed for any periodicity, i.e. 1D, 2D and 3D.

See also:

Example: Elastic tensor (page 306)

To compute the elastic tensor, request it in the `Properties` input block of AMS:

```
Properties
  ElasticTensor Yes/No
End
```

Note: The elastic tensor should be computed at the fully optimized geometry. One should therefore perform a geometry optimization of all degrees of freedom, **including the lattice vectors**. It is recommended to use a tight gradient convergence threshold for the geometry optimization (e.g. 1.0E-4). Note that all this can be done in one job by combining the *geometry optimization task* (page 43) with the elastic tensor calculation.

The elastic tensor (in Voigt notation) is printed to the output file and stored in the *engine result file* (page 17) in the `AMSResults` section (for 3D system, the elastic tensor in Voigt notation is a 6x6 matrix; for 2D systems is a 3x3 matrix; for 1D systems is just one number).

Options for the numerical differentiation procedure can be specified in the `ElasticTensor` input block:

```
ElasticTensor
  MaxGradientForGeoOpt float
  StrainStepSize float
End
```

ElasticTensor

Type Block

Description Options for numerical evaluation of the elastic tensor.

MaxGradientForGeoOpt

Type Float

Default value 0.0001

Unit Hartree/Angstrom

GUI name Maximum nuclear gradient

Description Maximum nuclear gradient for the relaxation of the internal degrees of freedom of strained systems.

StrainStepSize

Type Float

Default value 0.001

Description Step size (relative) of strain deformations used for computing the elastic tensor numerically.

Pressure (page 220) or *non-isotropic external stress* (page 221) can be included in your simulation via the corresponding *engine addons* (page 217).

The elastic tensor calculation supports AMS' *double parallelization* (page 18), which can perform the calculations for the individual displacements in parallel. This is configured automatically, but can be further tweaked using the keys in the `NumericalDifferentiation%Parallel` block:

```
ElasticTensor
  Parallel
    nCoresPerGroup integer
    nGroups integer
    nNodesPerGroup integer
  End
End
```

ElasticTensor

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

5.7 Numerical differentiation options

The following options apply whenever AMS computes gradients, Hessians or stress tensors via numerical differentiation.

```
NumericalDifferentiation
  NuclearStepSize float
  StrainStepSize float
End
```

NumericalDifferentiation

Type Block

Description Define options for numerical differentiations, that is the numerical calculation of gradients, Hessian and the stress tensor for periodic systems.

NuclearStepSize

Type Float

Default value 0.005

Unit Bohr

Description Step size for numerical nuclear gradient calculation.

StrainStepSize

Type Float

Default value 0.001

Description Step size (relative) for numerical stress tensor calculation.

AMS may use symmetry (key `NumericalDifferentiation%UseSymmetry`) in case of numerical differentiation calculations. If symmetry is used only symmetry unique atoms are displaced. Symmetry is only recognized if the starting geometry has symmetry. Symmetry is only used for molecules if the molecule has a specific orientation in space, like that the z-axis is the main rotation axis. If the GUI is used one can click the Symmetrize button (the star), such that the GUI will (try to) symmetrize and reorient the molecule. There are some cases that even after such symmetrization, the orientation of the molecule is not what is needed for the symmetry to be used in case of numerical differentiation calculations. If that is the case or if key `NumericalDifferentiation%UseSymmetry` is set to 'False', then no symmetry will be used.

The numerical differentiation calculation supports AMS' *double parallelization* (page 18), which can perform the calculations for the individual displacements in parallel. This is configured automatically, but can be further tweaked using the keys in the `NumericalDifferentiation%Parallel` block:

```
NumericalDifferentiation
  Parallel
    nCoresPerGroup integer
    nGroups integer
    nNodesPerGroup integer
  End
End
```

NumericalDifferentiation

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

VIBRATIONAL SPECTROSCOPY

6.1 General

The starting point is the Hessian of the system, being the second derivative of the energy with respect to the atomic coordinates.

The eigenvalues of the Hessian are the frequencies and the eigen vectors are the normal modes.

As the calculation of the full Hessian is very expensive there are several ways to avoid it, so that you only get a part of the full spectrum, or only modes for a region of the system, see *IR frequencies and normal modes* (page 153) section.

A full, partial, or approximate Hessian in itself can be useful for a (Hessian-based) geometry optimization or a transition state search.

Vibrational spectra are obtained by differentiating a property along the normal modes at a (local) minimum of the PES. So for spectra you need to optimize the geometry first, otherwise you get negative frequencies.

The Normal modes and or vibrational spectra can be requested via the `Properties` block

```
Properties
  NormalModes Yes/No
  Raman Yes/No
  VROA Yes/No
  VCD Yes/No
  Phonons Yes/No
End
```

When requesting the normal modes, the IR intensities are calculated, as they are very cheap.

See also:

[Tutorials on Vibrational Spectroscopy](#)

6.1.1 Where are the results?

Because the results of a vibrational spectroscopy calculation are tied to a particular point on the potential energy surface, they are found on the *engine output files* (page 17). Note also that the properties are not always calculated in every PES point that the AMS driver visits during a calculation. By default they are only calculated in *special* PES points, where the definition of special depends on the *task* (page 41) AMS is performing: For a *geometry optimization* (page 43) properties would for example only be calculated at the final, converged geometry. This behavior can often be modified by keywords special to the particular running task.

6.2 IR frequencies and normal modes

6.2.1 All vibrational Modes

The calculation of the normal modes of vibration can be requested with:

```
Properties
  NormalModes Yes/No
End
```

Typically used icw with *Task SinglePoint* (page 41), *Task GeometryOptimization* (page 43), or *Task TransitionStateSearch* (page 57). In case of geometry optimization or transition state search the normal modes will only be calculated at the final, converged geometry.

Properties

NormalModes

Type Bool

Default value No

GUI name Frequencies

Description Calculate the frequencies and normal modes of vibration, and for molecules also the corresponding IR intensities if the engine supports the calculation of dipole moments.

The molecular normal modes are normally calculated within the harmonic oscillation model. If the molecule is in its equilibrium conformation, it sits in the lowest point (at least locally) on the PES. The cross-section of the PES profile close to this point can then be assumed to be approximately parabolic, such that the second derivative of the energy w.r.t a nuclear coordinate can be interpreted as a force constant for the harmonic oscillation of an atom along this coordinate. Since molecular vibrations in polyatomics involve the simultaneous displacement of multiple atoms, this harmonic oscillator model can be generalized to multiple nuclear coordinates. The normal modes and their frequencies then become eigenvectors and eigenvalues of a force constant matrix, the Hessian:

$$H_{ij} = \frac{\partial^2 E}{\partial R_i \partial R_j}$$

The (non-mass-weighted) Hessian is saved in the engine result file as variable `AMSResults%Hessian`. It is not printed to the text output. The column/row indices are ordered as: x-component of atom 1, y-component of atom 1, z-component of atom 1, x-component of atom 2, etc.

Most *engines* (page 209) cannot calculate the Hessian analytically. The Hessian is then constructed column-wise through numerical differentiation of the energy gradients w.r.t. each nuclear coordinate. AMS will set up 2 single-point calculations (1 for the positive displacement, 1 for the negative displacement), and the requested engine will return the energy gradients at these displacements. These gradients are calculated analytically for most engines.

Note: Numerical calculation of the full Hessian requires 6N single points calculation, which can take a considerable amount of time for large systems. A mode selective method can be a fast alternative, see *mode scanning* (page 159), *mode refinement* (page 160), and *mode tracking* (page 163).

When requesting the normal modes calculation, integrated IR intensities are simultaneously calculated during the finite differentiation steps when constructing the Hessian (as long as dipole moments are supported by the engine). These IR intensities are calculated from the numerical dipole gradients:

$$I_{IR} = \frac{N\pi}{3c^2} \sum_{\alpha} \left(\sum_j \frac{\partial \mu_{\alpha}}{\partial R_j^m} Q_j^m \right)^2$$

Where α denotes the x-,y- and z-components of the dipole moment μ , and Q^m is the mass-weighted vibrational normal mode.

The resulting IR spectrum can be visualized by opening the engine result file with AMSspectra. The normal modes of vibration and the IR intensities are saved to the *engine result file* (page 17) in the Vibrations section.

Note: The calculation of the normal modes of vibration needs to be done the system's equilibrium geometry. So one should either run the normal modes calculation using an already optimized geometry, or combine both steps into one job by using the *geometry optimization task* (page 43) together with the `Properties%NormalModes` keyword.

Symmetry labels of the normal modes may be calculated if AMS uses symmetry in the calculation (key `UseSymmetry`). If symmetry is used, the normal modes are projected against symmetric displacements for each irrep. If that is not successful the symmetry label is 'MIX'. Symmetry is only recognized if the geometry is (al-most) perfectly symmetric and has a specific orientation in space. You can use the `Symmetrize` key in the *System block* (page 29) to symmetrize and reorient the molecule. If the AMSinput GUI module is used, one can click the Symmetrize button (the star) and the GUI will try to symmetrize and reorient the molecule.

Rescanning Imaginary modes

The `ReScanModes` keyword can be used to calculate more accurately frequencies of specific modes after a normal modes calculation. It is identical to the `ScanFreq` option that was available for older versions of ADF and BAND. Primarily used to identify spurious imaginary modes, and is on by default for this purpose. See also the *Mode Scanning* (page 159) task, which is an extension of this method, but which is not on by default.

```
NormalModes
  ReScanModes Yes/No
  ReScanFreqRange float_list
End
```

NormalModes

ReScanModes

Type Bool

Default value Yes

GUI name Re-scan modes

Description Whether or not to scan imaginary modes after normal modes calculation has concluded.

ReScanFreqRange

Type Float List

Default value [-10000000.0, 10.0]

Unit cm-1

Recurring True

GUI name Re-scan range

Description Specifies a frequency range within which all modes will be scanned. 2 numbers: an upper and a lower bound.

6.2.2 Symmetric Displacements

```
NormalModes
  Displacements Symmetric
End
```

Specify `Displacements Symmetric` to calculate the energy Hessian using finite differences in symmetry-adapted displacements, and the corresponding normal modes.

```
NormalModes
  SymmetricDisplacements
    Type [All | Infrared | Raman | InfraredAndRaman]
  End
End
```

If `Type Infrared` or `Type Raman` is specified then only irreps that result in non-zero intensities for the corresponding spectroscopy will be included in the calculation. Using this feature may save a lot of time for large symmetric molecules by skipping calculation of normal modes that would not contribute to the spectrum anyway. If `Type InfraredAndRaman` is specified then vibrations that have a non-zero IR or Raman intensity will be calculated. If `Type All` is specified then all vibrations will be calculated. For multi-dimensional irreps (such as E and T) only the first component will be computed. For any component beyond the first, the frequencies and intensities will be copied from the first one.

NormalModes

SymmetricDisplacements

Type Block

Description Configures details of the calculation of the frequencies and normal modes of vibration in symmetric displacements.

Type

Type Multiple Choice

Default value All

Options [All, Infrared, Raman, InfraredAndRaman]

GUI name Symm Frequencies

Description For symmetric molecules it is possible to choose only the modes that have non-zero IR or Raman intensity (or either of them) by symmetry. In order to calculate the Raman intensities the Raman property must be requested.

Warning: Specifying `Type Raman` alone does not trigger calculation of the Raman intensities. In order to calculate the Raman spectrum one should also specify `Raman True`.

Note: `Displacements Symmetric` will also produce a $3N$ -by- $3N$ Hessian matrix but if the `Type` key's argument is not `All` then this matrix will likely have many zero eigenvalues.

6.2.3 Mobile Block Hessian (MBH)

```
NormalModes
  Displacements Block
End
```

Specify `Displacements Block` for the Block Normal Modes option (also known as **Mobile Block Hessian**, or **MBH**^{1,2}). MBH is useful when calculating vibrational frequencies of a small part of a very large system (molecule or cluster). Calculation of the full spectrum of such a system may be inefficient and is unnecessary if one is interested in one particular part. Besides, it may be difficult to extract normal modes related to the interesting sub-system out of the whole spectrum. Using Block Normal Modes it is possible to treat parts of the system as rigid blocks. Each block will usually have only six frequencies related to its rigid motions compared to $3*N$ for when each atom of the block is treated separately.

MBH is suitable to calculate frequencies in partially optimized structures. Assume a geometry optimization is performed with the `Block` key in the `Constraints` input block [see constrained geometry optimizations]. During the geometry optimization, the shape of the block is not changed. The internal geometry of the block is kept fixed, but the block as a whole can still translate or rotate.

At the end of such a partial geometry optimization, the position and orientation of the block is optimized, thus the total force on the block is zero. However, there might be still some residual forces within a block, since those degrees of freedom were not optimized. A traditional frequency calculation performed on this partially optimized structure might result in non-physical imaginary frequencies without a clear interpretation. Therefore one should use an adapted formulation of normal mode analysis: the Mobile Block Hessian method. MBH does not consider the internal degrees of freedom of the block (on which residual forces) apply, but instead uses the position/orientation of the block as coordinates. In the resulting normal mode eigenvectors, all atoms within the same block move collectively.

Of course, MBH can also be applied on a fully optimized structure.

Accuracy

```
NormalModes
  BlockDisplacements
    AngularDisplacement float
    BlockAtoms integer_list
    BlockRegion string
    Parallel
      nCoresPerGroup integer
      nGroups integer
      nNodesPerGroup integer
    End
    RadialDisplacement float
  End
End
```

NormalModes

BlockDisplacements

Type Block

Description Configures details of a Block Normal Modes (a.k.a. Mobile Block Hessian, or MBH) calculation.

AngularDisplacement

¹ A. Ghysels, D. Van Neck, V. Van Speybroeck, T. Verstraelen and M. Waroquier, *Vibrational Modes in partially optimized molecular systems*, *Journal of Chemical Physics* 126, 224102 (2007) (<https://doi.org/10.1063/1.2737444>)

² A. Ghysels, D. Van Neck and M. Waroquier, *Cartesian formulation of the Mobile Block Hessian Approach to vibrational analysis in partially optimized systems*, *Journal of Chemical Physics* 127, 164108 (2007) (<https://doi.org/10.1063/1.2789429>)

Type Float

Default value 0.5

Unit Degree

Description Relative step size for rotational degrees of freedom during Block Normal Modes finite difference calculations. It will be scaled with the characteristic block size.

BlockAtoms

Type Integer List

Recurring True

Description List of atoms belonging to a block. You can have multiple BlockAtoms.

BlockRegion

Type String

Recurring True

Description The region to to be considered a block. You can have multiple BlockRegions, also in combination with BlockAtoms.

Parallel

Type Block

Description Configuration for how the individual displacements are calculated in parallel.

nCoresPerGroup

Type Integer

Description Number of cores in each working group.

nGroups

Type Integer

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Cores per task

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

RadialDisplacement

Type Float

Default value 0.005

Unit Angstrom

Description Step size for translational degrees of freedom during Block Normal Modes finite difference calculations.

The second derivatives of the energy with respect to Cartesian displacements of the free atoms and those with respect to block motions (3 translation plus 3 rotations) are calculated by numerical differentiation of the gradient. The accuracy of the second derivatives is determined by the accuracy of the gradient evaluation and the step size in the numerical

differentiation. The `RadialDisplacement` and `AngularDisplacement` parameters can be specified to set the step size for Cartesian displacements (translations) and block rotations respectively. The step size for angles is automatically scaled with the block size.

Note: Blocks should consist of at least 3 atoms (i.e. block of 1 or 2 atoms are not supported).

6.2.4 Mode Scanning

Mode Scanning can be used to obtain more accurate approximations for properties obtained by numerical differentiation along the vibrational normal modes (frequencies, intensities, Raman, etc.), without changing the modes themselves. Mode Scanning is an extension of the frequency scanning options (`ScanFreq`) that were part of ADF and BAND in earlier versions of the Amsterdam Modeling Suite. These latter options are still available as the `ReScanModes` keyword in the `NormalModes` block, if these are requested during a calculation.

- Primarily used to identify spurious imaginary modes.
- Improve numerical accuracy of normal mode properties.
- Rescanning modes using a different level of theory.

Theory

Vibrational normal modes are usually obtained as eigenvectors of the Hessian matrix. A common problem with this scheme however, is that due to numerical errors in constructing this Hessian, low-frequency vibrations may be reported to have imaginary frequencies instead. The Mode Scanning task allows for re-calculation of the frequency of these modes. The Mode Scanning task does not change the normal modes itself, only its properties. This Mode Scanning task allows you to confirm whether reported imaginary frequencies are attributed to transition states or whether they are simply due to numerical errors.

Given a user-supplied mode Q , the frequency is calculated from the force constant:

$$k = \frac{\partial^2 E}{\partial^2 Q}$$

$$\nu = \frac{1}{2\pi c} \sqrt{\frac{k}{\mu_r}}$$

This is again done by numerical differentiation of the energy gradients, requiring AMS to set up 2 single point calculations per selected normal mode. Integrated IR intensities are also calculated simultaneously (if dipole moments are supported by the *engine* (page 209)):

$$I_{IR} = \frac{N\pi}{3c^2} \sum_{\alpha} \left(\frac{\partial \mu_{\alpha}}{\partial Q^m} \right)^2$$

Where the derivative is with respect to the mass-weighted normal mode.

It is also possible to use this method to selectively re-calculate the normal mode properties for different engine settings. This has two distinct uses:

- If the modes were originally generated using a finite difference method, a different stepsize can be used. For strong vibrations (high frequencies), large stepsizes may cause inaccuracies due to increasing anharmonic contributions. For weak vibrations (low frequencies) on the other hand, stepsizes can often be too small. The displacements associated with these vibrations are small, which can give incorrect sampling of the PES profile. This should be compensated for by choosing a larger stepsize. The stepsize can be set using the `Displacement` key.

- Users can also recalculate modes using higher levels of theory. Modes generated from a full frequency analysis using e.g. DFTB can be recalculated using e.g. LDA DFT to obtain more realistic integrated IR intensities. The method used for the single point calculations can be set in the *Engine block* (page 209).

Input

A numerical frequency calculation is performed by requesting the `VibrationalAnalysis` task with `Type ModeScanning`:

```
Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeScanning
  Displacement 0.001
  NormalModes
    ModeFile adf.rkf
    # select all modes with imaginary frequencies
  ModeSelect
    ImFreq true
  End
End
End
```

The Mode Scanning tasks uses only the `NormalModes` block for its input handling. Here, `ModeFile` specifies the AMS output file containing the normal modes for which you want to calculate the frequencies. The `ModeSelect` block is used to specify which of the modes in this file should be recalculated, since we are often only interested in a select few of them. A more detailed overview of this block is given in the section *Selecting Modes* on the *main page* (page 171). Finally, `Displacement` can be used to specify the stepsize (in Bohr) for the finite differences. The stepsize is provided for displacements along the Cartesian normal modes.

The Mode Scanning module is the main driving force for the *Mode Tracking* (page 163) and *Vibrational Mode Refinement* (page 160) tasks, which provide more advanced options for refining not only the properties of the modes, but also the modes themselves. Consult the relevant pages for more information. Alternatively, a simplified version of Mode Scanning is available which follows the old implementation in ADF and BAND (as the `ScanFreq` option). This version can be enabled when doing a full frequency analysis by enabling the `Properties%NormalModes` keyword. See the *Full Analysis* (page 154) page for further details.

6.2.5 Mode Refinement

With this option you can improve the normal modes, by importing previously calculated modes and then applying a more accurate engine, or more accurate settings, typically for only part of the spectrum. The vibrational Mode Refinement method not only refines frequencies from a previous calculation, but also tries to correct the vibrational modes themselves.

- Refinement of spectral regions requires a sufficient number of modes in the basis to be accurate.
- 1-step refinement. No iterative improvement possible. (Unless followed by a separate Mode Tracking calculation.)
- Quality of the results depends on accuracy of the selected guess modes.

If we start from e.g. a semi-empirical method such as in MOPAC, we can get approximations for the vibrational modes. Mode Refinement then re-calculates part of the Hessian for a subset of these modes using a more accurate method such as GGA DFT, and updates the normal modes themselves to fit this more accurate method. It is intended to circumvent the expensive calculation of the Hessian if you are only interested in a (small) part of the full spectrum.

This is based on the method in reference³.

Because the Mode Refinement method uses linear combinations of the guess modes, its accuracy depends on the set of modes that is supplied.

- If we want to e.g. obtain a mode which includes a C=O stretch, then the initial set must contain a mode which has this C=O stretch, otherwise this cannot be included in the refined modes.
- If we are refining a region containing many similar modes, e.g. vibrations of aromatic ring backbones, and we only use part of this spectral region for the initial set, the set of refined modes will “drift” towards the centre of the spectral region as a result of mode-mixing. This is again an artefact of missing character in the modes.
- This mode-mixing may result in reduced accuracy for some of the modes, as this procedure minimizes the total error for all of the modes. Instead of having a couple of modes with large errors, mode-mixing tends to spread out the error across multiple normal modes. Adding 1 “bad” mode to the basis can then negatively affect your results.
- The advantage of Mode Refinement over Mode Tracking is the ability to refine entire spectral regions at once. If we have a good basis, Mode Refinement can be less computationally expensive than Mode Tracking. If you want to refine larger sections of the spectrum, Mode Refinement is therefore recommended. If you only want to calculate a select few modes, use Mode Tracking to avoid basis dependence and to assure accuracy of the obtained modes.
- For characteristic peaks, Mode Tracking shows very good convergence, and will thus be cheaper to use than Mode Refinement. For (semi-)degenerate modes however, Mode Refinement works better due to the poor tracking performance for these modes.

See also:

The [GUI tutorial](#) on Mode Refinement.

Theory

We are going to start from a set of normal modes b , obtained from e.g. a semi-empirical or force-field method. First, this task runs the *numerical frequency* (page 159) calculation for all selected normal modes, but this time using an ab initio method such as DFT. During the finite difference steps, we also calculate the projection of the Hessian onto the normal modes:

$$\sigma_i = H^m \cdot b_i^m = \frac{\partial^2 E}{\partial R_i^m \partial b^m}$$

This term is calculated through finite differences on the analytical gradients of the electronic energy along the mass-weighted normal modes b^m . The index i denotes the $3N$ nuclear coordinates. These projections are then used to construct a Rayleigh matrix:

$$\tilde{H}^m = B^{mT} \cdot H^m \cdot B^m = B^{mT} \cdot \Sigma$$

Here, B^m and Σ are matrices containing the b^m and σ vectors. The eigenvectors of \tilde{H}^m give us the coefficient series for linear combinations of the normal modes b^m such that we obtain a new set of modes q :

$$q^m = \sum_k c_k \cdot b_k^m$$

These modes q are the closest approximation to the DFT-modes that we could obtain from a linear combination of the approximate modes b . In other words: the approximate modes b are used as a basis for finding the modes from a more sophisticated theory.

³ T.Q. Teodoro, M.A.J. Koenis, S.E. Galembeck, V.P. Nicu, W.J. Buma, L. Visscher, *A frequency range selection method for vibrational spectra*, J. Phys. Chem. Lett., 9 (23), 6878 (2018) (<https://doi.org/10.1021/acs.jpcclett.8b02963>)

Input

This method inherently features a trade-off:

- The computational benefit comes from only performing the finite difference calculations for the selected modes. By only selecting a small set of modes that we are interested in, we minimize computational expense.
- The more modes we select, the larger the basis for constructing the refined modes. Running for a larger number of modes yields better results. (In the extreme case, running for all $3N$ modes equates to constructing the full Hessian.)

In practice, Mode Refinement requires you to select a reasonable portion of the spectrum to get accurate results. Specifically, you should select all modes in a region of the spectrum which look similar. Ring structures for instance often feature broad frequency regions with many ring distortions. Even if you are only interested in a couple of these, you should still select all modes in this region, to assure sufficient basis size. Vibrational modes involving ring substituents can however be omitted, which is where we save computation time.

If you are interested only in IR-active vibrations, you could further minimize the basis by only selecting the approximate modes which are IR-active (since adding the non-active modes to the linear expansion does not affect the IR-intensity of the refined modes). Do note that if the semi-empirical method used for calculating the approximate modes yields poor approximations for the dipole gradients, it may be safer to include also modes with very low IR intensity. This is because their low IR-activity may have only been due to the low accuracy of the approximate method.

See also:

A [tutorial](#) showing this basis representability.

A Mode Refinement calculation is set up by requesting the `VibrationalAnalysis` task with the `Type ModeRefinement`:

```
Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeRefinement
  Displacement 0.001
  NormalModes
    ModeFile adf.rkf
    ModeSelect
    ...
  End
  ScanModes true
End
End
```

The details of the calculation are specified in the `NormalModes` block. Here, `ModeFile` specifies the AMS output file containing the normal modes for which you want to calculate the frequencies. The `ModeSelect` block is used to specify which of the modes in this file will be selected for refinement. A more detailed overview of this block is given in the section *Selecting modes* on the [main page](#) (page 171). Finally, `Displacement` can be used to specify the stepsize (in Bohr) for the finite differences. The stepsize is provided for displacements along the Cartesian normal modes.

The `ScanModes` key in the `NormalModes` block can be used to automatically run a *numerical frequencies* (page 159) calculation on the new modes q . Mode Refinement uses a linear combination of modes and properties, all obtained through finite differences. These results may still contain some minor errors due to the accumulation of numerical errors from the linear expansion, or stepsize issues in the numerical frequency calculations. While commonly not necessary, it is possible to run an additional numerical refinement calculation on the new modes to minimize these errors. Only in exceptional cases will these errors be significant. Running this additional refinement step is therefore only necessary if you need complete certainty that the results are accurate.

6.2.6 Mode Tracking

The Mode Tracking task is an interface for mode- and intensity-tracking methods, adapted from the MoViPac suite⁴⁻⁵. These methods can be used to obtain select normal modes, without having to calculate the entire vibrational spectrum. It does this through an iterative procedure.

- Calculations are conducted for each mode separately. Converges fastest for characteristic (non-highly degenerate) modes.
- Iterative approximation to the true modes. Guaranteed to give the correct normal modes if the procedure converges.
- Will not necessarily reproduce the entire spectrum as multiple guess modes can converge to the same normal mode.

Mode Tracking uses information about the known parts of the Hessian to expand its basis iteratively:

- Missing C-O stretch character can thus be recovered in this procedure, and there is no basis dependency.
- For large regions with similar modes however, it is possible that multiple guess modes converge to the same normal mode. Running mode tracking for all modes in this region might not reproduce all unique normal modes.
- The advantage of Mode Refinement over Mode Tracking is the ability to refine entire spectral regions at once. If we have a good basis, Mode Refinement can be less computationally expensive than Mode Tracking. If you want to refine larger sections of the spectrum, Mode Refinement is therefore recommended. If you only want to calculate a select few modes, use Mode Tracking to avoid basis dependence and to assure accuracy of the obtained modes.
- For characteristic peaks, Mode Tracking shows very good convergence, and will thus be cheaper to use than Mode Refinement. For (semi-)degenerate modes however, Mode Refinement works better due to the poor tracking performance for these modes.

Mode Tracking starts with a *numerical frequency* (page 159) calculation, which refines the initial guess b^m for the selected mode. The error of this mode with respect to the true Hessian eigenvector is calculated. This error is used in a (Jacobi-)Davidson algorithm to generate an additional mode. In subsequent iterations, we use these modes as approximations to the true normal modes. In this way, the error of the mode is minimized iteratively, yielding a closer approximation to true normal modes. This is how Mode Tracking differs from the Mode Refinement methods, in that it guarantees that the obtained modes are correct (assuming the procedure has converged).

See also:

The [GUI tutorial](#) on Mode Tracking.

Theory

During the numerical frequency calculation, we obtain also the projection of the Hessian onto this mode:

$$\sigma_i = H^m \cdot b_i^m = \frac{\partial^2 E}{\partial R_i^m \partial b^m}$$

This term is calculated through finite differences on the analytical gradients of the electronic energy along the mass-weighted normal modes q^m . The index i denotes the $3N$ nuclear coordinates. From this projection a Rayleigh matrix

⁴ T. Weymuth, M.P. Haag, K. Kiewisch, S. Lubert, S. Schenk, C.R. Jacob, C. Herrmann, J. Neugebauer, M. Reiher, *MoViPac: Vibrational Spectroscopy with a Robust Meta-Program for Massively Parallel Standard Inverse Calculations*, *Journal of Computational Chemistry* 33, 2186 (2012) (<https://doi.org/10.1002/jcc.23036>)

⁵ S. Lubert, J. Neugebauer, M. Reiher, *Intensity tracking for theoretical infrared spectroscopy of large molecules*, *Journal of Chemical Physics* 130, 064105 (2009) (<https://doi.org/10.1063/1.3069834>)

is generated:

$$\tilde{H}^m = B^{mT} \cdot \Sigma$$

Here, B^m and Σ are matrices containing the b^m and σ vectors for all foregoing iterations. During each iteration k , if we have not converged, we generate an updated guess vector b_k^m , and so the number of vectors in the matrices above is equal to the number of iterations k . The eigenvectors of \tilde{H}^m give us the coefficient series for linear combinations of the guess modes b^m such that we obtain approximations for the true normal modes:

$$Q^m = \sum_k c_k \cdot b_k^m$$

Each iteration, we expand the vector basis B^m , which allows this series expansion to come closer to the true normal modes each time. We can also calculate the error of this mode with respect to how close it is to being an eigenvalue of the real Hessian:

$$r = \sum_k c_k \cdot [\sigma_k - \lambda \cdot b_k]$$

Here, λ is the corresponding eigenvalue of \tilde{H}^m . r is the residual vector, giving the error for each vector element. It should be zero if the mode is an exact eigenvector of the true Hessian.

Since \tilde{H}^m may give multiple eigenvectors, several approximate modes will be generated during those iterations. Out of these, 1 mode is identified as the mode of interest according to the specified *tracking method* (page 167). If the residual of this mode has been minimized sufficiently, the procedure has converged. If not, we generate a new guess vector b_k^m . There are 2 algorithms for generating this new guess, set by `UpdateMethod` in the `ModeTracking` block:

Davidson method

The Davidson method uses a pre-conditioner D to generate a new guess mode from the residual vector of the mode selected by the tracking method:

$$b_k^m = D^{-1} \cdot r$$

This preconditioner is constructed from an approximation of the Hessian:

$$D = H_A - \lambda \cdot I$$

The Davidson method works reasonably well, but can have trouble converging if the approximate modes or the Hessian are too accurate. This results as the new vectors that are generated do not necessarily extend the span of the basis.⁶

vdVorst-Sleijpen-Jacobi-Davidson

This variant of the Jacobi-Davidson scheme from Sleijpen & vdVorst⁶ automatically makes the new guess vector orthogonal to the normal mode selected by the tracking method:

$$b_k^m = \left(\frac{Q^m D^{-1} r}{Q^m D^{-1} Q^m} \right) D^{-1} Q^m - D^{-1} r$$

The new vector is therefore guaranteed to extend the span of the basis as much as possible, and thus also eliminates the aforementioned issue with the Davidson method. In general, it is therefore recommended to use this Jacobi-Davidson method since it is found to converge faster, and be more reliable, as a result of yielding better guess modes.

⁶ G.L.G. Sleijpen, H.A. van der Vorst, *A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems*, *SIAM Journal on Matrix Analysis and Applications* 17, 401 (1996) (<https://doi.org/10.1137/S0895479894270427>)

Input

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeTracking
  ...
  ModeTracking
    HessianGuess [Unit | File | UFF | Inline]
    HessianInline # Non-standard block. See details.
    ...
  End
  HessianPath string
  UpdateMethod [JD | D | I]
  MaxIterations integer
  ...
  GramSchmidt [True | False]
  GramSchmidtIterations integer
  GramSchmidtTolerance float
End
End

```

There are 4 methods to obtain the approximate Hessian H_A , used by both update methods. They are set by HessianGuess:

HessianGuess [Unit | File | UFF | Inline]

UFF is the default, which generates the approximate Hessian using UFF. While this Hessian may not yield the correct modes by itself, it produces good results as a preconditioner since it correctly represents the molecular structure.

File will read the Hessian from an AMS output file, which can be specified in HessianPath. Using a Hessian from a more advanced method will generally yield better results for the Jacobi-Davidson method. The Davidson method will however experience difficulties with convergence as the Hessian becomes too accurate.⁶

Inline will read a Hessian specified in the input file, in the HessianInline block. This allows you to use Hessians generated in external programs:

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeTracking
  ModeTracking
    HessianGuess Inline
    # Approximate Hessian for H2O: 3 x nAtoms = 9 so 9x9 Hessian
    HessianInline
      0.62088786  0.00000000  0.00000000 -0.31044393  0.00000000 -0.
↪21902068 -0.31044393  0.00000000  0.21902068
      0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.
↪00000000  0.00000000  0.00000000  0.00000000
      0.00000000  0.00000000  0.00000000  0.32143213 -0.15284008  0.00000000 -0.
↪16071607  0.15284008  0.00000000 -0.16071607
      -0.31044393  0.00000000 -0.15284008  0.33598889  0.00000000  0.
↪18593038 -0.02554496  0.00000000 -0.03309030
      0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.
↪00000000  0.00000000  0.00000000  0.00000000
      -0.21902068  0.00000000 -0.16071607  0.18593038  0.00000000  0.
↪15761846  0.03309030  0.00000000  0.00309761
      -0.31044393  0.00000000  0.15284008 -0.02554496  0.00000000  0.
↪03309030  0.33598889  0.00000000 -0.18593038

```

(continues on next page)

(continued from previous page)

```

0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.
↪00000000  0.00000000  0.00000000  0.00000000
0.21902068  0.00000000  -0.16071607  -0.03309030  0.00000000  0.
↪00309761  -0.18593038  0.00000000  0.15761846
    End
    End
End

```

Unit uses the unit matrix. This is evidently not a good approximation for the Hessian, and is not intended to be used for proper Mode Tracking runs. However: using a poor approximation for the Hessian can result in basis vectors being generated that we would not obtain otherwise. Running Mode Tracking with this option can allow you to “probe” the vector space to obtain guesses for normal modes, which can be used as starting points for proper Mode Tracking calculations. It is however generally recommended to instead do e.g. a DFTB or UFF run if your goal is to obtain guess modes.

UpdateMethod [JD | D | I]

JD vdVorst-Sleijpen variant of Jacobi-Davidson (Mode tracking default).

D Davidson

I No preconditioner (VST default). This is not recommended for typical mode tracking applications, but is useful for a variation of mode tracking, *Vibronic-Structure Tracking* (page 198).

In later iterations, the basis B^m will become larger. In order to improve the guess modes even further, an iterative Gram-Schmidt procedure is used to orthogonalize the new guess mode to the existing basis. An iterative procedure is necessary to account for numerical noise.

GramSchmidt [True | False] Expert key. Sets whether to perform this Gram-Schmidt orthogonalization step. It is True by default.

GramSchmidtTolerance float Expert key. Sets the absolute tolerance for orthogonality of the basis. It is evaluated with respect to the norm of the overlap vector between the new guess mode and the basis of the previous iteration $||b_k^{mT} B^m||$.

GramSchmidtIterations Expert key. Sets the maximum number of allowed iterations during the Gram-Schmidt procedure.

The default settings for the Gram-Schmidt procedure should work for almost all systems.

MaxIterations integer Finally, the Mode Tracking input block contains the MaxIterations key. It sets the maximum allowed number of iterations that the Mode Tracking calculation may go through. If this number is reached, the calculation will stop even if convergence was not achieved. If no value is supplied, a default of $3N/2$ will be used. This is approximately the maximum number of iterations where the procedure remains computationally competitive with the construction of the full Hessian.

Additional input parameters

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeTracking
  Displacement float
  ...
  NormalModes
    ScanModes [True | False]
    ...

```

(continues on next page)

(continued from previous page)

```
End
End
```

Displacement float is the displacement stepsize (in Bohr) that is used for calculating frequencies, IR intensities and the Hessian projections through finite differences. The stepsize is provided for displacements along the Cartesian normal modes.

ScanModes [True | False] key (False by default) in the `NormalModes` vibrational analysis sub-block can be used to automatically run a *numerical frequencies* (page 159) calculation on the new modes Q after the Mode Tracking calculation has finished. Ritz vectors are obtained here as linear combinations of the guess modes, which in turn follow from finite difference calculations. This makes it possible for numerical errors to accumulate in the normal modes. Only in exceptional cases will these errors be significant, and running this additional refinement step is therefore only necessary if you need complete certainty that the results are accurate.

Input: Tracking methods

The `TrackingMethod` parameter allows you to select what property of the normal modes you want to track. At the end of each iteration, we obtain a set of approximate normal modes. The tracking method identifies which of these modes fits best for some criterion, and either returns this mode as the calculation result, or, if convergence was not achieved, uses it to generating a new basis mode for the next iteration. In general these methods are distinguished in 3 categories:

```
Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeTracking
  ModeTracking
    TrackingMethod [OverlapInitial, DifferenceInitial, FreqInitial, IRInitial,
                  OverlapPrevious, DifferencePrevious, FreqPrevious, IRPrevious,
                  HighestFreq, HighestIR, LowestFreq, LowestResidual]
    ...
  End
End
```

Mode Tracking

The original tracking methods focus on obtaining as accurate as possible a normal mode for the system. This class of tracking methods focuses either on accuracy of the mode, or obtaining modes with particular vibrational character:

TrackingMethod [OverlapInitial, DifferenceInitial, FreqInitial, OverlapPrevious, DifferencePrevious, FreqPrevious, IRPrevious, HighestFreq, HighestIR, LowestFreq, LowestResidual]

OverlapInitial is the default tracking method. Here, we choose the mode which resembles most closely the guess mode that was initially supplied b_1^m . This is done by choosing the mode which has the greatest overlap with the initial guess vector. This method allows us to direct the optimization towards modes that e.g. involve particular atoms or include particular bending/stretching vibrations.

OverlapPrevious instead chooses the mode which resembles closest the approximate mode of the previous iteration Q_k^m . This procedure allows a bit more flexibility in the optimization. Since we essentially “forget” about earlier iterations, this procedure allows the optimization to correct errors in the initial guess. (It is possible for instance that the initial guess included 2 different bond stretches which do not mutually occur in the true modes. This method will then converge quicker to a mode involving only 1 of these stretches, whereas `OverlapInitial` will take a much larger number of iterations to achieve this, if it does so at all.) Do note that this means that the final mode that you obtain does not necessarily represent the mode you initially supplied.

DifferenceInitial works the same as `OverlapInitial`, except that it chooses the mode which has the smallest norm for the difference vector between the initial mode and the approximate normal modes of this iteration. The use of the difference vector prioritizes deviations in the dominant parts of the vibrational character. E.g. if a mode consists primarily of a CO stretch, plus some minor vibrations in a carbon backbone, it may be desired to prioritize getting the correct force constant for the dominant CO stretch. This is achieved using these difference vector methods. In general, overlap methods still work well in these situations, and the use of difference methods should only be necessary in extreme cases.

DifferencePrevious is also the same as `DifferenceInitial` except for the use of the difference vector norm as the selection criterion.

FreqInitial chooses the mode with the frequency closest to that of the initial guess. This allows us to direct the tracking towards modes in a particular frequency region of the spectrum. Note that convergence for these frequency-based methods is slightly slower since the character of the mode itself is not included in the selection criteria, allowing for larger differences in the modes between iterations.

FreqPrevious is similar to `FreqInitial` except that we choose the mode with the frequency closest to that of the previous iteration. This allows the optimization more freedom to move away from the frequency region of the initial guess, and thus allows to correct somewhat for poor initial guesses.

HighestFreq chooses the mode with the highest frequency. This method can be used if it is desired to track particular characteristic high-frequency vibrations.

LowestResidual chooses the mode which has the smallest norm for the residual vector (see the ‘Convergence’ section below.) This method only focuses on obtain the most accurate mode, regardless of vibrational character or where it lies in the spectrum. This method should generally only be used as a pre-conditioner if you have very little information on what the normal modes should look like. (Since it is basically a non-directed optimization.) This method will then try and find the normal mode closest to your guess. The approximate normal mode obtained this way will most likely not have converged yet, but should give you an indication of what the normal modes may look like. You can use these modes to refine your initial guess, and then do a new Mode Tracking run using any of the other tracking parameters to obtain the desired mode. Although this strategy is possible, it is generally recommended to use an approximate method to get an initial guess for the normal modes instead (as shown in the *examples* (page 241)).

Intensity Tracking

This class of methods focuses on tracking modes based on their intensity in e.g. the infrared spectrum, rather than focusing on getting a mode with a particular type of vibration.

TrackingMethod [IRInitial, IRPrevious, HighestIR] `IRInitial` chooses the mode with the IR intensity closest to that of the initial guess. This constrains the optimization to modes which are IR active, a property that may be lost when using mode tracking update methods.

`IRPrevious` similarly chooses the mode with the IR intensity closest to that of the previous iteration. This allows the method some more flexibility in varying the intensity of the vibration, and thus works better if the initial guess is not that good.

`HighestIR` chooses the mode with the highest IR intensity. This option can be used to find the modes associated with sharp peaks in the IR spectrum.

With Intensity Tracking, we essentially add an additional requirement to the modes: they must have a particular IR intensity. This constrained search has different convergence characteristics than conventional mode tracking, which you should take into account when setting up the mode tracking calculations.

- The majority of modes will have near-zero IR intensity. If we use a near-zero IR intensity mode as our initial guess, and request `IRInitial` or `IRPrevious`, then we could be tracking any of one of these. Conversely,

convergence behavior will be poor since the generated basis modes are essentially random. If you are trying to obtain a high IR-intensity mode, use an IR-susceptible mode.

Note: In our conventional work-flow, we recommend starting mode tracking or refinement calculations from a set of approximate normal modes obtained from a semi-empirical or force-field method. Note however, that these method often do not produce accurate IR intensities. When selecting the initial guess mode, do **not** use the `IRRange` or related options in the `ModeSelect` block. This will cause you to miss vibrations which were incorrectly labeled with low IR intensity, or vice versa. Instead, rely on chemical intuition to identify the modes which contain commonly IR active vibrational components (such as C-O or N-H stretches). You can use `AMSSpectra` in the GUI to visualize the vibrational modes, to help you in this process.

- To allow the intensity tracking procedure to converge faster, it is recommended to use the `IRPrevious` tag instead of the `IRInitial` tag. As discussed earlier, the former allows more flexibility in the optimization procedure, which counters the rigidity imposed by the intensity constraint. Intensity tracking methods often need this additional flexibility in generating guess modes to converge to the desired modes.
- **Poor Initial Guesses:** During each iteration, we still use the mode tracking methods to generate new basis modes. These basis modes try to expand the span of the basis with respect to the vibrational character of the modes. Note that this expansion does not guarantee that we will expand the basis specifically in the sub-span of IR-susceptible vibrations. If the initial guess for intensity tracking is correct, we already start our search in the sub-span vicinity of the normal modes. Basis expansion is then more efficient and there is a high chance that new guess modes sample the IR characteristic vibrations. For intensity tracking it is therefore discouraged to use poor initial guess modes.
- `HighestIR` is considered a “pure” intensity tracking method, in that it is used specifically to target characteristics of the IR spectrum irrespective of the underlying vibrational character. Consequently, the normal mode character can vary a lot between iterations. In order to assure that the procedure converges to the desired modes, it is recommended to use sufficiently strict tolerances (see the *Convergence* section). If the tolerances are too lax, the program may consider the modes to be “good enough” based on residual minimization, even though there may be another mode with a higher IR intensity. For this reason it is generally recommended to use `ToleranceForNorm` values 1 order of magnitude lower than the default, or around 0.00005.

Input: Selecting modes

It is possible to track multiple modes in a single Mode Tracking calculation. The Mode Tracking task will then run the Mode Tracking algorithm for each mode in order.

The initial guess for the mode which will be tracked can be supplied in several ways. This is governed by `ModeInputFormat`:

```
Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeTracking
  NormalModes
    ModeInputFormat [File | Inline | Hessian]
    ModeFile string
    ModeInline # Non-standard block. See details.
    ...
  End
  ModeSelect
  ...
  End
  MassWeighInlineMode [True | False]
End
End
```

ModeInputFormat [File | Inline | Hessian]

Inline will make the module read the mode from the input file. If this option is selected, you can supply the mode in the `ModeInline` block. It is possible to supply multiple modes by adding additional `ModeInline` blocks. The modes are given with one line for the x,y,z-displacement per atom, and in the same order, as the `Atoms` block in `System`:

```
ModeTracking
  TrackedMode Inline
  ModeInline
    0.00000000  0.00000000  -0.03815965
   -0.18888544  0.00000000  0.30281066
    0.18888544  0.00000000  0.30281066
  End
  ModeInline
    0.00000000  0.00000000  -0.02243153
    0.32132452  0.00000000  0.17800237
   -0.32132452  0.00000000  0.17800237
  End
  ...
End
```

File will make the module read modes from an AMS or engine output file, specified by `ModePath`. Modes generated using DFTB can be read from the `dftb.rkf` file and optimised using Mode Tracking for example. When this option is selected, all the vibrational modes present in the file are read first. The `ModeSelect` block then specifies for which of these modes you want to perform the Mode Tracking calculation.

Hessian will generate modes as the eigenvectors of the approximate Hessian selected for the preconditioner in `HessianGuess`. This also allows modes to be generated for Hessians obtained from external programs. `ModeSelect` specifies which of the generated vibrational modes are selected for Mode Tracking.

- Settings for the `ModeSelect` block are discussed on the [main page](#) (page 171).

MassWeighInlineMode [True | False] decides whether the initial guess modes need to be mass-weighted (default True). As discussed above, Mode Tracking uses mass-weighted normal modes. In most cases, the normal modes are given in regular Cartesian coordinates however. By setting `MassWeighInlineMode` true, these Cartesian modes are converted into mass-weighted modes by the program. If you supply a mass-weighted mode through the `ModeInline` block however, you do not need the program to do the mass-weighting, and you should set `MassWeighInlineMode` false.

Input: Convergence

```
Task VibrationalAnalysis
VibrationalAnalysis
  Type ModeTracking
  ModeTracking
    ToleranceForNorm float
    ToleranceForResidual float
    ToleranceForBasis float
  End
End
```

In order to guide the Mode Tracking procedure, several convergence criteria are used:

ToleranceForNorm float is the absolute tolerance for convergence of the norm of the residual vector. The residual vector is a vector containing the error for each element of the normal mode, and we use the norm as a measure for the total error. If the total error is smaller than this threshold, we consider the mode to be a true

normal mode and we stop iterating. Since the value of this norm depends on the length of the residual vector hence the number of atoms in the system, this tolerance is scaled internally to the number of atoms. 0.0005 is used as a default value for which most systems will converge to reasonably accurate modes in not too many iterations. If you want a more accurate approximation, you can decrease this value by e.g. 1 order of magnitude. (Consider running using the default settings, and reading the norm at convergence from the logfile. The new norm can be chosen to be lower than this value to ‘force’ the method into another iteration.)

ToleranceForResidual float is the absolute tolerance for the maximum component of the residual vector. Particularly in larger systems, where the vibration may be dominated by a small number of atoms, the error associated with the vibration of the majority of atoms may be small (the scaled residual norm will be small). The error for the atoms involved in the vibration may be comparatively large then, which is why we also check convergence for the maximum component of the error. Note that both the norm and this max. error are checked simultaneously. By varying strictness of the criteria for the norm and the max. error separately, you can prioritize either the total vibration or more localized character.

ToleranceForBasis float checks that the basis mode generated in the previous iteration, through the (Jacobi-)Davidson method, contributes to the approximate normal mode. Since the approximate mode is taken as a linear combination of the basis modes, its linear expansion coefficient must be larger than this tolerance.

The iterative procedure is stopped in one of two cases. Either both the residual criteria are achieved, in which case the mode is deemed to be converged and the program exits normally. Alternatively, the basis criterion is met in which case a warning is broadcast indicating that the desired level of accuracy of the mode may not have been reached yet, but the basis has stopped expanding. The default values for these parameters should be applicable for most cases, but can be adjusted if needed. If stricter criteria are required, it is recommended to adjust both `ToleranceForNorm` and `ToleranceForResidual`.

6.2.7 Selecting modes

Mode Scanning, Mode Refinement and Mode Tracking as well as VG-FC Vibronic-Structure, VG-FC Vibronic-Structure Refinement and VG-FC resonance Raman all require a set of normal modes to operate on. For Mode Scanning these are the modes that you want to calculate the properties of, for Mode Refinement these modes form the basis modes, and for Mode Tracking these are the initial guess modes. For the VG-FC based methods these modes are the modes responsible for the vibronic coupling to the electronic excitation (in VG-FC Vibronic-Structure Refinement they are refined first).

Note: VG-FC Vibronic-Structure Tracking does not require any normal modes and as such does not support the `ModeSelect` (nor does it support the `NormalModes` block for that matter).

These methods provide options to load a large set of modes, after which the program will filter out the modes of interest. This is done according to the keys set in the `ModeSelect` block.

Note: The `ModeSelect` block is part of the `NormalModes` block of the Vibrational Analysis input. All Vibrational Analysis methods share this block, with the exception of VG-FC Vibronic-Structure Tracking. The methods for obtaining the set of modes that we will filter can differ per method. Particularly Mode Tracking features a lot of additional options, and the vibronic variants feature more specialized options.

Below is an overview of all the available options for the `ModeSelect` block as they appear in the basic vibrational analysis tools. The vibronic variants are discussed in more detail on their respective documentation pages.

The options below are not mutually exclusive.

<pre>ModeSelect DisplacementBound float</pre>

(continues on next page)

(continued from previous page)

```

FreqAndIRRange float_list
FreqRange float_list
Full Yes/No
HighFreq integer
HighIR integer
IRRange float_list
ImFreq Yes/No
LargestDisplacement integer
LowFreq integer
LowFreqNoIm integer
LowIR integer
ModeNumber integer_list
End

```

ModeSelect**DisplacementBound****Type** Float**Description** Vibronic Structure (Refinement), Resonance Raman: Select all modes with a dimensionless oscillator displacement greater than the specified value.**FreqAndIRRange****Type** Float List**Unit** cm-1 and km/mol**Recurring** True**Description** Specifies a combined frequency and IR intensity range within which all modes will be selected. First 2 numbers are the frequency range, last 2 numbers are the IR intensity range.**FreqRange****Type** Float List**Unit** cm-1**Recurring** True**Description** Specifies a frequency range within which all modes will be selected. 2 numbers: an upper and a lower bound. Calculating all modes higher than some frequency can be achieved by making the upper bound very large.**Full****Type** Bool**Default value** No**GUI name** All modes**Description** Select all modes. This only make sense for Mode Scanning calculations.**HighFreq****Type** Integer**GUI name** # High frequencies**Description** Select the N modes with the highest frequencies.

HighIR**Type** Integer**GUI name** # High IR**Description** Select the N modes with the largest IR intensities.**IRRange****Type** Float List**Unit** km/mol**Recurring** True**Description** Specifies an IR intensity range within which all modes will be selected. 2 numbers: an upper and a lower bound.**ImFreq****Type** Bool**Default value** No**GUI name** All imaginary frequencies**Description** Select all modes with imaginary frequencies.**LargestDisplacement****Type** Integer**Description** Vibronic Structure (Refinement), Resonance Raman: Select the N modes with the largest VG-FC displacement.**LowFreq****Type** Integer**GUI name** # Low frequencies**Description** Select the N modes with the lowest frequencies. Includes imaginary modes which are recorded with negative frequencies.**LowFreqNoIm****Type** Integer**GUI name** # Low positive frequencies**Description** Select the N modes with the lowest non-negative frequencies. Imaginary modes have negative frequencies and are thus omitted here.**LowIR****Type** Integer**GUI name** # Low IR**Description** Select the N modes with the smallest IR intensities.**ModeNumber****Type** Integer List**GUI name** Mode numbers**Description** Indices of the modes to select.

6.2.8 Thermodynamics (ideal gas)

The following thermodynamic properties are calculated by default whenever normal modes are computed: entropy, internal energy, constant volume heat capacity, enthalpy and Gibbs free energy. Translational, rotational and vibrational contributions are calculated for entropy, internal energy and constant volume heat capacity.

The results are written to the output file (section: “Statistical Thermal Analysis”) and to the engine binary results file (section: “Thermodynamics”).

The thermodynamic properties are computed assuming an ideal gas, and electronic contributions are ignored. The latter is a serious omission if the electronic configuration is (almost) degenerate, but the effect is small whenever the energy difference with the next state is large compared to the vibrational frequencies. The thermal analysis is based on the temperature dependent partition function. The energy of a (non-linear) molecule is (if the energy is measured from the zero-point energy)

$$\frac{E}{NkT} = \frac{3}{2} + \frac{3}{2} + \sum_j^{3N-6} \left(\frac{h\nu_j}{2kT} + \frac{h\nu_j}{kT(e^{h\nu_j/(kT)} - 1)} \right) - \frac{D}{kT}$$

The summation is over all harmonic ν_j , h is Planck’s constant and D is the dissociation energy

$$D = D_0 + \sum_j \frac{h\nu_j}{2}$$

Contributions from low (less than 20 1/cm) frequencies to entropy, heat capacity and internal energy are excluded from the total values, but they are listed separately (so the user can add them if they wish).

As an alternative to outright excluding low-frequency contributions, a correction scheme is available that is based on interpolating between harmonic oscillator and free rotor values⁷⁸ (Li/Head-Gordon and Grimme). It can greatly reduce the impact of the inaccuracies of the harmonic oscillator model on thermodynamic properties at these low frequencies. The scheme corrects vibrational contributions to entropies, internal energies and constant volume heat capacities. This correction is applied automatically and its results are printed separately (in the text output, the corrected terms are marked with the symbol (c)). When applied, the correction considers all real frequencies, including those less than 20 1/cm.

The interpolation for a corrected thermodynamic property f at pressure p and temperature T in terms of harmonic oscillator terms f_{HO} , free rotor terms f_{FR} , and interpolator terms x for each harmonic oscillator frequency ν_j is:

$$f(p, T) = \sum_j x(\nu_j) \cdot f_{HO}(p, T, \nu_j) + (1 - x(\nu_j)) \cdot f_{FR}(p, T, \nu_j)$$

$$x(\nu_j) = \frac{1}{1 + \left(\frac{\nu_0}{\nu_j}\right)^\alpha}$$

Where α is an arbitrary exponent and ν_0 is the harmonic oscillator frequency around which x interpolates, with $x = 0.5$ when $\nu_j = \nu_0$ and $x \approx 1.0$ when $\nu_j \gg \nu_0$.

While the free rotor terms used for internal energies and heat capacities are the standard ones, the terms used for entropies have to use the rotors’ moments of inertia μ_{FR} and symmetry σ , which formally cannot be calculated from harmonic frequencies alone. The correction scheme instead estimates each moment of inertia as being of a $\sigma = 1$ free rotor whose first excited state has an energy equal to the given $h\nu_j$. After this, each obtained moment of inertia μ_{FR} is modified by an averaging moment of inertia μ_{av} to avoid grossly overestimating entropies at very small frequencies (less than around 1 1/cm):

$$\mu = \frac{\mu_{FR} \cdot \mu_{av}}{\mu_{FR} + \mu_{av}}$$

⁷ Yi-Pei Li, Joseph Gomes, Shaama Mallikarjun Sharada, Alexis T. Bell, Martin Head-Gordon, *Improved Force-Field Parameters for QM/MM Simulations of the Energies of Adsorption for Molecules in Zeolites and a Free Rotor Correction to the Rigid Rotor Harmonic Oscillator Model for Adsorption Enthalpies*, J. Phys. Chem. C 2015, 119, 4, 1840-1850 (<https://doi.org/10.1021/jp509921r>)

⁸ Stefan Grimme, *Supramolecular Binding Thermodynamics by Dispersion-Corrected Density Functional Theory*, Chem. Eur. J., 18: 9955-9964 (<https://doi.org/10.1002/chem.201200497>)

Input options

```

Thermo
  Temperatures float_list
  Pressure float
  LowFrequencyCorrector
    Alpha float
    Frequency float
    MomentOfInertia float
  End
End

```

Thermo**Type** Block**Description** Options for thermodynamic properties (assuming an ideal gas). The properties are computed for all specified temperatures.**Temperatures****Type** Float List**Default value** [298.15]**Unit** Kelvin**Description** List of temperatures at which the thermodynamic properties will be calculated.**Pressure****Type** Float**Default value** 1.0**Unit** atm**Description** The pressure at which the thermodynamic properties are computed.**LowFrequencyCorrector****Type** Block**Description** Options for the dampener-powered free rotor interpolator that corrects thermodynamic quantities for low frequencies. See DOI:10.1021/jp509921r and DOI:10.1002/chem.201200497.**Alpha****Type** Float**Default value** 4.0**Description** The exponent term used in the dampener.**Frequency****Type** Float**Default value** 100.0**Unit** cm-1**Description** The frequency around which the dampener interpolates between harmonic oscillator and free rotor quantities.**MomentOfInertia**

Type Float**Default value** 1e-44**Unit** kg m²**GUI name** Averaging Moment of Inertia**Description** The moment of inertia used to restrict entropy results for very small frequencies (generally around less than 1 cm⁻¹).

Gibbs free energy change for a gas phase reaction

Here an example is given how to calculate the free energy change for a reaction. In the AMS output of a normal modes calculation you can find the electronic bonding energy and nuclear kinetic energies, at room temperature. Example part of the AMS output of a nonlinear molecule:

```

Zero-point energy (Hartree):      0.0333
...
...
Temp                               Transl      Rotat      └
↪Vibrat      Total
-----
↪--      -----
298.15      Entropy (cal/mol-K):      34.441      11.474      0.
↪137      46.052
Nuclear Internal Energy (kcal/mol):      0.889      0.889      20.
↪941      22.718
Constant Volume Heat Capacity (cal/mol-K):      2.981      2.981      0.
↪565      6.526

Summary of energy terms
                                hartree          eV          kcal/
↪mol          kJ/mol          -----          -----          -----
↪
Energy from Engine:      -0.743995039793930      -20.2451      -466.
↪86      -1953.36
Nuclear Internal Energy:      0.036203917534227      0.9852      22.
↪72      95.05
Internal Energy U:      -0.707791122259703      -19.2599      -444.
↪14      -1858.31
pV/n = RT:      0.000944186013486      0.0257      0.
↪59      2.48
Enthalpy H:      -0.706846936246217      -19.2343      -443.
↪55      -1855.83
-T*S:      -0.021880868282982      -0.5954      -13.
↪73      -57.45
Gibbs free energy:      -0.728727804529199      -19.8297      -457.
↪28      -1913.27

```

The Energy from Engine = -466.86 kcal/mol. It depends on the engine how this energy is calculated. In the ADF and BAND engines the energy is normally calculated with respect to (artificial) spherical averaged neutral atoms.

The Nuclear Internal Energy = zero point energy + 3 kT + small correction term = 22.72 kcal/mol. 3 kT = 3/2 kT for rotation, and 3/2 kT for translation (i.e. 1/2 kT for each degree of freedom). The small correction term is a term

due to the vibration partition function, depending on the temperature not only the ground state vibrational levels are occupied, see also the previous discussion.

The Internal Energy $U = \text{Energy from Engine} + \text{Nuclear Internal Energy} = -466.86 + 22.72 = -444.14$ kcal/mol. Gas phase $pV/n = RT = 8.314472 * 298.15 / 4184 = 0.59$ kcal/mol. The enthalpy $H = U + pV = -444.14 + 0.59 = -443.55$ kcal/mol. The Gibbs free energy $G = H - TS = -443.55 - 298.15 * 46.052/1000 = -457.28$ kcal/mol.

For a calculation of the free energy change for reaction (ΔG), you will have to do this for the reactant and product molecules, and add and subtract these energies, for each molecule proportional to the number of molecules that take place in the reaction. Application of ADF for obtaining enthalpy, entropy and Gibbs free energy can for instance be obtained in Refs.^{9,10}.

6.2.9 Moments of inertia

In case normal modes are computed in AMS, AMS also reports the moments of inertia of the molecule in units of amu bohr² (amu = atomic mass unit) and its corresponding principal axes.

6.2.10 Partial Vibrational Spectra (PVDOS)

The Partial Vibrational Spectra (also known as PVDOS) is computed by default whenever normal modes are requested. The PVDOS $P_{I,n}$ for atom I and normal mode n is defined as:

$$P_{I,n} = \frac{m_I |\vec{\eta}_{I,n}|^2}{\sum_p m_I |\vec{\eta}_{I,p}|^2}$$

where m_I is the nuclear weight of atom I , and $\vec{\eta}_{I,n}$ is the displacement vector for atom I in normal mode n .

Tip: The Partial Vibrational Spectra (PVDOS) can be visualized using the **AMSSpectra** GUI module (**Vibrations** → **Partial Vibrational Spectra (PVDOS)**). When plotting a partial vibrational spectrum, the IR intensity of normal modes is scaled by the corresponding PVDOS of the selected atoms.

The PVDOS matrix is not printed to the text output, but only saved to the engine binary output (.rkf) in the variable `Vibrations%PVDOS`.

6.3 Phonons

Collective oscillations of atoms around their equilibrium positions, giving rise to lattice vibrations, are called phonons. AMS can calculate phonon dispersion curves within standard harmonic theory, implemented with a finite difference method. Within the harmonic approximation we can calculate the partition function and therefore thermodynamic properties, such as the specific heat and the free energy.

See also:

Example: Phonons for graphene (page 301), *Example: Phonons with isotopes* (page 302), *Example: User-defined Brillouin zone for phonon dispersion* (page 304) and diamond lattice optimization and phonons tutorial

The calculation of phonons is enabled in the `Properties` block.

⁹ M. Swart, E. Rösler, and F. M. Bickelhaupt, *Proton affinities of maingroup-element hydrides and noble gases: Trends across the periodic table, structural effects, and DFT validation*, *Journal of Computational Chemistry* 27, 1486 (2006) (<https://doi.org/10.1002/jcc.20431>)

¹⁰ M. Swart, and F. M. Bickelhaupt, *Proton Affinities of Anionic Bases: Trends Across the Periodic Table, Structural Effects, and DFT Validation*, *Journal of Chemical Theory and Computation* 2, 281 (2006) (<https://doi.org/10.1021/ct0502460>).

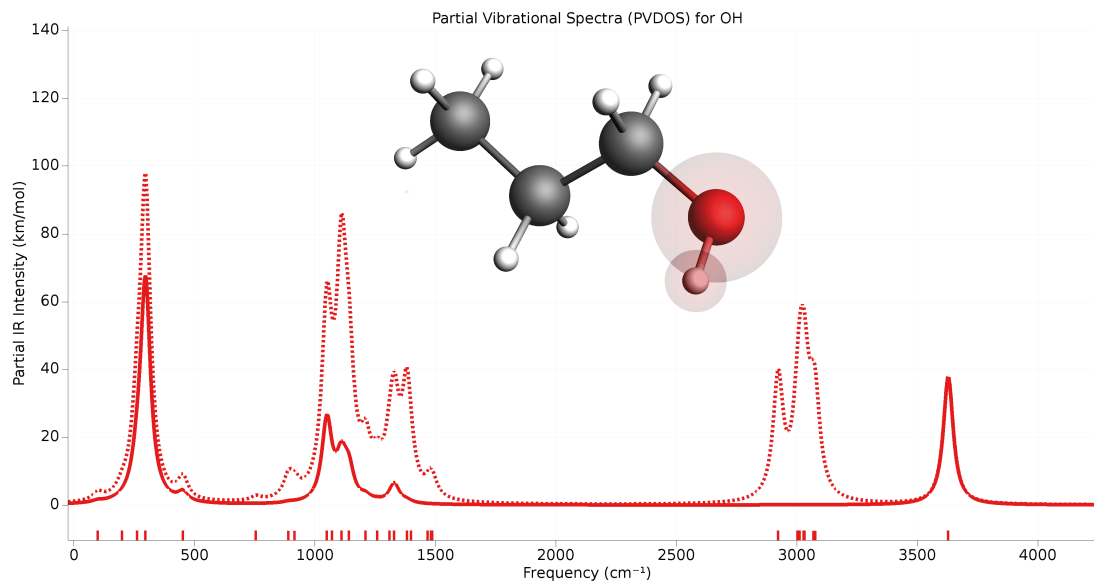


Fig. 6.1: Example of partial vibrational spectrum (PVDOS). The dotted line is the full IR spectrum of 1-propanol. The solid line is the PVDOS-scaled IR spectrum of the OH group (IR spectrum computed using GFN1-xTB).

```

Properties
  Phonons Yes/No
End

```

Note: Phonon calculations should be performed on optimized geometries, **including the lattice vectors**. This can be done by either using an already optimized system as input, or by combining the phonon calculation with the *geometry optimization task* (page 43) (you should set the `GeometryOptimization%OptimizeLattice` input option to `True`).

The details of the phonon calculations are configured in the `NumericalPhonons` block:

```

NumericalPhonons
  SuperCell # Non-standard block. See details.
  ...
End
  StepSize float
  DoubleSided Yes/No
  Interpolation integer
  NDosEnergies integer
  AutomaticBZPath Yes/No
  BZPath
    Path # Non-standard block. See details.
    ...
  End
End
Parallel
  nCoresPerGroup integer
  nGroups integer
  nNodesPerGroup integer
End

```

(continues on next page)

(continued from previous page)

End

NumericalPhonons**SuperCell****Type** Non-standard block**Description** Used for the phonon run. The super lattice is expressed in the lattice vectors. Most people will find a diagonal matrix easiest to understand.

The most important setting here is the super cell transformation. In principle this should be as large as possible, as the phonon bandstructure converges with the size of the super cell. In practice one may want to start with a 2x2x2 cell and increase the size of the super cell until the phonon band structure converges:

```
NumericalPhonons
  SuperCell
    2 0 0
    0 2 0
    0 0 2
  End
End
```

By default the phonon dispersion curves are computed for the standard path though the Brillouin zone (see <https://doi.org/10.1016/j.commatsci.2010.05.010>). One can request the a different path using the following keywords (for an example of how to specify a user-defined path see *Example: User-defined Brillouin zone for phonon dispersion* (page 304)):

```
NumericalPhonons
  AutomaticBZPath Yes/No
  BZPath
    Path # Non-standard block. See details.
    ...
  End
End
End
```

NumericalPhonons**AutomaticBZPath****Type** Bool**Default value** Yes**GUI name** Automatic BZ path**Description** If True, compute the phonon dispersion curve for the standard path through the Brillouin zone. If False, you must specify your custom path in the [BZPath] block.**BZPath****Type** Block**Description** If [NumericalPhonons%AutomaticBZPath] is false, the phonon dispersion curve will be computed for the user-defined path in the [BZPath] block. You should define the vertices of your path in fractional coordinates (with respect to the reciprocal lattice vectors) in the [Path] sub-block. If you want to make a jump in your path (i.e. have a discontinuous path), you need to specify a new [Path] sub-block.**Path**

Type Non-standard block

Recurring True

Description A section of a k space path. This block should contain multiple lines, and in each line you should specify one vertex of the path in fractional coordinates. Optionally, you can add text labels for your vertices at the end of each line.

Other keywords in the `NumericalPhonons` block modify the details of the numerical differentiation procedure and the accuracy of the results:

NumericalPhonons

StepSize

Type Float

Default value 0.04

Unit Angstrom

Description Step size to be taken to obtain the force constants (second derivative) from the analytical gradients numerically.

DoubleSided

Type Bool

Default value Yes

Description By default a two-sided (or quadratic) numerical differentiation of the nuclear gradients is used. Using a single-sided (or linear) numerical differentiation is computationally faster but much less accurate. Note: In older versions of the program only the single-sided option was available.

Interpolation

Type Integer

Default value 100

Description Use interpolation to generate smooth phonon plots.

NDosEnergies

Type Integer

Default value 1000

Description Nr. of energies used to calculate the phonon DOS used to integrate thermodynamic properties. For fast compute engines this may become time limiting and smaller values can be tried.

The numerical phonon calculation supports AMS' *double parallelization* (page 18), which can perform the calculations for the individual displacements in parallel. This is configured automatically, but can be further tweaked using the keys in the `NumericalPhonons%Parallel` block:

```
NumericalPhonons
  Parallel
    nCoresPerGroup integer
    nGroups integer
    nNodesPerGroup integer
  End
End
```

NumericalPhonons

Parallel**Type** Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup**Type** Integer**GUI name** Cores per group**Description** Number of cores in each working group.**nGroups****Type** Integer**GUI name** Number of groups**Description** Total number of processor groups. This is the number of tasks that will be executed in parallel.**nNodesPerGroup****Type** Integer**GUI name** Nodes per group**Description** Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

6.4 (Resonance) Raman

6.4.1 Raman

In this method the Raman scattering spectrum is calculated from the geometrical derivatives of the frequency-dependent polarizability. Engine ADF is required. Raman scattering intensities and depolarization ratios for all or a selected number of molecular vibrations at a certain laser frequency can be calculated. The Raman scattering calculation is very similar to an IR intensity calculation. In fact, all IR output is automatically generated as well. At all distorted geometries the dipole polarizability tensor is calculated. This is time-consuming and is only feasible for small molecules.

```
Properties
  Raman Yes/No
End
```

```
Raman
  IncidentFrequency float
  FreqRange float_list
End
```

If a `FreqRange` is included the Raman intensities are calculated for a range of vibrational frequencies only. Using this option is a fast alternative for calculating all Raman intensities.

Properties

Raman**Type** Bool**Default value** No**Description** Requests calculation of Raman intensities for vibrational normal modes.**Raman****IncidentFrequency****Type** Float**Default value** 0.0**Unit** eV**Description** Frequency of incident light.**FreqRange****Type** Float List**Unit** cm-1**Recurring** True**GUI name** Frequency range**Description** Specifies a frequency range within which all modes will be scanned. 2 numbers: an upper and a lower bound.

6.4.2 Resonance Raman: excited-state finite lifetime

Resonance Raman spectroscopy uses incident light with a wavelength close to that of an electronic transition. In this method (Ref.¹¹) the resonance Raman-scattering (RRS) spectra is calculated from the geometrical derivatives of the frequency-dependent polarizability. Engine ADF is required. The polarizability derivatives are calculated from resonance polarizabilities by including a finite lifetime (phenomenological parameter) of the electronic excited states.

```
Raman
  FreqRange float_list
  IncidentFrequency float
  LifeTime float
End
```

Raman**FreqRange****Type** Float List**Unit** cm-1**Recurring** True**GUI name** Frequency range**Description** Specifies a frequency range within which all modes will be scanned. 2 numbers: an upper and a lower bound.**IncidentFrequency**

¹¹ L. Jensen, L. Zhao, J. Autschbach and G.C. Schatz, *Theory and method for calculating resonance Raman scattering from resonance polarizability derivatives*, *Journal of Chemical Physics* 123, 174110 (2005) (<https://doi.org/10.1063/1.2046670>)

Type Float**Default value** 0.0**Unit** eV**Description** Frequency of incident light.**LifeTime****Type** Float**Default value** 0.0**Unit** hartree**Description** Specify the resonance peak width (damping) in Hartree units. Typically the lifetime of the excited states is approximated with a common phenomenological damping parameter. Values are best obtained by fitting absorption data for the molecule, however, the values do not vary a lot between similar molecules, so it is not hard to estimate values. A typical value is 0.004 Hartree.

It is similar to the simple excited-state gradient approximation method (see next section) if only one electronic excited state is important, however, it is not restricted to only one electronic excited state. In the limit that there is only one possible state in resonance the two methods should give more or less the same results. However, for many states and high-energy states and to get resonance Raman profiles (i.e., Raman intensities as a function of the energy of the incident light beam) this approach might be more suitable. The resonance Raman profiles in this approach are averaged profiles since vibronic coupling effects are not accounted for.

6.4.3 Resonance Raman: VG-FC

According to a the time-dependent picture of resonance-Raman (RR) scattering the relative intensities of RR scattering cross sections are, under certain assumptions, proportional to the square of the excited-state energy gradients projected onto the ground-state normal modes of the molecule (see Ref.¹²). For an alternative implementation of RR scattering using a finite lifetime of the excited states, and a discussion of some of the differences, see the previous section. Engine ADF or DFTB is required.

The vertical gradient Franck-Condon (VG-FC) method, also called the Independent Mode Displaced Harmonic Oscillator (IMDHO) model, we use to calculate vibrationally resolved absorption spectra can also be applied to the calculation of resonance Raman spectra. In resonance Raman spectroscopy a molecule is excited from its ground state to some electronically excited state. After a short period of time, the molecule then relaxes back to its electronic ground state. However, when doing so, it might end up in a different vibrational state than it started off in. The result is an energy difference between the incident and emitted photon. One can then plot the intensity for different energy differences to produce what is known as a Raman spectra. Resonance Raman spectroscopy uses incident light with a wavelength close to that of an electronic transition.

AMS supports the calculation of such spectra by modeling the vibronic coupling of electronic transitions using the VG-FC model. This model is discussed also on the *Vibronic-Structure* (page 196) documentation page. Here we will discuss the modifications necessary to use the VG-FC model for resonance Raman spectroscopy. It is worth noting that this VG-FC resonance Raman application does not support the mode selective options. As a result the VG-FC Resonance Raman application will always first perform a full frequency analysis to obtain its normal modes.

Theory

While the basic theory behind the VG-FC model is explained in detail on the *Vibronic-Structure* (page 196) documentation page, we will briefly summarize the most important points here, as well as the modifications necessary for

¹² J. Neugebauer, E.J. Baerends, E. Efremov, F. Ariese and C. Gooijer, *Combined Theoretical and Experimental Deep-UV Resonance Raman Studies of Substituted Peryenes*, *Journal of Physical Chemistry A* 109, 2100 (2005) (<https://doi.org/10.1021/jp045360d>)

its application to resonance Raman spectroscopy. It applies the harmonic approximation to both the ground state and excited state PES and then goes on to assume that neither frequency changes nor normal mode rotations occur. Thus the excited state PES is a shifted version of the ground state PES. We do not include temperature effects (so all initial states will be ground states) and work at the Franck-Condon point. Under these assumptions, the Raman polarizability of a particular excited state n , for a transition between initial and final vibrational states I and F can be written as:

$$(\alpha_{n,ij})_{F \leftarrow I} = \mu_{n,i} \mu_{n,j} \int_0^\infty \langle F | I_n(t) \rangle e^{i[\omega - (E_{n,0} - E_{m,0})]t} \cdot e^{-\Gamma t} dt$$

Here, i, j label the components of the polarizability tensor and $\langle F | I_n(t) \rangle$ denotes the overlap of the initial state I , propagated along the excited state PES with the final state F . Under the *assumptions of the VG-FC model* (page 188), this overlap is equal to:

$$\langle F | I_n(t) \rangle = \prod_{j=1}^{N_{modes}} \left\{ \frac{(-1)^{m_j} \Delta_{n,j}^{m_j}}{2^{m_j/2} m_j!} (1 - e^{-i\omega_j t})^{m_j} \right\} \exp \left[-\frac{\Delta_{n,j}^2}{2} (1 - e^{-i\omega_j t}) \right]$$

Where the m_j denote the excitation number of mode j in final state F . For a more detailed discussion, we refer to¹³. The only parameters that appear in our expression are the dimensionless oscillator displacements $\Delta_{n,j}$ that represent displacement of the excited state PES along normal mode j . Under the simplifying assumptions of the VG-FC, these can be obtained from the ground state normal modes and a single excited state gradient. The Raman intensity is then proportional to the square of the polarizabilities:

$$\sigma(\omega)_{F \leftarrow I} \propto \sum_{i,j} \left| \sum_n (\alpha_{n,ij})_{F \leftarrow I} \right|^2$$

A spectrum is then generated by including various different final states F , which are defined by different combinations of normal mode excitation numbers, and assigning a relative intensity to each transition equal to the above expression. AMS only supports spectra which display relative intensities so the results are plotted in arbitrary units and are normalized such that the largest peak reaches an intensity of 1.

Input

The calculation setup for resonance Raman spectra largely proceeds as it does for *Absorption Spectra* (page 196). We need a set of ground state normal modes as well as an excited state gradient. The former are calculated at the start using the selected AMS engine, or, in case the user has a pre-calculated set of normal modes, these can be read from a .rkf file using the `ModeFile` key in the `NormalModes` sub-block. In this latter case, the engine is not used. The `ModeSelect` block can be used to select specific modes from the full set of normal modes for which the spectrum should be calculated. For details see the *Mode Select* (page 171) documentation on the main page. If one simply wants the spectrum for the full set of normal modes, the `Full` key in the `ModeSelect` block can be set to `True`. The excited state information is passed to the application via the `ExcitationSettings` block.

Another point to note is that since our states are labeled by discrete indices we will be calculating stick spectra (which can be homogeneously broadened in `amsspectra`). By contrast, the absorption spectra produced by `VibronicStructure` are raw `x,y` data. Due to this difference in nature of the Raman spectrum compared to the absorption spectrum, this method uses the `ResonanceRaman` block for input options related to its spectrum (as opposed to the `AbsorptionSpectrum` block).

The `ExcitationSettings` block is discussed on the *Vibronic-Structure* (page 204) page. One important difference with the latter is that Resonance Raman calculations are supported for more than one excitation at once. This is more important for the case of Raman spectra as the intensity associated with a set of transitions is not equal to the sum of their individual intensities (we sum over electronic states n before we square the polarizabilities). Here we will address settings specific to the Raman spectrum, all of which can be found in the `ResonanceRaman` block. A short example of how a typical input file might look is included at the end of this section.

¹³ T. Petrenko and F. Neese, *Analysis and prediction of absorption band shapes, fluorescence band shapes, resonance Raman intensities, and excitation profiles using the time-dependent theory of electronic spectroscopy* *The Journal of Chemical Physics* 127, 164319 (2007) (<https://doi.org/10.1063/1.2770706>)


```

Task VibrationalAnalysis
VibrationalAnalysis
  Task ResonanceRaman
  ResonanceRaman
    IncidentFrequency float
    LifeTime float
    RamanOrder integer
    RamanRange float_list
    MaximumStates integer
  End
  ...
End

```

IncidentFrequency float Frequency of incident light.

LifeTime float sets the value of Γ (in Hartree) that controls the exponential damping in our integral. This phenomenological parameter can be interpreted as the (inverse) life time of the Raman excited state and can be used to help the results agree with experiment. The default value of $4.5e-4$ is on the low end of reasonable values but should provide a good starting point for most cases.

RamanOrder integer determines the set of final states and overtones to be included in the spectrum. It is an integer and the application considers only final states such that the sum of excitation numbers of all normal modes is less than or equal to this number. Setting this to 1 means we only include the fundamental band.

RamanRange float_list this keyword specifies the frequency range (in cm^{-1}) the Raman shift is restricted to lie in. This prevents us from including excessively many states and overtones for high frequency modes. The default is $[0, 2000] cm^{-1}$ but this can be changed to whatever is desired.

MaximumStates integer Expert key. Due to the combinatorial explosion of included final states that occurs for combinations of large values of the raman order, large molecules and wide spectrum ranges, there is a maximum number of final states that can be included in the spectrum. This is to prevent the program from using excessive amounts of memory/computation times. The user can set this number using the `MaximumStates` key but this should be done with caution.

Finally we give an example of a typical `VibrationalAnalysis` block for a resonance Raman calculation. This also gives an idea of how the settings that were not explicitly mentioned above work:

```

VibrationalAnalysis
  Type ResonanceRaman
  NormalModes
    ModeSelect
      Full True
    End
  End
  ExcitationSettings
    ExcitationInfo File
    ExcitationFile ./your_excitation/dftb.rkf
    Singlet
      A 1 2 4
    End
  End
  ResonanceRaman
    RamanOrder 3
    RamanRange 0.0 3000.0
  End
End

```

6.5 VROA: (Resonance) vibrational Raman optical activity

The normal and resonance VROA spectra are calculated from geometric derivatives of the different generalized polarizabilities obtained using linear response theory which may include a damping term to account for the finite lifetime. Engine ADF is required. These polarizabilities are the electric dipole - electric dipole polarizability, the electric dipole - magnetic dipole polarizability, and the the electric dipole - electric quadrupole polarizability. For resonance VROA one should include a finite lifetime.

```
Properties
  VROA Yes/No
End
```

```
Raman
  FreqRange float_list
  IncidentFrequency float
  LifeTime float
End
```

Properties

VROA

Type Bool

Default value No

Description Requests calculation of VROA for vibrational normal modes.

Raman

FreqRange

Type Float List

Unit cm-1

Recurring True

GUI name Frequency range

Description Specifies a frequency range within which all modes will be scanned. 2 numbers: an upper and a lower bound.

IncidentFrequency

Type Float

Default value 0.0

Unit eV

Description Frequency of incident light.

LifeTime

Type Float

Default value 0.0

Unit hartree

Description Specify the resonance peak width (damping) in Hartree units. Typically the lifetime of the excited states is approximated with a common phenomenological damping parameter. Values are best obtained by fitting absorption data for the molecule, however, the values do

not vary a lot between similar molecules, so it is not hard to estimate values. A typical value is 0.004 Hartree.

6.5.1 Engine ADF

In the ADF engine a method is implemented to calculate both on- and off-resonance vibrational Raman optical activities (VROAs) of molecules using time-dependent density functional theory, see Ref.¹⁴. This is an extension of a method to calculate the normal VROA by including a finite lifetime of the electronic excited states in all calculated properties. The method is based on a short-time approximation to Raman scattering and is, in the off-resonance case, identical to the standard theory of Placzek. The normal and resonance VROA spectra are calculated from geometric derivatives of the different generalized polarizabilities obtained using linear response theory which includes a damping term to account for the finite lifetime. Gauge-origin independent results for normal VROA have been ensured using either the modified-velocity gauge or gauge-included atomic orbitals. In ADF2016 the velocity gauge tensors required for the calculation of VROA are now correctly calculated with the life time damping parameter. With these complex tensors fixed, resonance VROA intensities are now origin invariant in the velocity gauge, see also Ref.¹⁵.

6.6 VCD: Vibrational Circular Dichroism

Vibrational circular dichroism (VCD) is the differential absorption of left and right circularly polarized infrared light by vibrating molecules. Most engines can be used to calculate VCD with the approximate Atomic polar tensor (APT) model. Engine ADF is required for the more accurate analytical VCD.

```
Properties
  VCD Yes/No
End
```

Properties

VCD

Type Bool

Default value No

Description Requests calculation of VCD for vibrational normal modes.

VCDtools (page 238) is a program that can be used to do an analysis of the VCD spectrum. *VCDtools* (page 238) can be used with the AMS-GUI module AMSspectra.

6.6.1 Atomic polar tensor (APT) model

In the so-called atomic polar tensor (APT) model the atomic axial tensors (AATs) can be calculated from electric dipole gradients. Note that the APT model may not be very reliable for predicting VCD bands and its implementation should not be blindly applied beyond a quick assessment. Results using the engine DFTB can be found in¹⁶.

For the engines BAND and DFTB only the APT model can be used. In case of the engine ADF the default is to calculate the VCD analytically, see next section. One can calculate VCD using the APT model with the ADF engine if one includes:

¹⁴ L. Jensen, J. Autschbach, M. Krykunov, and G.C. Schatz, *Resonance vibrational Raman optical activity: A time-dependent density functional theory approach*, *Journal of Chemical Physics* 127, 134101 (2007) (<https://doi.org/10.1063/1.2768533>)

¹⁵ D.V. Chulhai and L. Jensen, *Simulating Surface-Enhanced Raman Optical Activity Using Atomistic Electrodynamics-Quantum Mechanical Models*, *Journal of Physical Chemistry A* 118, 9069 (2014) (<https://doi.org/10.1021/jp502107f>)

¹⁶ T.Q. Teodoro, M.A.J. Koenis, R. Rüger, S.E. Galembeck, W.J. Buma, V.P. Nicu, L. Vissche, *Use of Density Functional Based Tight Binding Methods in Vibrational Circular Dichroism*, *Journal of Physical Chemistry A* 122, 9435 (2018) (<https://doi.org/10.1021/acs.jpca.8b08218>)

```
NormalModes
  Hessian Numerical
End
```

6.6.2 Analytical VCD in ADF

In the ADF engine the VCD intensities are calculated using Stephens' equations for VCD. For the calculation of the atomic axial tensors (AATs), analytical derivatives techniques and London atomic orbitals (the so called GIAO) are employed. As a result the calculated rotational strengths are origin independent, and therefore the common origin gauge is used¹⁷.

New in AMS2020 is that one can calculate analytical VCD also for open-shell systems in a spin-unrestricted calculation.

Calculation of the AATs requires an analytical frequencies calculation. This limits the choice of functionals that can be used for VCD calculations.

The accuracy of the vibrational rotational strengths are determined by the accuracy of the harmonic force field, atomic polar tensors (APT) and AATs. The most critical parameter being the harmonic force field. Thus, for a fair comparison with experimental data, accurate geometries and functionals that yield accurate force fields (e.g. BP86, OLYP, etc) should be used. Our tests showed that the BP86 functional in combination with TZP basis sets is always a safe choice. For a comparison of VCD spectra calculated with various functionals (e.g. BP86, OLYP, BLYP, B3PW91 and B3LYP) see¹⁷. Regarding the geometries, we recommend the following strict settings, 10^{-4} for the geometry convergence of the gradients, and BeckeGrid quality good. The default settings should be used for the calculation of the frequencies.

By default, only the vibrational rotational strengths are printed in the AMS output file. For a deeper insight regarding the origin of the VCD intensity of a given normal mode one can use the auxiliary program *VCDtools* (page 238). *VCDtools* (page 238) can be used with the ADF-GUI module ADFspectra.

¹⁷ V.P. Nicu J. Neugebauer S.K. Wolff and E.J. Baerends, *A vibrational circular dichroism implementation within a Slater-type-orbital based density functional framework and its application to hexa- and hepta-helicenes*, *Theoretical Chemical Accounts* 119, 245 (2008) (<https://doi.org/10.1007/s00214-006-0234-x>)

VIBRATIONALLY RESOLVED ELECTRONIC SPECTRA

7.1 AH-FC: Adiabatic Hessian Franck-Condon

Electronic spectra, such as absorption, emission, phosphorescence, and ionization, may contain a vibrational structure. In the Adiabatic Hessian Franck-Condon (AH-FC) method one needs to do a frequency calculation both at the ground state as well as the excited state of interest. The model makes the following assumptions:

- 1. It employs the adiabatic (Born-Oppenheimer) approximation and treats the nuclei as moving in an effective potential defined by the electronic configuration.
- 2. It works at the Franck-Condon point and assumes that the transition occurs at the ground state equilibrium structure for absorption and at the excited state equilibrium structure for emission.
- 3. It applies the harmonic approximation to both the ground and excited state potential energy surfaces.

Franck-Condon factors can be calculated for the transition between the two electronic states, which can be done with the *FCF Module* (page 189), described below. These Franck-Condon factor can then be used to predict the relative intensities of absorption or emission lines in the electronic spectra. Note that the Herzberg-Teller effect is not taken into account.

In the vertical gradient Franck-Condon (VG-FC) method some extra assumptions are made compared to the AH-FC method. This approach is particularly effective for large molecules as it shows linear scaling with the number of normal modes.

- See the documentation of the *VG-FC method* (page 196).

7.1.1 FCF module: Franck-Condon Factors

fcf is an auxiliary program which can be used to calculate Franck-Condon factors from two vibrational mode calculations¹.

fcf requires an ASCII input file where the user specifies the .rkf files from two vibrational mode calculations, carried out for two different electronic, spin or charge states of the same molecule. These calculations can be either numerical or analytical. The number of vibrational quanta that have to be taken into account for both states in the evaluation of the Franck-Condon factors have to be specified.

fcf produces a (binary) KF file TAPE61, which can be inspected using the KF utilities. Furthermore, *fcf* writes the frequencies, vibrational displacements and electron-phonon couplings for both states too the standard output, including any error messages.

¹ J.S. Seldenthuis, H.S.J. van der Zant, M.A. Ratner and J.M. Thijssen, *Vibrational Excitations in Weakly Coupled Single-Molecule Junctions: A Computational Analysis*, *ACS Nano* 2, 1445 (2008) (<https://doi.org/10.1021/nn800170h>)

Theory

Franck-Condon factors are the squares of the overlap integrals of vibrational wave functions. Given a transition between two electronic, spin or charge states, the Franck-Condon factors represent the probabilities for accompanying vibrational transitions. As such, they can be used to predict the relative intensities of absorption or emission lines in spectroscopy or excitation lines in transport measurements.

When a molecule makes a transition to another state, the equilibrium position of the nuclei changes, and this will give rise to vibrations. To determine which vibrational modes will be active, we first have to express the displacement of the nuclei in the normal modes:

$$\mathbf{k} = \mathbf{L}'^T \mathbf{m}^{1/2} (\mathbf{B}_0 \mathbf{x}_0 - \mathbf{x}'_0)$$

Here, \mathbf{k} is the displacement vector, \mathbf{L} is the normal mode matrix, \mathbf{m} is a matrix with the mass of the nuclei on the diagonal, \mathbf{B} is the zero-order axis-switching matrix and \mathbf{x}_0 is the equilibrium position of the nuclei. For free molecules, depending on symmetry constraints, the geometries of both states may have been translated and/or rotated with respect to each other. To remove the six translational and rotational degrees of freedom, we can center the equilibrium positions around the center of mass and rotate one of the states to provide maximum overlap. The latter is included with the zero-order axis-switching matrix \mathbf{B} , implemented according to².

When we have obtained the displacement vector, it is trivial to calculate the dimensionless electron-phonon couplings. They are given by:

$$\lambda = (\mathbf{\Gamma}/2)^{1/2} \mathbf{k}$$

Here, $\mathbf{\Gamma} = 2\pi\omega/h$ is a vector containing the reduced frequencies.³ The Huang-Rhys factor \mathbf{g} is related to λ as:

$$\mathbf{g} = \lambda^2$$

The reorganization energy per mode is

$$\mathbf{E} = (h/2\pi) * \omega \lambda^2$$

When the displacement vector \mathbf{k} , the reduced frequencies $\mathbf{\Gamma}$ and $\mathbf{\Gamma}'$, and the Duschinsky rotation matrix $\mathbf{J} = \mathbf{L}'^T \mathbf{B}_0 \mathbf{L}$ have been obtained, the Franck-Condon factors can be calculated using the two-dimensional array method of Ruhoff and Ratner³.

There is one Franck-Condon factor for every permutation of the vibrational quanta over both states. Since they represent transition probabilities, all Franck-Condon factors of one state which respect to one vibrational state of the other state must sum to one. Since the total number of possible vibrational quanta, and hence the total number of permutations, is infinite, in practice we will calculate the Franck-Condon factors until those sums are sufficiently close to one. Since the number of permutations rapidly increases with increasing number of vibrational quanta, it is generally possible to already stop after the sum is greater than about two thirds. The remaining one third will be distributed over so many Franck-Condon factors that their individual contributions are negligible.

In the limiting case of one vibrational mode, with the same frequency in both states, the expression for the Franck-Condon factors of transitions from the ground vibrational state to an excited vibrational state are given by the familiar expression:

$$|l_{0,n}|^2 = e^{-\lambda^2} \lambda^{2n} / n!$$

² G.M. Sando and K.G. Spears, *Ab Initio Computation of the Duschinsky Mixing of Vibrations and Nonlinear Effects*, *Journal of Physical Chemistry A* 105, 5326 (2001) (<https://doi.org/10.1021/jp004230b>)

³ P.T. Ruhoff and M.A. Ratner, *Algorithms for computing Franck-Condon overlap integrals*, *International Journal of Quantum Chemistry* 77, 383 (2000) ([https://doi.org/10.1002/\(SICI\)1097-461X\(2000\)77:1%3C383::AID-QUA38%3E3.0.CO;2-0](https://doi.org/10.1002/(SICI)1097-461X(2000)77:1%3C383::AID-QUA38%3E3.0.CO;2-0))

Input

The input for *fcf* is keyword oriented and is read from the standard input. *fcf* recognizes several keywords, but only two have to be specified to perform the calculation. All input therefore contains at least two lines of the following form:

```
$AMSBIN/fcf << eor
  STATES state1 state2
  QUANTA 11 12
eor
```

Documentation for all keys of *fcf*:

```
$AMSBIN/fcf << eor
  States state1 state2
  Quanta 11 12
  Lambda float
  Modes first last
  Rotate [True | False]
  Translate [True | False]
  Spectrum FreqMin FreqMax NFreq
eor
```

Lambda

Type Float

Default value 0.0

Description Optional. The minimum value of the electron-phonon coupling for a mode to be taken into account in the calculation. Together with the MODES option, this provides a way to significantly reduce the total number of Franck-Condon factors. As with the MODES option, always check if the results do not change too much.

Modes

Type Integer List

Default value [0, 0]

Description Optional. The first and last mode to be taken into account in the calculation. By default, all modes are taken into account. This option can be used to effectively specify an energy range for the Franck-Condon factors. When using this options, always check if the results (electron-phonon couplings, ground state to ground overlap integral, average sum of Franck-Condon factors, etc.) do not change too much.

Quanta

Type Integer List

Default value [0, 0]

Description The maximum number of vibrational quanta to be taken into account for the two states. Franck-Condon factors will be calculated for every permutation of up to and including 11/12 quanta over the vibrational modes.

Rotate

Type Bool

Default value No

Description Optional, recommended to be included. Rotate the geometries to maximize the overlap of the nuclear coordinates.

Spectrum

Type Float List

Description Optional. If included, the vibrational spectrum is calculated. A histogram of the spectrum is calculated for the frequency range defined by three numbers: FreqMin (minimum frequency for which the spectrum is calculated), FreqMax (maximum frequency for which the spectrum is calculated), NFreq (number of frequencies for which the spectrum is calculated). Usage example: 'Spectrum 0.0 2.0E4 1000'

States

Type String

Description The filenames of two results files of a numerical or analytical frequency calculation. The calculations must have been performed on the same molecule, i.e. the type, mass and order of occurrence of all the atoms (or fragments) has to be the same in both files.

Translate

Type Bool

Default value No

Description Optional, recommended to be included. Move the center of mass of both geometries to the origin.

Only a few keys from the .rkf file are used for the calculation of the Franck-Condon factors. Disk space usage can be significantly reduced by extracting just these keys from the .rkf file before further analysis. The following shell script will extract the keys from the KF file specified by the first argument and store them in a new KF file specified by the second argument using the *cpkf* utility:

```
#!/bin/sh
cpkf $1 $2 General Molecule Vibrations
```

Result: TAPE61

After a successful calculation, *fcf* produces a TAPE61 KF file. All results are stored in the Fcf section:

contents of TAPE61	comments
firstmode, lastmode	the first and last vibrational mode taken into account
lambda	the minimum value of the electron-phonon coupling
maxl1, maxl2	maximum level (or maximum number of vibrational quanta) in both states
translate, rotate	whether the TRANSLATE and ROTATE options were specified in the input
natoms	number of atoms in the molecule
mass	atomic mass vector (\mathbf{m})
xyz1, xyz2	equilibrium geometries of both states (\mathbf{x}_0 and \mathbf{x}_0')
b0	zero-order axis-switching matrix matrix (\mathbf{B}_0)
nmodes	number of vibrational modes with a non-zero frequency
gamma1, gamma2	reduced frequencies of both states (Γ and Γ')
lmat1, lmat2	mass-weighted normal modes of both states (\mathbf{L} and \mathbf{L}')
jmat	Duschinsky rotation matrix (\mathbf{J})
kvec1, kvec2	displacement vectors for both states (\mathbf{k} and \mathbf{k}' , kvec1 is used for the calculation of the Franck-Condon factors)
lambda1, lambda2	electron-phonon couplings for both states (λ and λ')
maxp1, maxp2	maximum number of permutations of maxl1/maxl2 quanta over the vibrational modes
i0	ground state to ground state overlap integral ($I_{0,0}$)
freq1, freq2	frequencies of every permutation of the vibrational quanta for both states
fcf	maxp1 by maxp2 Franck-Condon factor matrix
fcsum1, fcsum2	average sum of the Franck-Condon factors for both states

In addition to producing a binary TAPE61 file, *fcf* also writes the frequencies, displacement vectors and electron-phonon couplings for both states to the standard output.

7.1.2 FCF example absorption and fluorescence

In this example it is assumed that the molecule has a singlet ground state S_0 , and the interesting excited state is the lowest singlet excited state S_1 . First one needs to do the ground state geometry optimization, followed by a frequency calculation. For the ground state frequency calculation we will use `AMS_JOBNAME=S0`. Next one needs to do an excited state geometry optimization. Here it is assumed that the lowest singlet excited state S_1 is of interest:

```
AMS_JOBNAME=S1_GEO $AMSBIN/ams <<eor
...
Task GeometryOptimization
Engine ADF
...
Excitation
  Onlysing
  Lowest 1
End
ExcitedGO
  State A 1
  Singlet
End
EndEngine
eor
```

To get the frequencies for this excited state, numerical frequencies need to be calculated, at the optimized geometry of the first excited state:

```
AMS_JOBNAME=S1 $AMSBIN/ams <<eor
...
LoadSystem
  File S1_GEO.results/adf.rkf
End
Task SinglePoint
Properties
  NormalModes True
End
Engine ADF
...
Excitation
  Onlysing
  Lowest 1
End
ExcitedGO
  State A 1
  Singlet
End
EndEngine
eor
```

Next for the absorption spectrum, we look at excitations from the lowest vibrational state of the electronic ground state to the vibrational levels of the first singlet excited state S_1 ($S_1 \leftarrow S_0$), using the *FCF program* (page 189), which calculates the Franck-Condon factors between the vibrational modes of the two electronic states, with input

```
$AMSBIN/fcf << eor
STATES S0.results/adf.rkf S1.results/adf.rkf
QUANTA 0 5
SPECTRUM 0 10000 1001
TRANSLATE
ROTATE
eor
```

The number of vibrational quanta for the excited state should be larger in case of small molecules. See *the description of FCF program* (page 189) for more details.

For the fluorescence spectrum, we look at excitations from the lowest vibrational state of the first singlet excited state S_1 to the vibrational levels of the singlet ground state state S_0 ($S_1 \rightarrow S_0$). Input for the *FCF program* (page 189) is in this case:

```
$AMSBIN/fcf << eor
STATES S0.results/adf.rkf S1.results/adf.rkf
QUANTA 5 0
SPECTRUM -10000 0 1001
TRANSLATE
ROTATE
eor
```

The number of vibrational quanta for the ground state should be larger in case of small molecules.

Note that the FCF program calculates the spectrum relative to the 0-0 transition. Thus one should add to spectrum calculated with FCF the difference in energy of the lowest vibrational state of the ground state S_0 and the lowest vibrational state of the electronically singlet excited state S_1 .

7.1.3 FCF Example phosphorescence

In this example it is assumed that the molecule has a singlet ground state S_0 , and the interesting excited state is the lowest triplet excited state T_1 . Emission from a triplet state to a singlet state is spin forbidden, however, due to spin-orbit coupling such transitions may occur. In the following we assume that the geometry of the triplet excited state is not influenced much by spin-orbit coupling.

First one needs to do a the ground state geometry optimization, followed by a frequency calculation, using `AMS_JOBNAME=S0`. Next one needs to do an excited state geometry optimization of the lowest triplet excited state, followed by a frequency calculation.

```
AMS_JOBNAME=T1 $AMSBIN/ams <<eor
...
Task GeometryOptimization
Properties
  NormalModes True
End
Engine ADF
  Unrestricted
  SpinPolarization 2.0
EndEngine
eor
```

For the phosphorescence spectrum, we look at excitations from the lowest vibrational state of the first triplet excited state T_1 to the vibrational levels of the singlet ground state S_0 ($T_1 \rightarrow S_0$). Input for the *FCF program* (page 189) is in this case:

```
$AMSBIN/fcf << eor
STATES S0.results/adf.rkf T1.results/adf.rkf
QUANTA 5 0
SPECTRUM -10000 0 1001
TRANSLATE
ROTATE
eor
```

The number of vibrational quanta for the ground state should be larger in case of small molecules.

Note that the FCF program calculates the spectrum relative to the 0-0 transition. Thus one should add to spectrum calculated with FCF the difference in energy of the lowest vibrational state of the ground state S_0 and the lowest vibrational state of the electronically triplet excited state T_1 .

Zero field splitting (ZFS) and the radiative rate constants (i.e. radiative phosphorescence lifetimes) could be calculated with spin-orbit coupled ZORA time-dependent density functional theory (ZORA-TDDFT). With the ADF engine spin-orbit coupling can be treated self-consistently (i.e. non perturbatively) during both the SCF and TDDFT parts of the computation.

An alternative to the use of the unrestricted formalism to calculate the lowest triplet excited state is to use the TDDFT formalism:

```
AMS_JOBNAME=T1_GEO $AMSBIN/ams <<eor
...
Task GeometryOptimization
Engine ADF
  Excitation
    Onlytrip
    Lowest 1
  End
  ExcitedGO
```

(continues on next page)

(continued from previous page)

```

        State A 1
        Triplet
    End
EndEngine
eor

```

To get the frequencies for this excited state, numerical frequencies need to be calculated, at the optimized geometry of the first excited state.

```

AMS_JOBNAME=T1 $AMSBIN/ams <<eor
...
LoadSystem
    File T1_GEO.results/adf.rkf
End
Task SinglePoint
Properties
    NormalModes True
End
Engine ADF
...
Excitation
    Onlytrip
    Lowest 1
End
ExcitedGO
    State A 1
    Triplet
End
EndEngine
eor

```

7.2 VG-FC: Vertical Gradient Franck-Condon

Electronic spectra, such as absorption, emission, phosphorescence, and ionization, may contain a vibrational structure.

In the vertical gradient Franck-Condon (VG-FC) method some extra assumptions are made compared to the Adiabatic Hessian Franck-Condon (AH-FC) method. VG-FC is also called the Independent Mode Displaced Harmonic Oscillator (IMDHO) model. The AH-FC model (used in the *FCF Module* (page 189)) makes the following assumptions:

- 1. It employs the adiabatic (Born-Oppenheimer) approximation and treats the nuclei as moving in an effective potential defined by the electronic configuration.
- 2. It works at the Franck-Condon point and assumes that the transition occurs at the ground state equilibrium structure for absorption and at the excited state equilibrium structure for emission.
- 3. It applies the harmonic approximation to both the ground and excited state potential energy surfaces.

The VG-FC model makes following additional assumptions:

- 4. It assumes the excited state PES has the same shape as that of the ground state, but it is displaced from it, i.e. both states have the same normal modes and frequencies but different equilibrium geometries.
- 5. The excited state equilibrium structure is found from a Newton-Raphson step from the ground state geometry using the excited state gradient at this point.

Under these simplifying approximations we can reduce the ingredients necessary for a spectrum calculation to an excited state gradient and a set of ground state normal modes. Furthermore it uses a time-domain description of the

absorption cross-section (this introduces no further approximations) so we do not need any explicit Franck-Condon factors as is done in the *FCF Module* (page 189). This approach is particularly effective for large molecules as it shows linear scaling with the number of normal modes. Further details on how to use this method, and the VG-FC model in general can be found below.

The approximations we make remove the need for an excited state geometry optimization and frequency analysis. This means the most expensive step in a typical calculation is actually the ground state frequency analysis. The ability of Mode Tracking to generate a set of approximate normal modes, without performing a full frequency analysis can be applied to the calculation of vibrationally resolved optical spectra. This is exactly what is done in *Vibronic-Structure Tracking* (page 198). On the other hand, the ability of Mode Refinement to refine entire spectral regions at once means this method can be transferred to calculating vibrationally resolved optical spectra from a select number of (vibronically active) approximate normal modes. This idea is applied in *Vibronic-Structure Refinement* (page 200). Under suitable assumptions, the computational cost of calculating an approximation of the vibronic-structure can be reduced significantly through the use of these two methods, allowing for application to large molecules inaccessible to more detailed methods. While they are clearly best suited for these large molecules, they can also serve as an efficient method to obtain quick, first approximations for more moderately sized molecule.

With the VG-FC model one can calculate the 0-0 energy difference between the ground state and excited state. In this VG-FC method the vibrationally resolved emission spectrum will be a mirror image of the vibrationally resolved absorption spectrum, in which the 0-0 transitions are the same.

Note: The VG-FC model uses certain approximations which may not always be valid. Something to look out for that may affect the reliability of VG-FC results are large changes in conformation upon excitation. In this case the normal modes of the excited state may be completely different from those of the ground state, at which point the model is not expected to produce reliable results. Should non-adiabatic effects become important, the BO approximation could potentially break down as well. In these cases the *FCF Module* (page 189) may provide a more suitable alternative to the VG-FC model.

See also:

Tutorials: [Resonance Raman](#), [Vibrationally resolved electronic spectra with ADF](#), [Vibrationally resolved electronic spectra with DFTB](#)

7.2.1 Theory

Here we will introduce the theory behind the model used to compute vibronic couplings. We use the so-called Vertical Gradient Franck-Condon (VG-FC) method, also called Independent Mode Displaced Harmonic Oscillator model (IMDHO). The VG-FC model as used here works as follows: The PES in the ground state is assumed parabolic (the harmonic approximation). Upon excitation, we assume that the only change to the PES that occurs is a shift in origin (i.e. no stretching or rotation of the parabolic well). This means, both the normal modes and their frequencies remain the same in both the ground and excited state. The result is that a spectrum can be calculated using only a single excited state gradient at the optimized ground state geometry. Hence we avoid the expensive excited state geometry optimization and frequency analysis used by methods such as the *FCF Module* (page 189).

Moving to a slightly more detailed discussion, we can start with an expression for the absorption cross-section in terms of a sum over contributions from different Franck-Condon factors, each homogeneously broadened with an assumed Lorentzian of linewidth Γ . For excitation from the vibrational ground state of electronic state m , to electronic state n we get the following expression (Hartree atomic units are used throughout):

$$\sigma_{n \leftarrow m}(\omega) = \frac{4\pi\omega}{3c} |\mu_{mn}|^2 \sum_{l_{n,1}} \dots \sum_{l_{n,N_{modes}}} \frac{\Gamma \prod_{i=1}^{N_{modes}} |\langle l_{n,i} | 0_{m,i} \rangle|^2}{\left(E_{n,0} + \sum_i^{N_{modes}} \omega_i l_{n,i} - E_{m,0} - \omega \right)^2 + \Gamma^2}$$

Here the μ_{mn} is the transition dipole moment and the $\prod_{i=1}^{N_{modes}} |\langle l_{n,i} | 0_{m,i} \rangle|^2$ denote the Franck-Condon factors for excitation from the vibrational ground state to the vibrationally excited state of the excited state PES, with quantum

numbers $l_{n,i}$. This sum-over-states formulation is what is also used to compute the spectrum in the *FCF Module* (page 189). While efficient pre-screening and thresholding techniques can make this method applicable to moderately sized molecules, it is still quite expensive for large molecules. We can derive an expression which is equivalent to the one above, but which enjoys significant computational benefits. This is done by writing the cross-section as an integral over time:

$$\sigma_{n \leftarrow m}(\omega) = \frac{4\pi\omega}{3c} |\mu_{mn}|^2 \text{Re} \int_0^\infty \langle i|i_n(t) \rangle e^{i[\omega + E_{m,0}]t} \cdot e^{-\Gamma t} dt$$

Here, $\langle i|i_n(t) \rangle$ denotes the overlap of the initial state, propagated along the excited state PES with itself at time $t = 0$. This expression is still completely general, however, using the *assumptions of the VG-FC model* (page 188), this overlap admits a simple closed form expression:

$$\langle i|i_n(t) \rangle = \prod_{j=1}^{N_{modes}} \exp \left[-\frac{\Delta_{n,j}^2}{2} (1 - e^{-i\omega_j t}) \right] e^{-iE_{n,0}t}$$

This expression can be derived by different methods and admits a few interpretations, the details of which we will not go into here. The interested reader is referred to reference¹. Instead we will focus on its numerical implementation. The great computational advantage of this expression is that it scales only weakly with the number of degrees of freedom. Compare this with the sum-over-states formalism above, where the number of FCF's needed to have a converged sum proliferates for large molecules and the advantage becomes clear.

The $\Delta_{n,j}$ appearing in the expression are the dimensionless normal mode displacements at the equilibrium geometry in state n, compared to state m along normal mode j which has angular frequency ω_j . They represent the change in origin of the PES along a particular normal mode. These are the parameters which control the shape of the spectrum. Under the assumptions we listed above they can be calculated from the excited state energy gradient at the ground state equilibrium structure as:

$$\Delta_j = \frac{q_j^m \cdot \nabla_m E}{\omega_j^{3/2}}$$

Where we have suppressed the n subscript. The gradient is with respect to mass weighted coordinates and the q_j^m denote the mass-weighted normal modes.

Essentially, we use the (assumed) parabolic shape of the PES in the harmonic approximation along each of the normal modes to determine the displacement from the gradient projection along that mode. An important consequence of the simplified VG-FC model is that the Franck-Condon factors can be expressed in closed form:

$$|\langle l_i|0_i \rangle|^2 = \frac{\left(\frac{\Delta_i}{2}\right)^2}{l_i!} e^{-\frac{\Delta_i^2}{2}}$$

Modes with the largest displacement are then expected to also have the largest impact on the structure of the vibrational progression in the spectrum. This gives us an obvious candidate for a tracking method to be used with the Mode Tracking protocol. This approach is taken in the *Vibronic-Structure Tracking* (page 198) module.

Theory: Vibronic-Structure Tracking

Vibronic-structure tracking (VST) is a method for obtaining quick and approximate vibrationally resolved optical spectra for larger sized molecules. It is based on the mode-tracking algorithm and works by tracking those modes that are expected to have the largest impact on the vibronic-structure of the spectrum. Its AMS implementation follows that in reference¹.

¹ J.R. Reimers, K.R. Wilson and E.J. Heller, *Complex time dependent wave packet technique for thermal equilibrium systems: Electronic spectra* The Journal of Chemical Physics 79, 4749 (1983) (<https://doi.org/10.1063/1.445618>)

The modes that are tracked by the mode tracking routine are used to generate a UV/vis spectrum. This is done using the Vertical Gradient Franck-Condon (VG-FC) method, also called the Independent Mode Displaced Harmonic Oscillator (IMDHO) model as described on the *Vibronic-Structure* (page 196) page. This model reduces the necessary TD-DFT calculations to a single excited state gradient performed at the ground state equilibrium geometry. At this point it is typically the ground state frequency analysis which becomes the bottleneck for large molecules. Through the use of Mode Tracking we can circumvent this step and calculate only those modes which will significantly impact the vibronic fine structure. This yields a very cheap and efficient method for obtaining vibrationally resolved optical spectra.

The use of Mode Tracking within the context of vibronic-structure calculations proceeds as follows: from an initial guess for a normal mode, we construct a new basis vector which we hope will improve the quality of our initial guess. We then iteratively expand the basis by refining those modes which have the largest impact on the vibronic fine structure. We keep iterating until we deem the spectrum to be converged. We'll address the technical details of this procedure in this section.

Let us first consider how we can gauge which modes will have a large impact on our spectrum. Over on the *Vibronic-Structure* (page 196) page, we showed that according to the VG-FC model, the Franck-Condon factors can be written as:

$$|\langle l_i | 0_i \rangle|^2 = \frac{\left(\frac{\Delta_i}{2}\right)^2}{l_i!} e^{-\frac{\Delta_i^2}{2}}$$

This expression implies modes with large displacements Δ_j will have the largest Franck-Condon factors. Thus, we expect that modes with large oscillator displacements will contribute most strongly to the vibronic fine structure of the spectrum. This suggests a tracking method in addition to those described in the *Mode Tracking* (page 163) documentation. Namely, to track the mode with the largest Δ_j . This is the only setting for the `TrackingMethod` supported by *Vibronic-Structure Tracking* and is automatically set when choosing *Vibronic-Structure Tracking* as the *Vibrational Analysis* type. As a reminder, the oscillator displacements can be obtained from the assumed parabolic shape of the excited state PES and the known excited state gradient:

$$\Delta_j = \frac{q_j^m \cdot \nabla_m E}{\omega_j^{3/2}}$$

This expression for our normal mode displacements suggests that a reasonable guess for a hypothetical normal mode with a large effect on the spectrum is given by the normalized excited state gradient, as this maximizes the projection.

The choice of preconditioner is of course an important one. While better preconditioners such as the Jacobi-Davidson method can lead to fast and tightly directed convergence of the modes, this may not be ideal in the case of *Vibronic-Structure Tracking*, as we are not necessarily focused on obtaining converged modes but rather obtaining a converged spectrum. By default the method does not use any preconditioner for generating normal modes from the residual vectors to allow the procedure to more freely explore the entire space of modes. This means that the new basis vector produced on each iteration is the (normalized) residual vector of that iterations selected mode. Note that this is different from using the Davidson method with an identity matrix as guess for the Hessian. The user is still able to use any of the pre-conditioning options provided by the stand-alone mode tracking method, however we suggest the user uses the default `I` (for identity) as the `UpdateMethod`. This also conveniently means no approximate Hessian is required.

It is important to choose a suitable convergence criterion. When using *Vibronic-Structure Tracking*, the aim is to get an approximation of the progression in the spectrum. The convergence of the modes then takes a backseat to the convergence of the spectrum. Although the two usually go hand in hand, requiring that all modes be converged in the usual mode-tracking sense is likely to be far too restrictive. Thus the convergence criterion used for VST is the following: at every iteration, the previous iteration's spectrum is subtracted from the new one. The absolute difference is then integrated on the requested frequency range. Convergence is achieved once the so-obtained number drops below a pre-set convergence threshold. The result is that it may not be the case that all modes are exact normal modes yet, however further refinement will have a limited effect on the resulting spectrum, for example due to these modes having very small Franck-Condon factors.

Convergence according to this criterion is affected by the other options that the user chooses, see the section on how the adiabatic excitation energy is calculated.

Theory: Vibronic-Structure Refinement

Vibronic-Structure Refinement takes the idea of Mode Refinement, and applies it to the calculation of vibrationally resolved optical spectra. Under the assumptions of the Vibronic-Structure application, the most time-consuming step becomes the calculation of the ground state normal modes. However, we may not be interested in all normal modes that contribute to the Vibronic fine structure of a spectrum. By using Mode Refinement we can limit ourselves to the spectral region of interest, or to modes which we expect to have the largest impact on the fine structure. The details of the Mode Refinement protocol can be found on the relevant *documentation page* (page 160).

Theory: Adiabatic excitation energy

The last aspect we will comment on here is the calculation of the adiabatic excitation energy. The number we get from an excited state calculation is the vertical excitation energy (ΔE_{vert}), this number is generally unobservable in spectra. The more interesting quantity is the adiabatic excitation energy (ΔE_{ad}). The former is the difference in energy between the two PES'es at the ground state geometry, the latter is the difference in energy between the ground and excited state vibrational ground states. Since we assumed both PES'es to have the same frequencies, the zero-point energy of the two states are the same and hence, the adiabatic energy difference is the same as the 0-0 energy difference (ΔE_{0-0}). This is simply the difference between the bottom of the two PES'es. It can be reconstructed from the excited state gradients projected onto our normal modes, as well as their frequencies. For a molecule with N modes we obtain:

$$\Delta E_{ad} = \Delta E_{vert} - \sum_j^{N_{modes}} \frac{1}{2} \omega_j \Delta_j^2$$

Where the sum represents the reorganization energy stored in the excited oscillator modes. Depending on the desired information, one can use two different representations of the spectrum. The first is to simply plot the spectrum against an energy range with the ground state energy as an offset. This would represent a true absorption spectrum that can be compared directly with experiment. However, the energy difference $\Delta E_{vert} - \Delta E_{ad}$ in the VG-FC model is reconstructed only from the ground state equilibrium structure and can thus be somewhat unreliable. If the user is primarily interested in the shape of the spectrum, using the 0-0 excitation energy as an offset may be a more suitable choice. The latter is the default setting in AMS, but the method supports both options.

Note that in case of a Vibronic-structure tracking (VST) calculation we only have access to a small set of approximate normal modes, we will have an approximation of ΔE_{ad} that changes from iteration to iteration as new normal modes are introduced and the old normal modes are refined. If a spectrum relative of the 0-0 energy is requested, the shift caused by the iterative convergence of ΔE_{ad} is not taken into account, although it will still have some effect on the spectrum as it also appears in the integral that defines the spectrum. If one is interested in a converged approximation of ΔE_{0-0} it may be more reliable to provide an absolute frequency range for the spectrum as the shift in the peak locations will then also affect convergence.

7.2.2 Input: Vibronic-Structure all modes

As mentioned, we need both a set of ground state normal modes as well as an excited state gradient. All the normal modes are calculated using the AMS engine that was selected, or, in case the user has a pre-calculated set of normal modes, these can be read from a .rkf file using the `ModeFile` key in the `NormalModes` sub-block. In this latter case, the engine is not used. One can then use the keys in the `ModeSelect` block to filter out specific modes, or, simply select all modes using the `Full` key in this block. For further details on how to use the Mode Select block for more specialized selection options, see the *Mode Select* (page 171) documentation. As for the excited state information, this is passed to the application via the `ExcitationSettings` block. Additionally there are (optional) settings related to the appearance of the spectrum under the `AbsorptionSpectrum` block. For an overview of the input options see the list of keys at the end of this page.

For completeness we provide an example of what the user input may look like:


```

Task VibrationalAnalysis
VibrationalAnalysis
  Type VibronicStructure
  NormalModes
    # Select all modes present in the .rkf file
    ModeSelect
      Full True
    End
  End
  ExcitationSettings
    ExcitationInfo File
    ExcitationFile ./your_excitation.t21
    Singlet
      Bl.u 2
    End
  End
  AbsorptionSpectrum
    LineWidth 100.0
    AbsorptionRange -500.0 4000.0
  End
End

```

7.2.3 Input: Vibronic-Structure Tracking

VST is notably the only Vibrational Analysis task that does not support the `NormalModes` block as no initial normal modes are necessary. The excited state gradient fills this role instead. Furthermore, the recommended default of using the identity matrix as a preconditioner means that no approximate Hessian is required either.

One thing to note regarding the Mode Tracking settings is that VST features a convergence criterion not present in standard Mode Tracking. The tolerance for the convergence of the spectrum is set using the `ToleranceForSpectrum` keyword in the `ModeTracking` block. The default is set to 0.01 and should be sufficient for most purposes, but *restarting* (page 202) with a lower value may improve the reliability of the convergence.

Note: The usual Mode Tracking tolerances are still present in addition to the spectral tolerance mentioned above. This is so we do not track modes that have already converged. Instead, once a mode is converged in the usual mode tracking sense, we switch to the next mode that satisfies our tracking criterion. The defaults should normally apply, but may be loosened a bit to allow for more free subspace exploration.

A typical Vibronic-Structure Tracking run may be setup as follows:

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type VibronicStructureTracking
  # Select our excited state energy+gradient from a previous calculation
  ExcitationSettings
    ExcitationInfo File
    ExcitationFile ./your_excitation.t21
  End
  # Tuning how our spectrum will look
  AbsorptionSpectrum
    LineWidth 250.0
    AbsorptionRange -500.0 6000.0
  End
  ModeTracking

```

(continues on next page)

```
UpdateMethod I
End
End
```

Input: Restarting VST

The Vibronic-Structure Tracking task is fully restartable from a previously completed VST run. The main reason why it may be useful to perform a restart is related to the way the spectrum is generated. Most modules/applications in AMS that calculate a spectrum usually produce a stick spectrum. This stick spectrum is then convoluted against either a gaussian or lorentzian of specified width or area. This allows one to tweak the homogeneous broadening post-calculation to improve agreement with experiments. VST operates differently, in that the Lorentzian linewidth is specified at the start of the calculation as an essential input parameter. If upon convergence it becomes clear that a different linewidth would have been better suited for the spectrum at hand, one can restart the calculation with this new linewidth. The program then computes a new spectrum using these new settings and repeats the last iteration of the previous run. The latter is done because the linewidth directly influences the convergence (larger linewidths tend to converge more easily as the features are not as sharply resolved). If the spectrum does not change after the first iteration the program terminates as usual and the new spectrum is now available to the user. However, if this additional iteration does not pass the convergence test, the program keeps iterating until the spectrum converges for this new linewidth.

In addition to changing the linewidth post-calculation, the user is also free to change the spectral range that is to be computed as well as its resolution through the `FrequencyGridPoints` keywords. Alternatively, the tolerance used may be lowered if tighter convergence is required. Importantly however, none of the previous run's input settings are retrieved upon restart, so any settings the user doesn't want to change have to be same in the new run script as in the old. The easiest way to restart is to simply re-use the previous run's runscript under a new name and then changing the relevant settings.

The restart functionality is controlled by the `RestartPath` keyword. One simply states the path to a previous calculation's `.rkf` file from which the relevant data is to be read. The `.rkf` file of any VST runs contains a section labeled `VSTRestart` which contains the data needed to restart from that run. Restarting is then a simple one-liner in the `VibrationalAnalysis` block:

```
VSTRestartFile ./previous_run.results/ams.rkf
```

7.2.4 Input: Vibronic-Structure Refinement

Vibronic-Structure Refinement (VSR) assumes that the modes relevant to the vibronic-structure are, at least in an approximate sense, known a priori. The VSR method takes these approximate modes and refines them via the Mode Refinement algorithm. The approximate normal modes should thereby span a subspace containing the exact normal modes and, the more similar they are to the exact mode, the better they allow to reproduce the spectrum of a full ground state frequency analysis. Naturally, the refinement becomes more accurate with increasing dimensionality of the spanned subspace, although this increases the computational costs. In any case, the subspace needs to contain those modes that are most relevant for the vibronic progression for the entire approach to work, which leads to the question, how do we select such modes?

This is done using familiar `ModeSelect` block from the mode selective analysis tasks tasks. While all the usual options are available, for which further details can be found on the *Vibrational Analysis* (page 171) page, there are two selection methods that are geared specifically towards VSR. Within the VG-FC model, the parameters defining the effect a mode will have on the spectrum are the oscillator displacements Δ_j . We can determine these values from the projection of the approximate normal modes onto the gradient in combination with their respective frequencies. We are then interested in those modes with the largest displacement. To select these, two options are available:

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type VibronicStructureRefinement
  NormalModes
    ModeSelect
      LargestDisplacement integer
      DisplacementBound float
      ...
    End
  ...
End
...
End

```

LargestDisplacement integer sets an integer value N, to select the N modes with the largest displacements as calculated within the VG-FC model. This method limits the total number of modes we refine and thus places an upper bound on the computational cost required by the method, but it does not ensure that all relevant modes have been selected. In particular there may still be modes with large displacements that have been omitted and as such peaks or progressions may be missing from the spectrum.

DisplacementBound float Alternatively this selects all approximate modes with a displacement greater than the supplied lower bound. An appropriate choice for this value ensures all modes which visibly affect the spectrum (at least at the lower level of theory used to produce the approximate modes) are included in the subspace basis used for Mode Refinement. The displacements Δ_j are dimensionless and the largest ones will typically be of order one. Based on this, a value of 0.01 for this parameter will generally select all relevant modes. This of course comes at the trade-off that for general molecules it is not known how many such modes there will be and the computational cost may be larger than expected. But, if one is not discouraged by this, this will of course yield the most accurate results. Values up to 0.05 may still provide reasonable results while minimizing computation times. Using larger linewidths (AbsorptionSpectrum%LineWidth) for the homogeneous broadening can help to correct for the non-resolved modes.

The calculation setup for Vibronic-Structure Refinement is essentially the same as that for Vibronic-Structure, the only difference being that the modes that we provide will first be refined, before a spectrum is computed. A detailed description is given on the *Vibronic-Structure* (page 196) page. A typical VSR calculation can be setup as follows:

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type VibronicStructureRefinement
  # Select our normal modes from a previous calculation
  NormalModes
    ModeFile ams.rkf
    ModeSelect
      # select all modes with VG-FC displacements over 0.01
      DisplacementBound 0.01
    End
  End
  # Select our excited state energy+gradient from a previous calculation
  ExcitationSettings
    ExcitationInfo File
    ExcitationFile ./your_excitation.t21
    # Select second singlet excitation with symmetry label A
    Singlet
      A 2
    End
  End
  # Tuning how our spectrum will look
  AbsorptionSpectrum

```

(continues on next page)

(continued from previous page)

```

    LineWidth 250.0
    AbsorptionRange -500.0 6000.0
  End
End

```

7.2.5 Input: Excited State

Both the excited state gradient at the ground state equilibrium structure as well as the vertical excitation energy are necessary to obtain a spectrum. These can be provided by most TD-DFT programs, including ADF. Unfortunately, the AMS driver does not yet support computing these properties on the fly. As a result they are currently required as user input. One can use ADF or DFTB to compute them and then either read them from the produced output file, or copy and paste them to an inline input block.

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type ...
  ExcitationSettings
    ExcitationInputFormat [File | Inline]
    ExcitationFile string
    EnergyInline float
    GradientInline # Non-standard block. See details.
    ...
  End
  Singlet # Non-standard block. See details.
  ...
  End
  Triplet # Non-standard block. See details.
  ...
  End
End
End

```

ExcitationInputFormat [File | Inline] this keyword is used to specify whether the gradient and energy are to be read from a kf file or if one chooses to use the inline block for them. Currently, no TD-DFT engine is implemented in the AMS driver. Furthermore, ADF uses the old tape21 format for its output which is quite different from the more streamlined AMS format. The `inline` option allows one to bypass any possible confusion here. The `File` option requires the specification of the specific excitation to read, which we will discuss below.

GradientInline is the block where the gradient can be specified if the inline option is selected. The format can be either a N by 3 block (N rows, 3 columns) or a 3N long column vector if one happens to have a gradient in this format. The gradient values should be in the unit Hartree/Bohr.

EnergyInline float is used to specify the excitation energy. This will just be a single floating point number (in Hartree).

The resulting input would look like:

```

Task VibrationalAnalysis
VibrationalAnalysis
  Type ...
  ...
  ExcitationSettings
    ExcitationInfo Inline
    # Excited state gradient for transhexatriene 14 atoms x 3 coordinates (to be
    provided in Hartree/Bohr)

```

(continues on next page)

(continued from previous page)

```

GradientInline
  -0.03786125      0.01786798      0.00003833
    0.05322148     -0.00798712     -0.00004152
  -0.06658803      0.01373495      0.00004727
    0.06656379     -0.01374825     0.00002398
  -0.05318451      0.00799875     -0.00002097
    0.03783718     -0.01786722     0.00001362
    0.00382226      0.00327391      0.00001046
  -0.00046176     -0.00499971     -0.00003583
    0.00014312      0.00534412     -0.00001335
    0.00011081     -0.00558254     -0.00002107
  -0.00011074      0.00558350     -0.00000933
  -0.00014517     -0.00534330      0.00000839
  -0.00381513     -0.00327314     -0.00000248
    0.00046796      0.00499807      0.00000250

End
# The vertical excitation energy is simply a float (to be provided in Hartree)
EnergyInline 0.17062882
End
End

```

ExcitationFile string If `ExcitationInputFormat` is set to `File` one also has to set the `ExcitationFile` keyword to provide a path to the file from which the info should be read.

Singlet or Triplet In case the excited state information should be read directly from a file, one should specify which excitation AMS should read. This is done analogously to how this is done for excited state gradients in ADF. One chooses either the `Singlet` or `Triplet` blocks (for singlet-singlet or singlet-triplet excitations respectively). Once such a block is chosen, a line containing the symmetry label, followed by the number of the excitation of that symmetry. As for the symmetry labels, the notational convention used can be found in the [ADF manual appendix](#) on symmetry labels. If symmetry was disabled, either explicitly by the user or when using TD-DFTB which does not support symmetry, when calculating the excited state properties, the symmetry label `A` should be used. Vibronic-structure calculations are only supported for one excitation at a time.

The resulting input block looks something like this:

```

ExcitationSettings
  ExcitationInputFormat File
  ExcitationFile ./ExcitedState.results/myengine.rkf
  Singlet
    A 1
  End
End

```

7.2.6 Input: Producing the spectrum

To go from normal modes to an actual spectrum, we have to solve the integral that we introduced in the first section, the details of which will be discussed here.

For long times, the integrand is dominated by the exponential damping term. We can define a suitable cut-off by demanding that this damping term be smaller than some threshold. As mentioned, the integral is evaluated using a Chebyshev quadrature. The number of Chebyshev nodes used for the integral can then be determined from this integration limit and the time step size. This time step size should be large enough to accurately represent the highest frequencies that are present in the highly oscillatory integrand. The Nyquist frequency can be used as a guide here: the highest frequency should be represented by at least two samples each period. Note however, that in principle there is no upper bound to the frequencies present in the integrand due to the fact that modes can be excited to arbitrarily high

energy levels. Fortunately, the Franck-Condon factors will quickly act to dampen these high frequencies, restricting their significance. The user may either supply a time step size directly or leave this up to the program to determine. If the latter is chosen the highest relevant frequency is estimated by assuming that oscillator excitations are relevant up to about twice the Huang-Rhys parameter (which follow directly from the displacements). This default should in most cases be sufficient to produce a converged integral.

The keywords associated with the different input parameters are collected under the `AbsorptionSpectrum` block. It contains the following keys:

```
Task VibrationalAnalysis
VibrationalAnalysis
  Type ...
  ...
  AbsorptionSpectrum
    AbsorptionRange float_list
    FrequencyGridPoints integer
    LineWidth float
    SpectrumOffset [absolute | relative]
  End
End
```

LineWidth float sets the lorentzian line width and thus has a large impact on the appearance of the final spectrum. It may be used to improve the agreement of the computed spectrum with experiment. Another purpose that the linewidth may serve is that some low frequency modes may not be represented very accurately within the VG-FC model. However, these modes do not result in particularly distinct vibronic progressions but rather cause an unspecific broadening of existing peaks. Applying a homogeneous peak broadening by increasing the value of Γ may be an effective (albeit a little ad hoc) way of resolving such issues, should they occur. The default value is 200cm^{-1} but it is recommended to do some experimentation with this value.

AbsorptionRange float_list keyword specifies the frequency range (in cm^{-1}) that is to be computed by VST.

SpectrumOffset [absolute | relative] specifies whether the range provided by the `AbsorptionRange` keyword is relative to the 0-0 excitation energy (`Relative`), or relative to the vertical excitation energy (`Absolute`). Due to the limited accuracy of the VG-FC model at predicting ΔE_{0-0} the default is `Relative` with a `AbsorptionRange` of `[-500,4000]`.

FrequencyGridPoints integer sets the number of points we use on our frequency grid. It is set to 400 by default which generally produces a smooth looking spectrum.

The current implementation of VG-FC vibronic-structure supports only spectra for one excitation at a time. The spectrum is normalized such that the highest peak is equal to 1 in arbitrary units. For this reason the prefactor of the integral is irrelevant and transition dipole moments do not affect the appearance of the spectrum.

DIPOLE MOMENT, POLARIZABILITY, BOND ORDERS

This page in the AMS manual describes the calculation of the dipole moment, the polarizability, and bond orders.

```
Properties
  DipoleMoment Yes/No
  Polarizability Yes/No
  BondOrders Yes/No
End
```

Note that because these properties are tied to a particular point on the potential energy surface, they are found on the *engine output files* (page 17). Note also that the properties are not always calculated in every PES point that the AMS driver visits during a calculation. By default they are only calculated in *special* PES points, where the definition of special depends on the *task* (page 41) AMS is performing: For a *geometry optimization* (page 43) properties would for example only be calculated at the final, converged geometry. This behavior can often be modified by keywords special to the particular running task.

8.1 Charges, Dipole Moment, Polarizability

```
Properties
  Charges Yes/No
  DipoleMoment Yes/No
  DipoleGradients Yes/No
  Polarizability Yes/No
End
```

Properties

Charges

Type Bool

Default value No

Description Requests the engine to calculate the atomic charges.

DipoleMoment

Type Bool

Default value No

Description Requests the engine to calculate the electric dipole moment of the molecule. This can only be requested for non-periodic systems.

DipoleGradients

Type Bool

Default value No

Description Requests the engine to calculate the nuclear gradients of the electric dipole moment of the molecule. This can only be requested for non-periodic systems.

Polarizability

Type Bool

Default value No

Description Requests the engine to calculate the polarizability tensor of the system.

8.2 Bond orders & Molecule detection

Many engines can determine bond orders between atoms. For engines based on force fields, these might just be the bond orders used internally by the force field, while for quantum mechanical engines the bond orders are usually determined by analyzing the results of the quantum mechanical calculation, e.g. the electronic density. We refer users to the manuals of the respective engine for details.

```
Properties
  BondOrders Yes/No
End
```

Properties

BondOrders

Type Bool

Default value No

Description Requests the engine to calculate bond orders. For MM engines these might just be the defined bond orders that go into the force-field, while for QM engines, this might trigger a bond order analysis based on the electronic structure.

Based on the bond orders, the AMS driver can analyze the atomic connectivity graph in order to determine which sets of atoms together constitute molecules. This allows for example to monitor the changes in molecular composition during a reactive molecular dynamics calculation. For *molecular dynamics* (page 77) calculations this option is enabled by default. For other *tasks* (page 41) the molecular analysis can explicitly be requested in the `Properties` block.

```
Properties
  Molecules Yes/No
End
```

Properties

Molecules

Type Bool

Default value No

Description Requests an analysis of the molecular components of a system, based on the bond orders calculated by the engine.

Details of the molecule detection are configured in a dedicated block:


```
Molecules
  AdsorptionSupportRegion string
  BondOrderCutoff float
End
```

Molecules

Type Block

Description Configures details of the molecular composition analysis enabled by the Properties%Molecules block.

AdsorptionSupportRegion

Type String

GUI name Adsorption support region

Description Select region that will represent a support for adsorption analysis. Adsorbed molecules will receive an '(ads)' suffix after name of the element bonded to the support. Such elements will be listed separate from atoms of the same element not bonded to the support, for example, HOH(ads) for a water molecule bonded to a surface via one of its H atoms.

BondOrderCutoff

Type Float

Default value 0.5

Description Bond order cutoff for analysis of the molecular composition. Bonds with bond order smaller than this value are neglected when determining the molecular composition.

ENGINES

The engines are the core of the Amsterdam Modeling Suite: While the AMS driver steers the calculation over the potential energy surface in e.g. a *geometry optimization* (page 43) or *molecular dynamics* (page 77) calculation, the engines calculate energies and gradients and in this way *define* the PES on which the driver works.

```
Engine header
```

Engine

Type Block

Description The input for the computational engine. The header of the block determines the type of the engine.

The engine used for an AMS calculation is selected and configured with the special `Engine` block in the AMS input:

```
Engine DFTB
... input for the DFTB engine ...
EndEngine
```

Here the type of engine, e.g. DFTB as in the example above, is specified on the line that opens the block. Note that the `Engine` block ends with `EndEngine`, and is in this way different from all the other blocks in the AMS input, which close just with `End`. The content of the engine block is what we call the “engine input”. Generally the engine input consists of a series of blocks and keywords, and looks just like the AMS driver input. However, many engines have a lot of options and keywords, which are documented in a separate engine manual. In other words: This AMS driver manual documents all the keywords outside of the `Engine` block, while the individual engine manuals document the contents of the engine block.

9.1 Available engines

The following engines are available in the 2021.1 release of the Amsterdam Modeling Suite:

- **ADF** A DFT engine particularly strong in understanding and predicting structure, reactivity, and spectra of molecules.
- **BAND** An atomic-orbital based DFT engine aimed at periodic systems (crystals, slabs, chains) but supporting also molecular systems.
- **DFTB** An engine implementing Density Functional based Tight-Binding, a fast approximation to DFT.
- **ReaxFF** An engine for modeling chemical reactions with atomistic potentials based on the reactive force field approach.
- **MLPotential** Machine learning potentials (machine learning force fields).

- **ForceField** An engine implementing classical non-reactive force fields such as UFF (a non-reactive force field covering the entire periodic table).
- **Hybrid** Hybrid engine, for embedding and QM/MM calculations that combine multiple engines.
- **MOPAC** An engine wrapping the MOPAC code, a general-purpose semiempirical molecular orbital package for the study of solid state and molecular structures and reactions.
- **External (page 213)** A flexible scripting interface that allows advanced users to use external modeling programs as engines in AMS.
- **LennardJones (page 217)** A simple toy engine implementing a Lennard-Jones potential.

9.2 Summary of engine capabilities

Some options/properties can only be computed in combination with some of the engines (i.e., not all engines support all features). These tables summarize the capabilities of the engines of the Amsterdam Modeling Suite:

Table 9.1: Engine support for AMS driver options

Feature	ADF	BAND	DFTB	ReaxFF	MLPot.	ForceField	MOPAC
All elements available	✓	✓	✓ ¹			✓ ²	✓
<i>Non-periodic systems</i> (page 29)	✓	✓	✓	✓	✓ ³	✓	✓
<i>1D periodic systems</i> (page 29)		✓	✓			✓	✓
<i>2D periodic systems</i> (page 29)		✓	✓			✓	✓
<i>3D periodic systems</i> (page 29)		✓	✓	✓	✓ ³	✓	✓
Charged molecular systems	✓	✓	✓	✓		✓	✓
External electric field	✓	✓	✓				✓
External point charges	✓	✓	✓				

¹ All elements are available for the GFN1-xTB model, but other DFTB parameter sets are not parametrized for all elements.

² All elements are available with UFF (universal force field), but other force fields are not parametrized for all elements.

³ MLPotential backends may support non-periodic and/or 3D-periodic calculations.

Table 9.2: Engine support for AMS driver properties

Feature	ADF	BAND	DFTB	ReaxFF	MLPot.	Force-Field	MOPAC
<i>Atomic charges</i> (page 207)	✓	✓	✓	✓		✓	✓
<i>Bond orders</i> (page 208)	✓		✓	✓		✓	✓
<i>Dipole gradients</i> (page 207)	✓	✓	✓				✓
<i>Dipole moment</i> (page 207)	✓	✓	✓				✓
<i>Elastic tensor</i> (page 148)		✓	✓	✓	✓	✓	✓
<i>Gradients / Forces</i> (page 143)	✓	✓	✓	✓	✓	✓	✓
<i>Hessian</i> (page 144)	✓	✓	✓	✓	✓	✓	✓
<i>Infrared spectra</i> (page 153)	✓	✓	✓				✓
Molecule detection	✓		✓	✓		✓	✓
<i>Normal modes</i> (page 153)	✓	✓	✓	✓	✓	✓	✓
<i>PES point character</i> (page 144)	✓	✓	✓	✓	✓	✓	✓
<i>Phonons</i> (page 177)		✓	✓	✓	✓	✓	✓
<i>Polarizability</i> (page 207)	✓						
<i>Raman</i> (page 181)	✓						
<i>Stress tensor</i> (page 147)		✓	✓	✓	✓	✓	✓
<i>Thermodynamic properties</i> (page 173)	✓	✓	✓	✓	✓	✓	✓
<i>VCD</i> (page 187)	✓	✓	✓				✓
<i>VROA</i> (page 185)	✓						

Note: The features/options in the following tables are **engine-specific**, and are described in the corresponding engine's manual. The input for these options should be specified in the *Engine block* (page 7) section of the input file.

Table 9.3: Engine-specific capabilities

Feature	ADF	BAND	DFTB	ReaxFF	MLPot.	ForceField	MOPAC
EFG	✓ (doc)	✓ (doc)					
Electronic transport	✓ (doc)	✓ (doc)	✓ (doc)				
Energy decomposition analysis	✓ (doc)	✓ (doc)					
ESR/EPR	✓ (doc)						
Hyperpolarizabilities	✓ (doc)						
NMR	✓ (doc)						
Orbital energies	✓	✓	✓				✓
pKa							✓ (doc)
Solvation models	✓ (doc)	✓ (doc)	✓ (doc)				✓ (doc)
Spin-polarization	✓ (doc)	✓ (doc)	✓ (doc)				✓ (doc)
UV-VIS optical spectra	✓ (doc)		✓ (doc)				

Table 9.4: Engine-specific capabilities for periodic systems

Feature	ADF	BAND	DFTB	ReaxFF	MLPot.	ForceField	MOPAC
Density of states (DOS)		✓ (doc)	✓ (doc)				
Dielectric function		✓ (doc)					
Effective mass		✓ (doc)	✓ (doc)				
Electronic band structures		✓ (doc)	✓ (doc)				
k-space sampling		✓ (doc)	✓ (doc)				
X-ray form factors		✓ (doc)					

9.3 External programs as engines

The AMS driver allows running external programs as an engine. In this way users can combine the functionality in the AMS driver (tasks and PES point properties) with the energies and gradients of any molecular modeling program they have access to. For example, the graphical user interface supports setting up VASP as an external engine to the AMS driver.

In general, the interfacing between the AMS driver and the external program has to be done by the user in form of a small script, which allows users to hook up any external program without access to the source code of AMS. The graphical user interface of the Amsterdam Modeling Suite can then be used to analyze the results of these calculations.

An external engine is specified with `Engine External`. The keyword `Execute` is used to specify the command that is run to execute the external program:

```
Engine External
  Execute /path/to/my_interface_script.sh
EndEngine
```

The command can in principle be anything, as it will just be executed as is by the system shell. However, it should not use relative paths (e.g. to files in the directory where the input file is). We recommend writing the interfacing script in Python and using the Python interpreter that ships with AMS:

```
Engine External
  Execute "$AMSBIN"/amspython /path/to/my_python_interface_script.py
EndEngine
```

AMS then starts running and for every geometry prepares a folder in which the external engine is supposed to run. This is the folder in which the interface script specified with the `Execute` key is executed (so any relative paths are relative to that folder). AMS puts two files into this folder:

```
system.xyz
request.json
```

The `system.xyz` just contains the geometry AMS wants the external engine to calculate. It is an *extended format XYZ file* (page 313) with the `VEC1`, `VEC2`, `VEC3` extension at the end for periodic systems, e.g. diamond would look like this:

```
2
C      -0.51292147   -0.51292147   -0.51292147
C      0.51292147    0.51292147    0.51292147
VEC1   0.00000000    2.05168587    2.05168587
VEC2   2.05168587   -0.00000000    2.05168587
VEC3   2.05168587    2.05168587    0.00000000
```

The `request.json` file is just a small JSON file that specifies what exactly AMS wants the external engine to calculate:

```
{
  "title": "GOSTep28",
  "quiet": false,
  "gradients": true,
  "stressTensor": false,
  "hessian": true,
  "dipoleMoment": false,
  "properties": true,
```

(continues on next page)

(continued from previous page)

```
"prevResults": "GOSTep27"
}
```

The job of the interfacing script is now to read these files, run the external program and convert its output into a format understood by AMS. Generally these are simple text files with the name of the property and the extension `.txt`. The bare minimum the interfacing script needs to produce is the file `energy.txt` containing a single number, i.e. the total energy in atomic units (Hartree). Other properties are optional, and it is easiest to go through the `request.json` entries one by one to see what AMS might request and what the interfacing script could produce in response.

title Just a title for this particular engine run. It can be passed on to the external program if desired, or can just be ignored.

quiet Whether AMS wants the external engine to write to standard output. This can be ignored in principle, but that might lead to really incomprehensible text output files of AMS if the external engine has to be called many times, e.g. for numerical derivatives.

gradients Whether or not to calculate nuclear gradients. The interface script should put the gradients in a file called `gradients.txt` with `nAtoms` lines of 3 real numbers each, in atomic units, i.e. Hartree/Bohr. Note that AMS wants the gradients, not forces (beware the - sign!).

stressTensor Whether to calculate the stressTensor for periodic systems. Should be written to `stresstensor.txt` in atomic units.

hessian Whether to calculate the Hessian, that is just the second derivative of the energy with respect to the nuclear coordinates, *without* applying any mass weighing to it. If the Hessian has been calculated, it should be put in `hessian.txt` as a 3 `nAtoms` x 3 `nAtoms` matrix in atomic units.

dipoleMoment If true, calculate the dipole moment and put it in `dipolemoment.txt` in atomic units, in one line with three numbers.

properties This is set to true if AMS considers this “geometry” important and wants the engine to calculate further properties that the user might be interested in. In practice this is set to “true” for e.g. the final converged step in a geometry optimization, so that the user can then let the engine calculate e.g. the band structure, which one would not want to do at all the steps *during* the optimization. AMS can’t do anything with the properties that the engine might calculate, but the files will remain on disk for people to inspect them.

prevResults This is the title of a previous similar calculation that the engine has already performed. These results can be accessed in `./$prevResults/`, so for the example above `GOSTep28` can access the results from the previous step in the geometry optimization in `./GOSTep27/`. This is just the directory in which the interfacing script was run when the `title` field was set to `GOSTep27`, so files that were written back then are still accessible. They can in principle be used to restart for example the SCF of the engine from step to step. Of course all of that has to be done by the interfacing script. The AMS driver does not know anything about how to restart the external program and can only point the interfacing script to the right location.

That is really all there is to the external engine: AMS prepares a folder with `system.xyz` and `request.json` and runs the user’s interfacing script in there, which has to take care of preparing the input for the external engine, running it, and putting the results in the text files that AMS expects, e.g. `gradients.txt`.

Note for properties that are in one way or another derivatives of the energy, it is generally ok if the external engine does not calculate what was requested by the AMS driver in `request.json`. If AMS requests, for example, the gradients from the external engine, but then does not find the `gradients.txt` in the directory after the interfacing script has run, it will just assume that the engine was not capable of calculating the gradient analytically. AMS will then just do the gradient numerically by rerunning the external engine for displaced geometries, reading only the energy from `energy.txt`. In this sense it is only absolutely required for the external engine to produce the energy, the rest can be done numerically by AMS if required. It is of course best to let the engine do as much as possible, especially if it implements analytical derivatives. Note that currently AMS can not calculate the Hessian numerically for engines that do not provide gradients. This is just a technical limitation, as it is of course possible to do a second derivative numerically, but it is just not implemented in AMS yet. (And it would also be a very slow way to calculate a Hessian.)

In addition to the `Execute` keyword that specifies the interfacing script, it is also possible to use the `ExecuteAtEnd` keyword to specify another command to be run at the end of the calculation, for example after the last step of a geometry optimization or molecular dynamics simulation. This command is run in the results directory.

Moreover, the `Engine External` block can also contain some information about the capabilities of the external engines:

```
Engine External
  Execute {...}
  ExecuteAtEnd {...}
  Supports
    DipoleMoment      {true|false}
    PeriodicityNone   {true|false}
    PeriodicityChain  {true|false}
    PeriodicitySlab   {true|false}
    PeriodicityBulk   {true|false}
  End
EndEngine
```

The normal engines that come with AMS (e.g. DFTB and BAND) produce the engine output files with extension `.rkf` in the results directory, see [here](#) (page 17). These files are also produced when an external engine is used and the information on them (anything related to the shape of the PES at that point, e.g. normal modes, phonons, ...) can be visualized normally with the graphical interface. In addition to each engine output `.rkf` file, external engines will also produce a correspondingly named folder per engine file, which is just the working directory of the interfacing script for that particular invocation of the external program. These folders just contain the full output of the external program and anything that the interfacing script might have produced. In this way users still have access to all results from the external program, even if these results were not communicated back to the AMS driver.

This last point is probably best illustrated with a simple example. Consider the following job that uses an external engine to do a linear transit calculation of ethane, going from the staggered to the eclipsed configuration, calculating normal modes at all converged points along the path:

```
AMS_JOBNAME=ethane_torsion $AMSBIN/ams << EOF

Task PESScan

System
  Atoms
    C      0.00000000    0.00000000    0.76576000
    C      0.00000000    0.00000000   -0.76576000
    H     -0.88668938    0.51193036    1.16677000
    H      0.88668938    0.51193036    1.16677000
    H      0.00000000   -1.02386071    1.16677000
    H      0.00000000    1.02386071   -1.16677000
    H     -0.88668938   -0.51193036   -1.16677000
    H      0.88668938   -0.51193036   -1.16677000
  End
End

PESScan
  CalcPropertiesAtPESPoints True
  ScanCoordinate
    nPoints 5
    Dihedral 3 1 2 6 60.0 0.0
  End
End

Properties
```

(continues on next page)

(continued from previous page)

```

    NormalModes True
End

Engine External
    ...
EndEngine

EOF

```

If we run this job and look into the results folder, we will find the standard `ams.log` and `ams.rkf` as well as the usual engine result files `PESPoint(1).rkf` to `PESPoint(5).rkf`. Just as if we had used one of the native AMS engines, like DFTB. Each of these files can be opened in AMSspectra to visualize the normal modes for this particular point. For an external engine we additionally have one folder per engine file, so for this example we would have `PESPoint(1)/` to `PESPoint(5)/`. These are the folders in which the interfacing script ran for these particular points, so they contain all the native output files of the external program.

9.4 Toy engines

The AMS driver comes with a simple built-in toy engine that implements a Lennard-Jones potential. This can sometimes be useful for testing, as many properties of the Lennard-Jones gas/liquid/solid can be calculated analytically and compared to the results from AMS. Note that the potential is exactly the same for all elements, i.e. the N-N bond has exactly the same strength as the He-He bond.

The Lennard-Jones engine only has three keywords, which define the shape of the potential:

```

Engine LennardJones
    RMin    float
    Eps     float
    Cutoff  float
EndEngine

```

Cutoff

Type Float

Default value 15.0

Unit Angstrom

Description The distance at which the interaction is truncated.

Eps

Type Float

Default value 1.0

Unit Hartree

Description The depth of the potential well.

RMin

Type Float

Default value 1.0

Unit Angstrom

Description The distance of the potential minimum.

9.5 Engine add-ons

Engine add-ons can be used to augment the results returned from an engine.

9.5.1 Dispersion corrections

For engines that do not natively support Grimme's D3 and D4 dispersion corrections, the D3Dispersion and D4Dispersion engine add-ons can be used to add such corrections.

```

EngineAddons
  D3Dispersion
    Damping [BJ | Zero]
    Enabled Yes/No
    Functional string
    a1 float
    a2 float
    s6 float
    s8 float
    sr6 float
  End
  D4Dispersion
    Enabled Yes/No
    Functional [...]
    Verbosity [Silent | Normal | Verbose | VeryVerbose]
    a1 float
    a2 float
    s6 float
    s8 float
    s9 float
  End
End

```

EngineAddons

D3Dispersion

Type Block

Description This block configures the add-on that adds the Grimme D3 dispersion correction to the engine's energy, gradients, and stress tensor.

Damping

Type Multiple Choice

Default value BJ

Options [BJ, Zero]

Description Type of damping: BJ (Becke-Johnson) or Zero. BJ is recommended for most applications.

Enabled

Type Bool

Default value No

Description Enables the D3 dispersion correction add-on.

Functional

Type String

Default value PBE

Description Use the D3 parameterization by Grimme for a given xc-functional. Accepts the same values as the `-func` command line option of the official `dftd3` program. Note: the naming convention is different from elsewhere in the AMS suite. For example, BLYP should be called `b-lyp`.

a1

Type Float

Description The `a1` parameter. Only used if Damping is set to BJ. If set, it overwrites the `a1` value for the chosen functional.

a2

Type Float

Description The `a2` parameter. Only used if Damping is set to BJ. If set, it overwrites the `a2` value for the chosen functional.

s6

Type Float

Description The `s6` parameter, global scaling parameter. If set, it overwrites the `s6` value for the chosen functional.

s8

Type Float

Description The `s8` parameter. If set, it overwrites the `s8` value for the chosen functional.

sr6

Type Float

Description The `sr6` parameter. Only used if Damping is set to Zero. If set, it overwrites the `sr6` value for the chosen functional.

D4Dispersion

Type Block

Description This block configures the add-on that adds the Grimme D4(EEQ) dispersion correction to the engine's energy, gradients, stress tensor and Hessian.

Enabled

Type Bool

Default value No

Description Enables the D4 dispersion correction add-on.

Functional

Type Multiple Choice

Default value PBE

Options [HF, BLYP, BPBE, BP86, BPW, LB94, MPWLYP, MPWPW91, OLYP, OPBE, PBE, RPBE, REVPBE, PW86PBE, RPW86PBE, PW91, PW91P86, XLYP, B97, TPSS, REVTPSS, SCAN, B1LYP, B3LYP, BHLYP, B1P86, B3P86, B1PW91, B3PW91, O3LYP, REVPBE0, REVPBE38, PBE0, PWP1, PW1PW, MPW1PW91, MPW1LYP, PW6B95,

TPSSH, TPSS0, X3LYP, M06L, M06, OMEGAB97, OMEGAB97X, CAM-B3LYP, LC-BLYP, LH07TSVWN, LH07SSVWN, LH12CTSSIRPW92, LH12CTSSIFPW92, LH14TCALPBE, B2PLYP, B2GPPLYP, MPW2PLYP, PWPB95, DSDBLYP, DSDPBE, DSDPBEB95, DSDPBEP86, DSDSVWN, DODBLYP, DODPBE, DODPBEB95, DODPBEP86, DODSVWN, PBE02, PBE0DH, DFTB(3ob), DFTB(mio), DFTB(pbc), DFTB(matsci), DFTB(ob2), B1B95, MPWB1K, REVTPSSH, GLYP, REVPBE0DH, REVTPSS0, REVDSDPBEP86, REVDSDPBEPBE, REVDSDBLYP, REVDODPBEP86, B97M, OMEGAB97M, R2SCAN]

Description Use the D4 parameterization by Grimme for a given xc-functional.

Verbosity

Type Multiple Choice

Default value Silent

Options [Silent, Normal, Verbose, VeryVerbose]

Description Controls the verbosity of the dftd4 code. Equivalent to the `-silent` and `-verbose` command line switches of the official dftd4 program.

a1

Type Float

Description The a1 parameter, see D4 article. The physically reasonable range for a1 is [0.0,1.0]. If set, it overwrites the a1 value for the chosen functional.

a2

Type Float

Description The a2 parameter, see D4 article. The physically reasonable range for a2 is [0.0,7.0]. If set, it overwrites the a2 value for the chosen functional.

s6

Type Float

Description The s6 parameter, see D4 article. The physically reasonable range for s6 is [0.0,1.0]. If set, it overwrites the s6 value for the chosen functional.

s8

Type Float

Description The s8 parameter, see D4 article. The physically reasonable range for s8 is [0.0,3.0]. If set, it overwrites the s8 value for the chosen functional.

s9

Type Float

Description The s9 parameter, see D4 article. If set, it overwrites the s9 value for the chosen functional.

9.5.2 Pressure

Pressure can be included using the following keyword (this can only be used for **3D periodic systems**):

```
EngineAddons
  Pressure float
End
```

EngineAddons**Pressure****Type** Float**Default value** 0.0**Unit** GPa**Description** Add a hydrostatic pressure term to the engine's energy and stress tensor. Can only be used for 3D periodic boundary conditions.

The engine's energy will include the following extra term: $P \cdot V$, where V is the volume of the unit cell. The engine's stress tensor will include the following extra term: $P \cdot \mathbb{I}$, where \mathbb{I} is the identity matrix. This can only be used for 3D periodic boundary conditions (i.e. bulk). The energy and stress tensor printed in output and written to the `.rkf` binary file will include these extra terms (i.e. the printed stress tensor is the sum of the internal stress and the external stress due to pressure).

When studying the effect that pressure has on the structure and properties of your system, one should generally start by *optimizing the structure* (page 43) **including the lattice vectors** under pressure. Properties such as *phonons* (page 177) or *elastic tensor* (page 148) can then be computed at the relaxed geometry. If you are investigating phase transitions under pressure (or if you simply expect some symmetry breaking) you should disable symmetry and/or perturb the initial geometry of your system.

See also:

Example: Periodic lattice optimization under pressure (page 243)

Warning: If you want to include pressure in *molecular dynamics* (page 77) calculations, you should not use this engine addon, but use the MD-specific pressure option.

9.5.3 Non-isotropic external stress

An **non-isotropic external stress tensor** can be included by using the following keywords:

```
EngineAddons
  ExternalStress
    StressTensorVoigt float_list
    UpdateReferenceCell Yes/No
  End
End
```

EngineAddons**ExternalStress****Type** Block**Description** This block configures the addon that adds external stress term to the engine's energy and stress tensor.**StressTensorVoigt****Type** Float List**Unit** a.u.**GUI name** External stress tensor

Description The elements of the external stress tensor in Voigt notation. One should specify 6 numbers for 3D periodic system (order: xx,yy,zz,yz,xz,xy), 3 numbers for 2D periodic systems (order: xx,yy,xy) or 1 number for 1D periodic systems.

UpdateReferenceCell

Type Bool

Default value No

Description Whether or not the reference cell should be updated every time the system changes (see documentation).

The energy and stress tensor printed in output and written to the `.rkf` binary file already include the corresponding extra terms, i.e. the printed stress tensor is the sum of the internal stress and the input-specified external stress.

When studying the effect that an external stress has on the structure and properties of your system, one should generally start by *optimizing the structure* (page 43) **including the lattice vectors** under the external stress (depending on the magnitude of the applied external stress, you might have to adjust the *stress energy per atom convergence threshold* (page 44)). If your system is symmetric, you should **disable symmetry** when optimizing structures under external stress. Be aware that the geometry optimization might go completely astray (e.g. the material will break apart) if you apply a) too large shear stress or b) too large tension stress (too large negative stresses for the diagonal values).

Following this paper⁴ from Parrinello and Rahman, the extra energy term due to a non-isotropic external stress⁵ is defined with respect to a *reference unit cell*, which in our case is the unit cell at the beginning of the simulation. If, during the simulation, large deformations of the unit cell occur, the above mentioned energy expression is only approximately consistent with the stress tensor. This affects the calculation differently depending on whether the stress tensor is computed by the engine or by AMS via numerical differentiation: a) If the stress tensor is computed directly by the engine, for large unit-cell deformations the energy might increase during the optimization; b) If the stress tensor is computed by AMS via numerical differentiation, the actual final value of the stress tensor might not match perfectly the external stress specified in the input (for large cell deformations, this error can be in the order of 10%).

After the optimization under external stress is converged, it is therefore good practice to validate the results. To do this, you should compute the *stress tensor* (page 147) at the optimized geometry by performing a *single point calculation* (page 41) **without** applying the external stress. The values in the computed stress tensor should have opposite sign compare to the external stress applied during the optimization. If the values differ too much, then you can run a second geometry optimization under external stress starting from the optimized geometry.

See also:

Example: Boron nitride optimization under external stress (page 245)

An alternative option is to set the key `ExternalStress%UpdateReferenceCell` to `True`; this will update the reference unit cell at every optimization step, effectively changing the definition of the energy expression at every geometry optimization step. The energy might not consistently go down during the optimization, but the resulting internal stress will match much better the applied external stress. This option should only be used during geometry optimizations (i.e. it should not be used when computing properties such as the elastic tensor).

See also:

Example: Graphene optimization under external stress (page 247)

⁴ M. Parrinello, and A. Rahman, *Polymorphic transitions in single crystals: A new molecular dynamics method*, *Journal of Applied Physics* 52, 7182 (1981) (<https://doi.org/10.1063/1.328693>)

⁵ The energy term due to a non-isotropic external stress is:

$$E_{\text{stress}} = p_h(V - V_0) + \frac{1}{2}V_0\text{Tr}[(\sigma - p_h\mathcal{I})(h_0^T)^{-1}h^T h h_0^{-1} - \mathcal{I}]$$

where: σ is the external stress tensor, p_h is the hydrostatic pressure associated with σ (i.e. the average of the diagonal elements of σ), V is the volume of the unit cell (for 2D periodic systems this is the area of the cell, and for 1D periodic systems is the length of the cell), V_0 is the volume of the *reference unit cell*, h are the lattice vectors in matrix form, h_0 are the lattice vectors of the reference unit cell in matrix form and \mathcal{I} is the identity matrix.

9.5.4 Atom energies

This add-on adds an element-dependent energy for each atom in a system, effectively applying an energy shift. It does not affect any other property.

```
EngineAddons
  AtomEnergies # Non-standard block. See details.
  ...
End
End
```

EngineAddons

Type Block

Description This block configures all the engine add-ons.

AtomEnergies

Type Non-standard block

Description Add an element-dependent energy per atom. On each line, give the chemical element followed by the energy (in atomic units).

The below example will add -0.5 Hartree for every H atom and -5.0 Hartree for every C atom in the system to the total energy. The add-on will ignore any atoms of other elements:

```
EngineAddons
  AtomEnergies
    H -0.5
    C -5.0
  End
End
```

9.5.5 Restraints

A restraint is potential energy function (a spring) attached to a certain coordinate, for example, an interatomic distance, the minimum of the potential energy being at the specified optimal value. A restraint can have one or two parameters: a ForceConstant and, for some types, a F(Inf) value. The ForceConstant parameter corresponds to second derivative of the restraint potential energy $\frac{d^2V(x)}{dx^2}$ for any x (harmonic restraints) or only at $x = 0$ (other restraints). Here, x is a deviation from the restraint's optimal value.

Geometry restraints can be added using the following input block:

```
Restraints
  Profile [Harmonic | Hyperbolic | Erf]
  fInfinity float
  Distance atomIdx1 atomIdx2 OptValue {Profile} {fInfinity}
  Angle
  Dihedral
  SumDist
  DifDist
End
```

The shape of the restraint potential function is defined by its profile type (Harmonic, Hyperbolic, or Erf) and the appropriate parameter(s). The Harmonic profile is most suitable for geometry optimizations but may result in very large forces that can be problematic in molecular dynamics. For MD simulations the Hyperbolic or Erf may be more suitable because the restraint force is bounded by a user-defined value. A Harmonic restraint is

defined as $V(x) = C\frac{x^2}{2}$. A Hyperbolic restraint is defined as $V(x) = a(\sqrt{1 + (bx)^2} - 1)$. An Erf restraint is defined as $V(x) = a(bx \cdot \operatorname{erf}(bx) + \frac{e^{-(bx)^2} - 1}{\sqrt{\pi}})$, which corresponds to an indefinite integral of the error function $\frac{dV(x)}{dx} = ab \cdot \operatorname{erf}(bx)$. The a and b parameters are calculated from the FInfinity and ForceConstant values. The Erf profile is very similar to Hyperbolic however its derivative converges to F(Inf) much faster.

The following keys define global parameters for all restraints.

Restraints

Profile

Type Multiple Choice

Default value Harmonic

Options [Harmonic, Hyperbolic, Erf]

GUI name Default restraint profile

Description Select the default type of restraint profile. The harmonic profile is most suitable for geometry optimizations but may result in very large forces that can be problematic in molecular dynamics. For MD simulations the Hyperbolic or Erf may be more suitable because the restraint force is bounded by a user-defined value. A per-restraint profile type can be specified after the ForceConstant value on the corresponding restraint line.

FInfinity

Type Float

Default value 1.0

GUI name Default F(inf)

Description Specify the default asymptotic value for the restraint force for the Hyperbolic and Erf profiles, in Hartree/Bohr or Hartree/radian. A per-restraint value can be specified after the profile type on the corresponding restraint line.

The following keys define individual restraints. The force constant, profile type and the F(Inf) value can be set for each restraint individually.

Restraints

Distance

Type String

Recurring True

Description Specify two atom indices followed by the distance in Angstrom and, optionally, by the ForceConstant (default is 1.0 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the distance between the two specified atoms at the given value. For periodic systems this restraint follows the minimum image convention.

Angle

Type String

Recurring True

Description Specify three atom indices $i\ j\ k$ followed by an angle in degrees and, optionally, by the ForceConstant (default is 0.3 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the i - j - k angle at the given value. For periodic systems this restraint follows the minimum image convention.

Dihedral

Type String

Recurring True

Description Specify four atom indices i j k l followed by an angle in degrees and, optionally, by the ForceConstant (default is 0.1 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the i-j-k-l dihedral angle at the given value. For periodic systems this restraint follows the minimum image convention.

DifDist

Type String

Recurring True

Description Specify four atom indices i j k l followed by the distance in Angstrom and, optionally, by the ForceConstant (default is 1.0 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the difference R(ij)-R(kl) at the given value. For periodic systems this restraint follows the minimum image convention.

SumDist

Type String

Recurring True

Description Specify four atom indices i j k l followed by the distance in Angstrom and, optionally, by the ForceConstant (default is 1.0 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the sum R(ij)+R(kl) at the given value. For periodic systems this restraint follows the minimum image convention.

A default value for the force constant depends on the restraint type: 1.0 Hartree/Bohr for Distance, SumDist and DifDist, 0.3 Hartree/radian for Angle and 0.1 Hartree/radian for Dihedral.

Below is an example if the Restraints block.

```
Restraints
  Profile Hyperbolic      # Change the default profile type
  fInfinity 10.0         # Change the asymptotic value for the restraint force away_
  ↪from optimum
#  Type      Atoms      OptValue  FC   Profile  F(Inf)
  Distance  1 2          5.0       1.0   Erf      1.0
  Angle     1 2 3        90.0
  Dihedral  4 1 2 3     180.0     0.1
#  The next two together are equivalent to imposing two distance restraints: R(23) to_
  ↪0.85 Angstrom and R(14) to 0.65 Angstrom
  SumDist   1 4 2 3     1.5       # Keep R(14)+R(23) as close to 1.5 Angstrom as possible
  DifDist   1 4 2 3     0.2       # Keep R(14)-R(23) as close to 0.2 Angstrom as possible
End
```

9.5.6 Wall potential

This add-on adds a time-independent spherical inward repulsive potential around the origin of the form

$$E_{\text{wall}}(r) = k \log(1 + \exp(-b(R - r)))$$

where k, R, and b are parameters, and r is the distance to the origin. The potential vanishes inside the radius R and yields a constant inward pointing force outside of R, determined by the parameters b (Gradient) and k (Prefactor)⁶.

⁶ S. Grimme, *Exploration of Chemical Compound, Conformer, and Reaction Space with Meta-Dynamics Simulations Based on Tight-Binding Quantum Chemical Calculations*, Journal of Chemical Theory and Computation 15, 2847-2862 (2019) (<https://doi.org/10.1021/acs.jctc.9b00143>)

```
EngineAddons
  WallPotential
    Enabled Yes/No
    Gradient float
    Prefactor float
    Radius float
  End
End
```

EngineAddons**Type** Block**Description** This block configures all the engine add-ons.**WallPotential****Type** Block**Description** This block configures the add-on that adds a spherical wall potential to the engine's energy and gradients.**Enabled****Type** Bool**Default value** No**Description** Enables the wall potential add-on.**Gradient****Type** Float**Default value** 10.0**Unit** 1/Angstrom**Description** The radial gradient outside the sphere.**Prefactor****Type** Float**Default value** 0.01**Unit** Hartree**Description** The multiplier for the overall strength of the potential.**Radius****Type** Float**Default value** 30.0**Unit** Angstrom**Description** The radius of the sphere, wherein the potential is close to zero.

10.1 ChemTraYzer

10.1.1 General information

The ChemTraYzer scripts, developed in the [Leonhard group](http://www.ltt.rwth-aachen.de/forschung/molekulare_thermodynamik/molekulare_thermodynamik/project/ChemTraYzer_Modellierung_chemi/) (http://www.ltt.rwth-aachen.de/forschung/molekulare_thermodynamik/molekulare_thermodynamik/project/ChemTraYzer_Modellierung_chemi/) at Aachen University, provide a methodology for deducing quantitative reaction models from reactive molecular dynamics (MD) simulations by identifying, quantifying, and evaluating elementary reactions of MD trajectories.

For usage with AMS, the scripts have been slightly modified and use RDKit instead of the openbabel-bindings. The original paper, Automated Discovery of Reaction Pathways, Rate Constants, and Transition States Using Reactive Molecular Dynamics Simulations, can be found [here](http://dx.doi.org/10.1021/acs.jctc.5b00201) (http://dx.doi.org/10.1021/acs.jctc.5b00201).

10.1.2 Graphical user interface

See the [combustion ReaxFF tutorial](#).

10.1.3 Command line execution

1. Open a terminal:

- on Linux: Ctrl+Alt+T usually works, otherwise browse your application menus
- on Windows: double click `adf_command_line.bat` in your ADF installation directory (e.g. `C:/ADF2016.106`), enter 'sh' to go to a shell

2. cd into the directory where your trajectories are located with the terminal

3. Assuming your trajectory file is located in directory called `jobname.results`, running ChemTraYzer is a two step procedure

- run the processing step (with default values)

```
$AMSBIN/amspython $AMSHOME/scripting/standalone/chemtrayzer/processing_scm.py  
jobname.results
```

- run the analysis step (with default values)

```
$AMSBIN/amspython $AMSHOME/scripting/standalone/chemtrayzer/analyzing_scm.py  
jobname.results
```

You will probably want to adjust the settings. A list of all available options is shown by calling the scripts without arguments, e.g. `$AMSBIN/amspython $AMSHOME/scripting/standalone/chemtrayzer/analyzing_scm.py`

10.1.4 Results

ChemTraYzer creates several output files, the most relevant ones being:

reac.spec.tab

1st line: indices of the species, e.g. S1;S2;S3;etc...

2nd line: SMILES of the species, e.g. ;C;CC;CCO;CCOO;CO;O;[H];etc...

3rd - end: timestep, counts of the species, e.g. 150;0;0;2;... // step 150, current count of S3 == 2

reac.rate.tab

1st line: indices of the reactions, e.g. t [steps];R0;R0*;R1;R1*; // a "*" marks the back reaction

2nd line: the reactions formulated with the species indices, e.g. S0 + S0 + S0 + S13 -> S65; // see reac.spec.tab

3rd line: the reactions formulated with sum formulas, e.g. CH3 + H -> CH4, etc...

4th - ...: timestep; rate constant of the reaction; rate constant of the back reaction; etc...

reac.pic/

folder containing the 2D structures (if obabel was able to create them) reac.pic/xyz/ xyz files of the species as they first appeared in the trajectory

10.2 Trajectory Analysis

analysis is a standalone program that performs analysis of molecular dynamics trajectories created with AMS. It can produce histograms and radial distribution functions. It is also used under the hood in [AMSmovie \(MD Properties menu bar\)](#).

This is an example showing how to compute the oxygen-oxygen radial distribution function of a MD simulation using the analysis utility program:

```
$AMSBIN/analysis <<eor
Task RadialDistribution

TrajectoryInfo
  Trajectory
    KFFilename ams.results/ams.rkf
    Range 1 1000 2
  End
End

RadialDistribution
  NBins 1000
  AtomsFrom
    Element O
  End
  AtomsTo
    Element O
```

(continues on next page)

(continued from previous page)

```

End
End
eor

```

The analysis program reads one or more trajectory files (filename.rkf) from an AMS molecular dynamics (MD) or a Grand Canonical Monte Carlo (GCMC) simulation. The file information is supplied in the `TrajectoryInfo` input block. In this block, a separate `Trajectory` subblock needs to be supplied for each trajectory file. The `Trajectory` subblock contains a mandatory keyword `KFFilename`, and an optional keyword `Range`. The latter contains the initial frame to be read, the final frame to be read, and optionally the stepsize. By default all frames on the trajectory file are read.

```

TrajectoryInfo
  NBlocksToCompare integer
  Trajectory
    KFFilename string
    Range integer_list
  End
End

```

TrajectoryInfo**Type** Block**Description** All the info regarding the reading of the trajectory files.**NBlocksToCompare****Type** Integer**Default value** 1**Description** Get an error estimate by comparing histograms for `NBlocks` time blocks of the trajectory.**Trajectory****Type** Block**Recurring** True**Description** All info regarding the reading of a single trajectory file.**KFFilename****Type** String**Default value** ams.rkf**Description** The name of the AMS trajectory file.**Range****Type** Integer List**Description** Two or three values: start frame, end frame, step size.

All tools in the analysis program provide an option to obtain information on the equilibration of the simulation. If the optional keyword `NBlocksToCompare` in the `TrajectoryInfo` block is set to a value N higher than 1, the trajectory is divided into N blocks, and the analysis results for each block are compared. The variation in the analysis result is provided as a standard deviation.

10.2.1 Radial Distribution Function (RDF)

The Analysis tool computes radial distribution functions $g(r)$ if the Task keyword is set to RadialDistribution.

```
Task [RadialDistribution | Histogram | AutoCorrelation]
```

Task

Type Multiple Choice

Options [RadialDistribution, Histogram, AutoCorrelation]

Description The analysis task.

Further details on the radial distribution functions are then set in the RadialDistribution block. If more than one RadialDistribution block is present in the input, more than one radial distribution function will be computed. The result is printed to output as text, as well as stored in a binary file (analysis.kf).

Description

A radial distribution function $g(r)$, or pair correlation function, is a density of distances between particles, relative to the average distance density. The x -axis variable represents a distance r , while the y -axis represents the relative density of that distance. For a complete homogeneous system of particles the $g(r)$ values for the distances between all particles equals 1 everywhere.

Two sets of atoms S_{from} and S_{to} , of length n_{from} and n_{to} respectively, are specified with the keywords AtomsFrom and AtomsTo in the RadialDistribution block. As a result the program computes $n_{\text{from}} * n_{\text{to}}$ distances r_{ij}^s between atom i in S_{from} and atom j in S_{to} for each trajectory frame s out of a total of n_{frames} frames.

A normalized histogram is then computed from these distances, resulting in a function $N(r)$.

$$N(r) = \frac{1}{n_{\text{frames}}} \sum_{s=1}^{n_{\text{frames}}} \sum_{i=1}^{n_{\text{from}}} \sum_{j=1}^{n_{\text{to}}} \delta(r_{ij}^s - r).$$

This histogram is converted to a density, by dividing all values $N(r)$ with the volume $V(r) = 4\pi r^2 dr$ of a sphere-slice at radius r with thickness dr .

The density is further converted to a relative density by dividing with the total density of the system $\rho_{\text{tot}} = \frac{n_{\text{from}} * n_{\text{to}}}{V_{\text{tot}}}$, yielding the final radial distribution function $g(r)$.

$$g(r) = \frac{N(r)}{V(r) * \rho_{\text{tot}}}$$

Options

Non-periodic systems The above equation assumes that the volume V_{tot} of the system is a well-defined quantity. This assumption is correct for systems with 3D periodicity, where the V_{tot} is defined as the volume of the periodic cell. In such a system the value of r can be no larger than r_{max} , the radius of the largest sphere that can be placed inside the periodic cell.

If a system is non-periodic in one or more direction, then the program still computes a $g(r)$, only if the radius r_{max} is supplied by the user with the Range keyword in the RadialDistribution block. The radius is the second value supplied.

```
RadialDistribution
  Range float_list
End
```

RadialDistribution

Type Block

Recurring True

Description All input related to radial distribution functions.

Range

Type Float List

Description Either one, two, or three real values. If one it is the stepsize. If two, it is the minimum value and the maximum value. If three, it is the minimum value, the maximum value, and the stepsize. The stepsize overrides NBins.

In this case the volume V_{tot} is assumed to be the volume of a sphere with radius r_{max} .

NPT simulations The above equation further assumes that the volume V_{tot} is constant throughout the simulation. The $g(r)$ of the trajectory from an NPT simulation can still be computed, and in this case V_{tot} is the average value of the volume of the periodic cell.

Simulations with varying numbers of atoms The above equation also assumes that n_{from} and n_{to} remain constant throughout the simulation. However, in a Molecular Gun simulation particles can be added to the system, and in a GCMC simulation particles can be both added and removed from the system. Nonetheless, the program still computes a $g(r)$ in these situations.

If the `AtomsFrom` and `AtomsTo` blocks contain element names (supplied with the recurring `Element` keyword), then every time atoms are added to or removed from the system, the sets of atoms S_{from} and S_{to} are re-evaluated.

If the `AtomsFrom` and `AtomsTo` blocks contain atom numbers (supplied with the recurring `Atom` keyword), these numbers are updated in the sets S_{from} and S_{to} every time atoms are added to or removed from the system. If one of the atoms from the set disappears, the number of distances contributing to the $g(r)$ decreases.

Note: Currently, the values of n_{from} and n_{to} in the normalization factor are taken from the last frame of the simulation.

Warning: If multiple trajectories are supplied, and the number of atoms changes between the end of one trajectory and the beginning of another, this may result in an error in the atom numbers used by the program internally.

10.2.2 Histogram

The Analysis program computes histograms if the `Task` keyword is set to `Histogram`.

```
Task [RadialDistribution | Histogram | AutoCorrelation]
```

Task

Type Multiple Choice

Options [RadialDistribution, Histogram, AutoCorrelation]

Description The analysis task.

Further details on the histogram need to be specified in the `Histogram` block. If more than one `Histogram` block is present in the input, more than one histogram will be computed. The result is printed to output as text, as well as stored in a binary file (`analysis.kf`). By default the histogram contains the number of occurrences of a certain value, but the normalized occurrence is provided if the keyword `Normalized` in the `Histogram` block is specified.

```
Histogram
  Normalized Yes/No
End
```

Histogram

Normalized

Type Bool

Default value No

Description Give the normalized histogram.

Histograms can be computed for every quantity stored on the molecular dynamics trajectory file (ams.rkf) in the section History. Example quantities are PotentialEnergy, KineticEnergy, TotalEnergy, Temperature. In the histogram block, this quantity is selected with the keyword Variable in the Axis subblock. If more than one Axis subblock is present, the dimensionality of the histogram is increased: Three Axis subblocks result in a 3D histogram.

For each histogram axis, the number of bins can be selected with the NBins keyword in the Axis block, in which case the range of values along each axis is automatically determined. The default NBins value is 100.

Alternatively, a range and a stepsize can be selected with the keyword Range in the Axis subblock. The keyword Range can contain one, two, or three values: 1: Only a stepsize. 2: A smallest value and a largest value. 3: A smallest value, a largest value, and the stepsize.

```

Histogram
  Axes
    Axis
      NBins integer
      Range float_list
      Variable string
    End
  End
End

```

Histogram

Type Block

Recurring True

Description All input related to histograms.

Axes

Type Block

Description Specifications for the histogram axes.

Axis

Type Block

Recurring True

Description Specifications for a single histogram axis.

NBins

Type Integer

Default value 100

Description The number of bins along the histogram axis.

Range

Type Float List

Description Either one, two, or three real values. If one it is the stepsize. If two, it is the minimum value and the maximum value. If three, it is the minimum value, the maximum value, and the stepsize. The stepsize overrides NBins.

Variable**Type** String**Description** The quantity along the histogram axis.

10.2.3 Autocorrelation Functions

The Analysis program computes autocorrelation functions (ACF) if the `Task` keyword is set to `AutoCorrelation`.

```
Task [RadialDistribution | Histogram | AutoCorrelation]
```

Task**Type** Multiple Choice**Options** [RadialDistribution, Histogram, AutoCorrelation]**Description** The analysis task.

Further details need to be specified in the `AutoCorrelation` block. If more than one `AutoCorrelation` block is present in the input, more than one ACF will be computed. The result is printed to output as text, as well as stored in a binary file (`analysis.kf`).

```
AutoCorrelation
  Atoms
    Atom integer
    Element string
  End
  DataReading [Auto | AtOnce | BlockWise]
  InputValues
    Values float_list
  End
  MaxStep integer
  NPointsHighestFreq integer
  Normalized Yes/No
  Property [Velocities | DipoleMomentFromCharges | InputValues |
↔DiffusionCoefficient | DipoleDerivativeFromCharges | PressureTensor]
  TimeStep float
  UseAllValues Yes/No
  UseTimeDerivative
    Enabled Yes/No
    ProjectOutRotations Yes/No
  End
  VecElements
    Index integer
  End
End
```

AutoCorrelation**Atoms****Type** Block

Description Relevant if `Property` is set to `Velocities`, `DipoleMomentFromCharges`, `DipoleDerivativeFromCharges`, or `DiffusionCoefficient`. Atom numbers or elements for the set of atoms for which the property is read/computed. By default all atoms are used.

Atom

Type Integer

Recurring True

Description Atom number.

Element

Type String

Recurring True

Description Element Symbol Atom.

DataReading

Type Multiple Choice

Default value Auto

Options [Auto, AtOnce, BlockWise]

Description The KF data can be read in and handled once, or blockwise. The former is memory intensive, but mostly faster. If Auto is selected, the data is read at once if it is less than 1 GB, and blockwise if it is more.

InputValues

Type Block

Description Relevant is Property is set to InputValues. All input values (a vector on each line).

Values

Type Float List

Recurring True

Description The values at each step (on a single line)

MaxStep

Type Integer

Description The maximum interval of the autocorrelation. The default is half of the number of provided frames.

NPointsHighestFreq

Type Integer

Default value 4

Description The number of points (timesteps) used for the highest frequency displayed in spectrum. This determines up to which frequency the spectrum is displayed. If the spacing between time-steps used for the ACF is 1 fs, then by default the maximum frequency displayed is 0.25 fs⁻¹ (or 8339 cm⁻¹). A higher number selected here, will result in a lower maximum frequency returned by the program. The default value is 4. and the lowest possible value (spectrum up to highest possible frequency) is 2.

Normalized

Type Bool

Default value Yes

Description Determines if the ACF is normalized. Keyword is overruled (set to False) if Property is set to DiffusionCoefficient.

Property**Type** Multiple Choice**Default value** DipoleMomentFromCharges**Options** [Velocities, DipoleMomentFromCharges, InputValues, DiffusionCoefficient, DipoleDerivativeFromCharges, PressureTensor]**Description** Compute the ACF either from velocities (from rkf), the dipole moment (from atomic charges in rkf), or from values specified in input. If DiffusionCoefficient is selected the unnormalized velocity autocorrelation function is computed and integrated.**TimeStep****Type** Float**Description** Relevant if Property is set to InputValues. The time separating the entries (in fs). If Property is set to Velocities, DipoleMomentFromCharges, or DiffusionCoefficient, then the property can be obtained from an RKF file, and the timestep is read from the RKF file as well. The read value then overrides this keyword.**UseAllValues****Type** Bool**Default value** No**Description** By default the same number of values are used for each t-step in the ACF. This has the advantage that all values in the ACF are equally reliable, but it does mean that for the smaller timesteps much of the data is not used. To switch this off and use all data, UseAllValues can be set to true**UseTimeDerivative****Type** Block**Description** Possibly use the time derivative of the selected property (e.g. velocity or dipole moments).**Enabled****Type** Bool**Default value** No**Description** Enable the use of the time derivative of the property.**ProjectOutRotations****Type** Bool**Default value** No**Description** Take the rotations out of the time derivative.**VecElements****Type** Block**Description** A set of indices referring to a subset of the property vector. Works in combination with the atoms block. For example, in combination with the property Velocities, the Atoms block allows the selection of a subset of atoms, while the VecElements block allows the selection of a subset of vector elements (e.g. 1 and 2 for the elements x and y). Currently not implemented with InputValues.**Index**

Type Integer**Recurring** True**Description** Element of the property vector.

Description

An autocorrelation function $C(t)$ describes the average correlation (overlap) of a (vector) property \mathbf{A} with itself as a function of time.

$$C(t) = \langle \mathbf{A}(0) \cdot \mathbf{A}(t) \rangle$$

The average runs over all time-intervals $(t_0, t_0 + t), (t_1, t_1 + t), \dots, (t_N, t_N + t)$, with $t_N = t_n - t_m$. Here n is the total number of simulation steps in the trajectory, and m is the number of discrete t values for which $C(t)$ is computed. The value m can be set with the keyword `MaxStep`, and defaults to half the total number of simulation steps. If applicable, the average also runs over all possible contributions to \mathbf{A} at each simulation timestep. The normalized autocorrelation function $c(t)$ describes the decorrelation of the property with time, and always starts at 1.0 at $t = 0$.

$$c(t) = \frac{\langle \mathbf{A}(0) \cdot \mathbf{A}(t) \rangle}{\langle \mathbf{A}(0) \cdot \mathbf{A}(0) \rangle}$$

In most cases short timescale fluctuations are important, so frequent storage of the desired property is required (when preparing the molecular dynamics simulation, set the `Frequency` keyword in the `Trajectory` block of the `MolecularDynamics` settings low, preferably to 1).

A power spectrum is automatically computed by Fourier transform of the autocorrelation function, and provides information on the frequencies of the signal. When the selected property is the dipole moment, the power spectrum matches the IR spectrum.

Options

Autocorrelation functions can be computed for different simulation properties: 1) Dipole moments from atomic charges 2) Velocities 3) User provided values.

```
AutoCorrelation
  Property [Velocities | DipoleMomentFromCharges | InputValues |
↪DiffusionCoefficient | DipoleDerivativeFromCharges | PressureTensor]
End
```

AutoCorrelation

Type Block**Recurring** True**Description** All input related to auto correlation functions.

Property

Type Multiple Choice**Default value** DipoleMomentFromCharges**Options** [Velocities, DipoleMomentFromCharges, InputValues, DiffusionCoefficient, DipoleDerivativeFromCharges, PressureTensor]**Description** Compute the ACF either from velocities (from rkf), the dipole moment (from atomic charges in rkf), or from values specified in input. If `DiffusionCoefficient` is selected the unnormalized velocity autocorrelation function is computed and integrated.

With the keyword `Normalized` a normalized ACF is computed, and with the keyword `MaxStep` the number of values n in the autocorrelation function ($t = [0, t_1, t_2, \dots, t_n]$) can be set. The default value is half of the total number of simulation steps used.

A subset of atoms for which the property `A` should be selected/computed can be provided in the block `Atoms`. The block can contain element names (recurring keyword `Element`), or individual atom numbers (recurring keyword `Atom`).

```
AutoCorrelation
  Atoms
    Atom integer
    Element string
  End
End
```

AutoCorrelation

Type Block

Recurring True

Description All input related to auto correlation functions.

Atoms

Type Block

Description Relevant if Property is set to `Velocities`, `DipoleMomentFromCharges`, `DipoleDerivativeFromCharges`, or `DiffusionCoefficient`. Atom numbers or elements for the set of atoms for which the property is read/computed. By default all atoms are used.

Atom

Type Integer

Recurring True

Description Atom number.

Element

Type String

Recurring True

Description Element Symbol Atom.

10.2.4 Diffusion Coefficient

The diffusion coefficient can be computed as the integral over the velocity autocorrelation function.

$$D = \frac{1}{3} \int_{t=0}^{t=t_{max}} \langle \mathbf{v}(0) \cdot \mathbf{v}(t) \rangle dt$$

The factor $\frac{1}{3}$ corrects for the dimension of the system, which we assume to be always 3.

The diffusion coefficient is computed if the task `AutoCorrelation` is selected, and if in the `AutoCorrelation` block `DiffusionCoefficient` is selected as the `Property`.

```
$AMSBIN/analysis <<eor
  Task AutoCorrelation
  AutoCorrelation
    Property DiffusionCoefficient
```

(continues on next page)

End eor

AutoCorrelation**Type** Block**Recurring** True**Description** All input related to auto correlation functions.**Property****Type** Multiple Choice**Default value** DipoleMomentFromCharges**Options** [Velocities, DipoleMomentFromCharges, InputValues, DiffusionCoefficient, DipoleDerivativeFromCharges, PressureTensor]**Description** Compute the ACF either from velocities (from rkf), the dipole moment (from atomic charges in rkf), or from values specified in input. If DiffusionCoefficient is selected the unnormalized velocity autocorrelation function is computed and integrated.

Again, a subset of atoms can be selected with the subblock `Atoms`.

The value of the diffusion coefficient is written to the output, as well as to the KF file.

10.3 VCD Analysis: VCDtools

The auxiliary program *VCDtools* provides insight regarding the origin of the VCD intensity of a given normal mode. This is done by analyzing and visualizing the various contributions to the total Electric (E) and Magnetic (M) Dipole Transition Moments (DTM). *e.g.*, the contributions from atoms, electrons, nuclei and molecular fragments. The *VCDtools* program is accessible through the AMS-GUI and allows one to:

- 1) Scale the size of the atoms with the magnitude of their atomic EDTM or MDTM contributions.
- 2) Visualize the various contributions to the EDTMs and MDTMs by arrows.
- 3) Perform a General Coupled Oscillator (GCO) analysis to evaluate the interactions between the different parts of a molecule.
- 4) Decompose the total EDTM and MDTM into nuclear and electronic contributions to gain insight into effects induced in the VCD spectra by charge transfer.
- 5) Perform a VCD robustness analysis.

See also:

Tutorial: [Analysis of the VCD spectrum of Oxirane with VCDtools](#)

VCDtools requires only the standard result file `adf.rkf` from an VCD calculation using the *ADF* engine. Once this file is opened with *AMSSpectra* the *VCDtools* program will be accessible through the standard user interface of *AMSSpectra*.

VCDtools produces one ASCII file, `VCDtools.out` where all output from the program is printed. This file will be located in the `.results` folder in the directory of the opened `adf.rkf` result file. Additionally, *VCDtools* will run on the background every time a new normal mode is selected. It then computes the atomic EDTM and MDTM which can be displayed inside *AMSSpectra* in various ways.

The following references should be cited when publishing results obtained with VCDtools Refs.¹²³.

10.3.1 General Theory

The VCD intensity associated with the fundamental vibrational transition for a given normal mode j is given by the rotational strength (RS):

$$R_{01}(j) = -i\vec{E}_{01}^{tot}(j) \cdot \vec{M}_{10}^{tot}(j)$$

where $E_{01}^{tot}(j)$ and $M_{10}^{tot}(j)$ are the total EDTM and MDTM of normal mode j , i is the unit imaginary number.

When the harmonic approximation the total EDTM and MDTM can be written as sums of atomic contributions:

$$\vec{E}_{01}^{tot}(j) = \sum_{\lambda=1}^N \vec{E}_{01}^{\lambda}(j)$$

$$\vec{M}_{10}^{tot}(j) = \sum_{\lambda=1}^N \vec{M}_{10}^{\lambda}(j)$$

where N is the total number of atoms in the molecule and λ runs over all atoms.

10.3.2 General Coupled Oscillator Analysis

The GCO analysis computes the contribution to total rotational strength from the interaction of two molecular fragments. This information provides important insight regarding the source of the VCD intensity and the robustness of the computed VCD sign. In the following, a brief summary of the GCO theory originally published in Ref.¹ is given.

Following the original coupled oscillator VCD mechanism, the atoms of a molecule are grouped into three fragments: **A**, **B**, and **R**. The fragments **A** and **B** represent the important Coupling Oscillator (CO) fragments, while fragment **R** is the ancillary fragment containing the rest of the atoms. As a result, the total RS can be decomposed in three components:

$$R_{01}(j) = R_{01}^{GCO}(j) + R_{01}^{IF}(j) + R_{01}^R(j)$$

The first contribution $R_{01}^{GCO}(j)$, is the GCO contribution to the RS and is given by the interaction between the EDTM and MDTM of fragments **A** and **B**:

$$R_{01}^{GCO}(j) = -i \cdot \left[\vec{E}_{01}^A(j) \cdot \vec{M}_{10}^B(j) + \vec{E}_{01}^B(j) \cdot \vec{M}_{10}^A(j) \right]$$

The $R_{01}^{IF}(j)$ term represents the contribution from the individual fragments (IF) **A** and **B** without any interaction with other fragments:

$$R_{01}^{IF}(j) = -i \cdot \left[\vec{E}_{01}^A(j) \cdot \vec{M}_{10}^A(j) + \vec{E}_{01}^B(j) \cdot \vec{M}_{10}^B(j) \right]$$

and the $R_{01}^R(j)$ contribution contains all contributions from fragment **R**:

$$R_{01}^R(j) = -i \cdot \left[\vec{E}_{01}^A(j) \cdot \vec{M}_{10}^R(j) + \vec{E}_{01}^R(j) \cdot \vec{M}_{10}^A(j) \right]$$

$$- i \cdot \left[\vec{E}_{01}^B(j) \cdot \vec{M}_{10}^R(j) + \vec{E}_{01}^R(j) \cdot \vec{M}_{10}^B(j) + \vec{E}_{01}^R(j) \cdot \vec{M}_{10}^R(j) \right]$$

¹ V.P. Nicu, *Revisiting an old concept: the coupled oscillator model for VCD. Part I: the generalised coupled oscillator mechanism and its intrinsic connection to the strength of VCD signals*, *Physical Chemistry Chemical Physics* 18, 21202 (2016) (<https://doi.org/10.1039/C6CP01282E>)

² V.P. Nicu, J. Neugebauer and E.J. Baerends, *Effects of Complex Formation on Vibrational Circular Dichroism Spectra*, *Journal of Physical Chemistry A* 112, 6978 (2008) (<https://doi.org/10.1021/jp710201q>)

³ M.A.J. Koenis, O. Visser, L. Visscher, W.J. Buma, V.P. Nicu, *GUI Implementation of VCDtools, A Program to Analyze Computed Vibrational Circular Dichroism Spectra*, *J. Chem. Inf. Model* 60, 259 (2020) (<https://pubs.acs.org/doi/abs/10.1021/acs.jcim.9b00956>)

In the above equations $\vec{E}_{01}^{\mathbf{X}}(j)$ and $\vec{M}_{01}^{\mathbf{X}}(j)$, with $\mathbf{X} = (\mathbf{A}, \mathbf{B}, \mathbf{R})$, are the EDTM and MDTM associated with the three fragments.

Using the origin dependency of the MDTM, the GCO contribution can be rewritten in a form similar to the original CO term:

$$R_{01}^{\text{GCO}}(j) = \frac{\pi\nu_j}{c} \cdot \vec{Y}^{\text{GCO}}(j) \cdot \left[\vec{E}_{01}^{\mathbf{A}}(j) \times \vec{E}_{01}^{\mathbf{B}}(j) \right]$$

where $\vec{Y}^{\text{GCO}}(j)$ is the general coupled oscillator vector.

As shown in ref.¹ most normal modes of a molecule (i.e. not just the carbonyl stretching modes) can be interpreted in terms of the GCO mechanism. That is, for most normal modes one can define the GCO fragments **A** and **B** fragments in such a way that the $R_{01}^{\text{GCO}}(j)$ term represent the dominant contributions to the rotational strengths. This second expression for $R_{01}^{\text{GCO}}(j)$ does not depend on the MDTM (which is origin dependent). As such, it allows one to interpret the VCD intensity of a mode in terms of interacting EDTMs that are associated with the various moieties of a molecule and their relative orientation.

The identification of the GCO fragments (which are normal mode dependent) is not always trivial, especially, in molecules without symmetry. Consequently, *VCDtools* offers several options for dividing the molecule into fragments. In *AMSSpectra* these fragments are referred as 'Regions' in analogy to the regions in the *AMSinput* program. Under the **Regions**-menu there are many options to set, alter and save the fragments. Additionally, *VCDtools* is able to make a guess for fragments **A** and **B** as will be discussed in more detail below.

10.3.3 Available options

Beside the standard calculation and visualization of the atomic contributions to the normal mode (NM) motion, EDTM and MDTM, *VCDtools* offers three more advanced tools that can be utilized when analyzing VCD spectra inside *AMSSpectra*:

- GCO Analysis
- Guess GCO Fragments
- NM Localization on Regions

The first option, "*GCO Analysis*", uses the above equations to decompose the RS in its different contributions. Printing both the values and important angles between the vectors. Before this option can be run two regions inside the molecule should be defined as fragments **A** and **B**. It is important that these regions do not contain the same atoms. Also since the decomposition is different for each NM, a specific NM should be selected.

The second option, "*Guess Fragments*", guesses which atoms belong the fragments **A** and **B** for a selected normal mode. In doing so the $R_{01}^{\text{GCO}}(j)$ is maximized while keeping $R_{01}^{\mathbf{R}}(j)$ low. Additionally, it ensures that the fragments are localized on a part of the molecule.

The third option, "*NM Localization on Regions*", computes the percentage of the mass-weighted normal mode motion that is located on the atoms in the selected regions. One or multiple regions can be computed at the same time and the localizations are determined for all modes within a selected frequency window.

EXAMPLES

11.1 Geometry optimization

11.1.1 Example: Simple geometry optimization

Download GO_formaldehyde_noSCC.run

```
#!/bin/sh

$AMSBIN/ams << EOF

  Task GeometryOptimization

  System
    Atoms [Bohr]
      C   0.0   0.0   -1.0
      O   0.0   0.0   1.247
      H   0.0  -1.738  -2.097
      H   0.0   1.738  -2.097
    End
  End

  Engine DFTB
    ResourcesDir Dresden
    Model DFTB0
    DispersionCorrection Auto
  EndEngine

EOF
```

11.1.2 Example: Two-stage geometry optimization with initial Hessian

Download 2StepGO.run

```
#!/bin/sh

# Preoptimization with DFTB and calculation of the Hessian
# =====
#
# We will reuse the geometry optimized at the DFTB level as a starting point for
# the DFT geometry optimization. We will also calculate the real Hessian with
```

(continues on next page)

(continued from previous page)

```
# DFTB and use that as the initial Hessian for the Quasi-Newton based
# optimization at the DFT level. DFTB is so fast compared to DFT, that all of
# this is basically instantaneous. Our goal here is really just to reduce the
# number of steps in the DFT geometry optimization. If we save just a single
# step there, the initial DFTB calculation will already have paid for itself ...
```

```
AMS_JOBNAME=dftb_preopt $AMSBIN/ams << EOF
```

```
# Specify the system geometry: Aspirin
```

```
System
```

```
  Atoms
```

```
  C      0.000000  0.000000  0.000000
  C      1.402231  0.000000  0.000000
  C      2.091015  1.220378  0.000000
  C      1.373539  2.425321  0.004387
  C     -0.034554  2.451759  0.016301
  C     -0.711248  1.213529  0.005497
  O     -0.709522  3.637718  0.019949
  C     -2.141910  1.166077 -0.004384
  O     -2.727881  2.161939 -0.690916
  C     -0.730162  4.530447  1.037168
  C     -0.066705  4.031914  2.307663
  H     -0.531323 -0.967191 -0.007490
  H      1.959047 -0.952181 -0.004252
  H      3.194073  1.231720 -0.005862
  H      1.933090  3.376356 -0.002746
  O     -2.795018  0.309504  0.548870
  H     -2.174822  2.832497 -1.125018
  O     -1.263773  5.613383  0.944221
  H     -0.337334  4.693941  3.161150
  H      1.041646  4.053111  2.214199
  H     -0.405932  3.005321  2.572927
```

```
End
```

```
End
```

```
# Do a geometry optimization.
```

```
Task GeometryOptimization
```

```
# Also compute the Hessian at the optimized geometry.
```

```
Properties
```

```
  Hessian True
```

```
End
```

```
# Settings for the DFTB engine:
```

```
Engine DFTB
```

```
  Model DFTB3
```

```
  ResourcesDir DFTB.org/3ob-3-1
```

```
EndEngine
```

```
EOF
```

```
# Geometry optimization with DFT
```

```
# =====
```

```
AMS_JOBNAME=dft_opt $AMSBIN/ams << EOF
```

(continues on next page)

(continued from previous page)

```

# Start from the geometry that is already optimized at the DFTB level.
LoadSystem
  File dftb_preopt.results/dftb.rkf
End
# (equivalent to loading the system from dftb_preopt.results/ams.rkf)

Task GeometryOptimization

GeometryOptimization
  InitialHessian
    # Load the DFTB Hessian as the initial Hessian for the
    # Quasi-Newton based optimizer.
    Type FromFile
    File dftb_preopt.results
  End
End

# Settings for the BAND engine:
Engine BAND
  Basis Type=TZP
  XC GGA=PBE
EndEngine

EOF

```

11.1.3 Example: Periodic lattice optimization under pressure

Download `Diamond_under_pressure.run`

```

#!/bin/sh

# Calculate the phonon dispersion curves for diamond under pressure.

# Loop over pressure values (in GPa):
for P in -40 0 40 160 ; do
AMS_JOBNAME=pressure_$P $AMSBIN/ams << EOF

  Task GeometryOptimization

  System
    Atoms
      C -0.44625 -0.44625 -0.44625
      C 0.44625 0.44625 0.44625
    End
    Lattice
      0.0 1.785 1.785
      1.785 0.0 1.785
      1.785 1.785 0.0
    End
  End

  GeometryOptimization
    OptimizeLattice Yes
    Convergence Gradients=1e-5 StressEnergyPerAtom=1E-5
    Method Quasi-Newton
  End
End

```

(continues on next page)

(continued from previous page)

```

End

EngineAddons
  Pressure $P
End

Properties
  # Request the calculation of phonons at the optimized geometry.
  Phonons Yes
End

NumericalPhonons
  SuperCell
    2 0 0
    0 2 0
    0 0 2
  End
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/mio-1-1
  KSpace
    Type Symmetric
    Symmetric KInteg=5
  End
  Technical AnalyticalStressTensor=False
EndEngine

EOF
done

```

11.1.4 Example: Phase Transition Due To External Nonuniform Stress

Download `Diamond_from_pressure.run`

```

#!/bin/sh

# Starting from hexagonal graphite we optimize the crystal
# under the influence of a non-uniform stress tensor.
# The system will undergo a phase transition to a diamond structure
AMS_JOBNAME=Graphite2Diamond $AMSBIN/ams << eor

Task GeometryOptimization
EngineAddons
  ExternalStress
    StressTensorVoigt 0 0 0.01 0 -0.0003 0
  End
End
System
  Atoms
    C 0.0000 0.0000 1.6507
    C 0.0000 0.0000 -1.6507
    C 0.0000 -1.4225 1.6507
    C 0.0000 1.4225 -1.6507

```

(continues on next page)

(continued from previous page)

```

End
  Lattice
    1.2319 -2.1338 0.0000
    1.2319 2.1338 0.0000
    0.0000 0.0000 6.6029
  End
End
GeometryOptimization
  MaxIterations 300
  OptimizeLattice Yes
  Convergence
    Energy 1.0e-6
    Gradients 1.0e-5
    Step 1.0e-4
  End
End
UseSymmetry No
Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/3ob-3-1
  DispersionCorrection D3-BJ
  KSpace
    Quality Basic
  End
EndEngine

eor

# The diamond structure is still compressed under the influence of the external_
↪stress.
# We now disable this term and relax the unit cell with the same DFTB model
AMS_JOBNAME=DiamondRelaxation $AMSBIN/ams << eor

Task GeometryOptimization
LoadSystem
  File Graphite2Diamond.results/ams.rkf
End
GeometryOptimization
  MaxIterations 300
  OptimizeLattice Yes
  Convergence
    Energy 1.0e-6
    Gradients 1.0e-5
    Step 1.0e-4
  End
End
UseSymmetry No
LoadEngine Graphite2Diamond.results/dftb.rkf

eor

```

11.1.5 Example: Boron nitride optimization under external stress

Download ExternalStress_BN.run

```
#!/bin/sh

# Some (fairly arbitrary) external stress in atomic units:
external_stress='0.001 0.0003 -0.0007 0.0005 0.0004 0.001'

# The external stress applied here will result in pretty large deformations.
# Since the external stress energy term becomes more and more approximate the
# larger the deformation from the initial cell is, after the first optimization,
# one should run a second geometry optimization starting from the converged
# results of the first one.

# It is always good practice to validate the optimization under external stress
# by computing the stress tensor of the final optimized geometry: the stress
# tensor at the optimized geometry should match the external stress applied during
# the optimization, but with reversed signs.

AMS_JOBNAME=first_go $AMSBIN/ams << eor

Task GeometryOptimization

System
  Atoms
    B 0.0 0.0 0.0
    N 0.905 0.905 0.905
  End
  Lattice
    0.0 1.81 1.81
    1.81 0.0 1.81
    1.81 1.81 0.0
  End
  SuperCell 2 2 2
End

GeometryOptimization
  OptimizeLattice Yes
  # The convergence threshold should be smaller than the applied external stress:
  Convergence Gradients=1.0E-4
End

# We want the external stress to break the symmetry:
UseSymmetry No

EngineAddons
  ExternalStress
    StressTensorVoigt $external_stress
  End
End

Engine ForceField
EndEngine
eor

# The second geometry optimization starting from the results of the first one:
AMS_JOBNAME=second_go $AMSBIN/ams << eor

Task GeometryOptimization
```

(continues on next page)

(continued from previous page)

```

LoadSystem
  File first_go.results/ams.rkf
End

GeometryOptimization
  OptimizeLattice Yes
  Convergence Gradients=1.0E-5
End

UseSymmetry No

EngineAddons
  ExternalStress
    StressTensorVoigt $external_stress
  End
End

LoadEngine first_go.results/forcefield.rkf
eor

# In the final calculation we simply compute the stress tensor of the final
# geometry, which should be the opposite of the external stress tensor applied
# during the optimization.

AMS_JOBNAME=sp $AMSBIN/ams << eor

Task SinglePoint

LoadSystem
  File second_go.results/ams.rkf
End

UseSymmetry No

Properties StressTensor=Yes

LoadEngine first_go.results/forcefield.rkf
eor

echo "Applied external stress tensor (order: xx,yy,zz,yz,xz,xy):"
echo $external_stress
echo "Resulting final stress tensor:"
$AMSBIN/amsreport sp.results/forcefield.rkf -r "AMSResults%StressTensor#12.6f##3"

```

11.1.6 Example: Graphene optimization under external stress

Download ExternalStress_graphene.run

```

#!/bin/sh

# Here we perform an optimization under external stress using the
# UpdateReferenceCell option. When using this option the energy during
# the geometry optimization will not necessarily go down, but the final
# stress will match the ExternalStress with much higher accuracy.

```

(continues on next page)

(continued from previous page)

```
AMS_JOBNAME=go $AMSBIN/ams << eor

Task GeometryOptimization

System

  Atoms
    C 0.0          0.0          0.0
    C 1.23647204352 0.7138774661 0.0
  End
  Lattice
    1.23647204352 -2.1416324015 0.0
    1.23647204352  2.1416324015 0.0
  End
End

UseSymmetry no

EngineAddons
  ExternalStress
    StressTensorVoigt 0.01 0.02 0.003
    UpdateReferenceCell Yes
  End
End

GeometryOptimization
  Method SCMGO
  OptimizeLattice Yes
  Convergence Gradients=1.0E-4
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/psc-0-3
  KSpace Quality=Normal
EndEngine
eor

# Here we compute the stress tensor at the optimized geometry to validate
# the optimization under stress

AMS_JOBNAME=sp $AMSBIN/ams << eor

Task SinglePoint

LoadSystem
  File go.results/ams.rkf
End

UseSymmetry no

Properties
  StressTensor Yes
End

LoadEngine go.results/dftb.rkf
```

(continues on next page)

(continued from previous page)

```

eor

echo "Resulting final stress tensor:"
$AMSBIN/amsreport sp.results/dftb.rkf -r "AMSResults%StressTensor#12.6f##2"

```

11.1.7 Example: Constrained optimizations

Download constraints.run

```

#!/bin/sh

# This example demonstrates the setup of all different types of constraints.
# Note that all constraints types can be combined with each other, as long as
# the resulting set of constraints actually makes sense. (It must of course be
# possible to satisfy all of them at the same time. AMS is not able to check
# that and you might get really surprising results if that is not the case ...)

# 1. Angle constraints
# =====

AMS_JOBNAME=angle "$AMSBIN/ams" << EOF

  Task GeometryOptimization

  GeometryOptimization
    Convergence Step=1.0e-3
  End

  System
    Atoms
      O   0.001356   0.000999   0.000000
      H   0.994442  -0.037855   0.000000
      H  -0.298554   0.948531   0.000000
    End
  End

  Constraints
    # Fix the H--O--H angle to 125 degrees.
    Angle  3 1 2 125.0
  End

  Engine DFTB
    Model SCC-DFTB
    ResourcesDir Dresden
    DispersionCorrection Auto
  EndEngine

EOF

# 2. Distance constraints
# =====

AMS_JOBNAME=dist "$AMSBIN/ams" << EOF

```

(continues on next page)

(continued from previous page)

```
Task GeometryOptimization

GeometryOptimization
  Convergence Step=1.0e-3
End

System
  Atoms
    O  0.001356  0.000999  0.000000
    H  0.994442 -0.037855  0.000000
    H -0.298554  0.948531  0.000000
  End
End

Constraints
  # Fix the O--H bond distances to 1.03 Angstrom.
  Distance 1 2 1.03
  Distance 1 3 1.03
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir Dresden
  DispersionCorrection Auto
EndEngine

EOF

# 3. Dihedral angle constraint
# =====

AMS_JOBNAME=dihed "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  Convergence Step=1.0e-3
End

System
  Atoms
    C  -0.004115  -0.000021  0.000023
    C  1.535711   0.000022  0.000008
    H  -0.399693   1.027812  -0.000082
    H  -0.399745  -0.513934   0.890139
    H  -0.399612  -0.513952  -0.890156
    H   1.931188   0.514066   0.890140
    H   1.931432   0.513819  -0.890121
    H   1.931281  -1.027824   0.000244
  End
End

Constraints
  # Fix the dihedral angle H(6)--C(2)--C(1)--H(3) to 20 degrees.
  Dihedral 6 2 1 3 20.00
```

(continues on next page)

(continued from previous page)

```

End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir Dresden
  DispersionCorrection Auto
EndEngine

EOF

# 4a. Fixed atom constraint (Atoms keyword)
# =====

AMS_JOBNAME=atom "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  Convergence Energy=1.0e-6 Gradients=1.0e-4 Step=1.0e-3
  CoordinateType Cartesian
End

System
  Atoms
    C   -0.2460249052   -1.70363153    0.0005128649944
    O   1.152833576    -1.81594932   -0.0004409224206
    C   1.489235475     0.61782051    10.0004771689226
    O   0.5700116914     0.627761615   10.0005491194077
  End
End

Constraints
  # Fix atom 1 and 2 at their initial positions.
  Atom 1
  Atom 2
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF

# 4b. Fixed atom constraint (AtomList keyword)
# =====

AMS_JOBNAME=atomlist "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  Convergence Energy=1.0e-6 Gradients=1.0e-4 Step=1.0e-3
  CoordinateType Cartesian
End

```

(continues on next page)

(continued from previous page)

```
System
  Atoms
    C  -0.2460249052  -1.70363153   0.0005128649944
    O   1.152833576   -1.81594932  -0.0004409224206
    C   1.489235475    0.61782051  10.0004771689226
    O   0.5700116914   0.627761615  10.0005491194077
  End
End

Constraints
  # Fix atom 1 and 2 at their initial positions.
  AtomList 1:2
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF

# 4c. Fixed atom constraint (FixedRegion keyword)
# =====

AMS_JOBNAME=region "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  Convergence Energy=1.0e-6 Gradients=1.0e-4 Step=1.0e-3
  CoordinateType Cartesian
End

System
  Atoms
    C  -0.2460249052  -1.70363153   0.0005128649944  region=fixed
    O   1.152833576   -1.81594932  -0.0004409224206  region=fixed
    C   1.489235475    0.61782051  10.0004771689226
    O   0.5700116914   0.627761615  10.0005491194077
  End
End

Constraints
  # Fix all atoms in region "fixed"
  FixedRegion fixed
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF
```

(continues on next page)

(continued from previous page)

```

# 4d. Fixed atom constraint (overlapping combination)
# =====
AMS_JOBNAME=combination "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  Convergence Energy=1.0e-6 Gradients=1.0e-4 Step=1.0e-3
  CoordinateType Cartesian
End

System
  Atoms
    C   -0.2460249052   -1.70363153    0.0005128649944   region=fixed
    O   1.152833576    -1.81594932   -0.0004409224206   region=fixed
    C   1.489235475     0.61782051    10.0004771689226
    O   0.5700116914    0.627761615   10.0005491194077
  End
End

Constraints
  Atom 1
  AtomList 1 2
  FixedRegion fixed
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF

# 5. Fixed coordinate constraint
# =====
AMS_JOBNAME=coord "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  Convergence Energy=1.0e-6 Gradients=1.0e-4 Step=1.0e-3
  CoordinateType Cartesian
End

System
  Atoms
    C   -0.2460249052   -1.70363153    0.0005128649944
    O   1.152833576    -1.81594932   -0.0004409224206
    C   1.489235475     0.61782051    10.0004771689226
    O   0.5700116914    0.627761615   10.0005491194077
  End
End

Constraints

```

(continues on next page)

(continued from previous page)

```
# Fix the x-coordinate of all atoms.
Coordinate 1 x
Coordinate 2 x
Coordinate 3 x
Coordinate 4 x
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF

# 6. Fixed atom constraint (in periodic system)
# =====

AMS_JOBNAME=pbcatom "$AMSBIN/ams" << EOF

  Task GeometryOptimization

  GeometryOptimization
    Convergence Step=1.0e-3
  End

  System
    Atoms
      C -1.23 -0.710140830 0.0
      C -1.23 -0.710140830 3.8
      C 0.0 0.0 0.4
      C 0.0 -1.42028166 3.355
    End

    Lattice
      1.23 -2.130422493309719 0.0
      1.23 2.130422493309719 0.0
    End
  End

  Constraints
    # Fix atom 1 and 3 at their initial positions.
    Atom 1
    Atom 3
  End

  Engine DFTB
    Model SCC-DFTB
    ResourcesDir DFTB.org/mio-1-1
    KSpace Quality=GammaOnly
  EndEngine

EOF

# 7. Block constraints (with listing the atoms in a block)
# =====
```

(continues on next page)

(continued from previous page)

```

AMS_JOBNAME=block_list "$SAMSBIN/ams" << EOF

Task GeometryOptimization

System
  Atoms
    C  0.5584839616765542   0.5023705181144142  -0.4625483159356394
    C  1.07173137896726    0.2125484528111251  -1.892767990599312
    C  1.699248504588085   -1.006061067555322  -2.191856791501442
    C  2.242484629452111   -1.236470028363516  -3.455616615521399
    C  2.18874580207099    -0.2444337131062739 -4.435483595049287
    C  1.604409798904145    0.9866950282217637 -4.135465239465763
    C  1.061086793296828    1.217355116664161  -2.871773146851866
    H  1.763625603740592   -1.780903563899969  -1.431707209662057
    H  2.716038261390732   -2.190869049673275  -3.672115451399807
    H  2.611833078693977   -0.4241619800888815 -5.420308290235123
    H  1.578029796368043    1.774138556616255  -4.884624561698751
    H  0.6247213391616491    2.187200330357715  -2.64521108544713
    C  1.303528070245188    -0.1416812092038768  0.7303699949711653
    C  0.8164830922475474   -1.314631142230651  1.326337082260565
    C  1.531799364672407    -1.947399963062604  2.342825210379356
    C  2.757684862125068    -1.432061688813837  2.765634667957531
    C  3.271640455523863    -0.2897364031184506  2.150731553729188
    C  2.556535912403799    0.3432056352653093  1.134221563049466
    H  -0.128925843064934   -1.7366201913903    0.9939642396630857
    H  1.133600273086767   -2.849990046242235  2.799740694330775
    H  3.31486005979636    -1.925049398411132  3.557912279830031
    H  4.236604921323707    0.1064455961800578  2.457138367063388
    H  2.976510069814392    1.222131876866508  0.6510413538003352
    C  -0.930165749820548    0.9153412637395284  -0.5420710991631585
    C  -1.791729737216814    0.6892660986048864  0.5418285200469819
    C  -3.111373625199894    1.139542032267652  0.5090625363459357
    C  -3.586568528476239    1.843983986018719  -0.5977864609101087
    C  -2.726152821786783    2.111108432452229  -1.663369105880468
    C  -1.406454626777386    1.660929752085611  -1.63085383469072
    H  -1.428888457076976    0.1571120160719108  1.417905619994904
    H  -3.76723983501283    0.9462006794587581  1.35432032282366
    H  -4.614972346570283    2.194578435055282  -0.6233521468909432
    H  -3.080200905921361    2.678981846821393  -2.520207901691867
    H  -0.7413545301831963    1.891248563160919  -2.459672151335554
    C  1.235557647765805    1.735720249011045  0.1803884343948648
    C  1.377191890012647    1.826646222422494  1.573181692925026
    C  1.905898822116255    2.975086608901246  2.16214311213053
    C  2.280792642899383    4.061906342938987  1.371311861877147
    C  2.105006642447361    3.998471351380415  -0.0115253875199488
    C  1.576317094651283    2.850163227898022  -0.6007264381779673
    H  1.072424817958776    0.9937816064904853  2.202306496283991
    H  2.017471491684088    3.023369029562452  3.242524256706377
    H  2.693031233132915    4.956641734238467  1.830324484771476
    H  2.372569859099136    4.8485771293401    -0.6342066225733602
    H  1.427765851939196    2.820397327218896  -1.677480576376967

  End
End

GeometryOptimization
  Convergence

```

(continues on next page)

(continued from previous page)

```

        Energy 1.0e-6
        Gradients 1.0e-4
        Step 1.0e-4
    End
End

Constraints
# Create blocks from the 4 phenyl groups by specifying the atom indices
# explicitly. (The indices follow the order in the System%Atoms block,
# where we happen to have the atoms belonging to the different phenyl
# groups consecutive.)
BlockAtoms 2 3 4 5 6 7 8 9 10 11 12
BlockAtoms 13 14 15 16 17 18 19 20 21 22 23
BlockAtoms 24 25 26 27 28 29 30 31 32 33 34
BlockAtoms 35 36 37 38 39 40 41 42 43 44 45
End

Engine DFTB
    Model DFTB3
    ResourcesDir DFTB.org/3ob-3-1
    DispersionCorrection D3-BJ
EndEngine

EOF

# 8. Block constraints (with named blocks)
# =====

AMS_JOBNAME=block_names "$AMSBIN/ams" << EOF

Task GeometryOptimization

System
  Atoms
    C 0.5584839616765542 0.5023705181144142 -0.4625483159356394
    C 1.07173137896726 0.2125484528111251 -1.892767990599312
↔region=phenyl1
    C 1.699248504588085 -1.006061067555322 -2.191856791501442
↔region=phenyl1
    C 2.242484629452111 -1.236470028363516 -3.455616615521399
↔region=phenyl1
    C 2.18874580207099 -0.2444337131062739 -4.435483595049287
↔region=phenyl1
    C 1.604409798904145 0.9866950282217637 -4.135465239465763
↔region=phenyl1
    C 1.061086793296828 1.217355116664161 -2.871773146851866
↔region=phenyl1
    H 1.763625603740592 -1.780903563899969 -1.431707209662057
↔region=phenyl1
    H 2.716038261390732 -2.190869049673275 -3.672115451399807
↔region=phenyl1
    H 2.611833078693977 -0.4241619800888815 -5.420308290235123
↔region=phenyl1
    H 1.578029796368043 1.774138556616255 -4.884624561698751
↔region=phenyl1
    H 0.6247213391616491 2.187200330357715 -2.64521108544713
↔region=phenyl1

```

(continues on next page)

(continued from previous page)

C	1.303528070245188	-0.1416812092038768	0.7303699949711653	└
↔region=phenyl2				
C	0.8164830922475474	-1.314631142230651	1.326337082260565	└
↔region=phenyl2				
C	1.531799364672407	-1.947399963062604	2.342825210379356	└
↔region=phenyl2				
C	2.757684862125068	-1.432061688813837	2.765634667957531	└
↔region=phenyl2				
C	3.271640455523863	-0.2897364031184506	2.150731553729188	└
↔region=phenyl2				
C	2.556535912403799	0.3432056352653093	1.134221563049466	└
↔region=phenyl2				
H	-0.128925843064934	-1.7366201913903	0.9939642396630857	└
↔region=phenyl2				
H	1.133600273086767	-2.849990046242235	2.799740694330775	└
↔region=phenyl2				
H	3.31486005979636	-1.925049398411132	3.557912279830031	└
↔region=phenyl2				
H	4.236604921323707	0.1064455961800578	2.457138367063388	└
↔region=phenyl2				
H	2.976510069814392	1.222131876866508	0.6510413538003352	└
↔region=phenyl2				
C	-0.930165749820548	0.9153412637395284	-0.5420710991631585	└
↔region=phenyl3				
C	-1.791729737216814	0.6892660986048864	0.5418285200469819	└
↔region=phenyl3				
C	-3.111373625199894	1.139542032267652	0.5090625363459357	└
↔region=phenyl3				
C	-3.586568528476239	1.843983986018719	-0.5977864609101087	└
↔region=phenyl3				
C	-2.726152821786783	2.111108432452229	-1.663369105880468	└
↔region=phenyl3				
C	-1.406454626777386	1.660929752085611	-1.63085383469072	└
↔region=phenyl3				
H	-1.428888457076976	0.1571120160719108	1.417905619994904	└
↔region=phenyl3				
H	-3.76723983501283	0.9462006794587581	1.35432032282366	└
↔region=phenyl3				
H	-4.614972346570283	2.194578435055282	-0.6233521468909432	└
↔region=phenyl3				
H	-3.080200905921361	2.678981846821393	-2.520207901691867	└
↔region=phenyl3				
H	-0.7413545301831963	1.891248563160919	-2.459672151335554	└
↔region=phenyl3				
C	1.235557647765805	1.735720249011045	0.1803884343948648	└
↔region=phenyl4				
C	1.377191890012647	1.826646222422494	1.573181692925026	└
↔region=phenyl4				
C	1.905898822116255	2.975086608901246	2.16214311213053	└
↔region=phenyl4				
C	2.280792642899383	4.061906342938987	1.371311861877147	└
↔region=phenyl4				
C	2.105006642447361	3.998471351380415	-0.0115253875199488	└
↔region=phenyl4				
C	1.576317094651283	2.850163227898022	-0.6007264381779673	└
↔region=phenyl4				
H	1.072424817958776	0.9937816064904853	2.202306496283991	└
↔region=phenyl4				

(continues on next page)

(continued from previous page)

```

      H   2.017471491684088   3.023369029562452   3.242524256706377
↪region=phenyl4
      H   2.693031233132915   4.956641734238467   1.830324484771476
↪region=phenyl4
      H   2.372569859099136   4.8485771293401    -0.6342066225733602
↪region=phenyl4
      H   1.427765851939196   2.820397327218896  -1.677480576376967
↪region=phenyl4
      End
    End

  GeometryOptimization
    Convergence
      Energy 1.0e-6
      Gradients 1.0e-4
      Step 1.0e-4
    End
  End

  Constraints
    # Use the region from System%Atoms to set up the block constraints.
    Block phenyl1
    Block phenyl2
    Block phenyl3
    Block phenyl4
  End

  Engine DFTB
    Model DFTB3
    ResourcesDir DFTB.org/3ob-3-1
    DispersionCorrection D3-BJ
  EndEngine

EOF

# 9. Frozen strain components
# =====

AMS_JOBNAME=freezestrain "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  OptimizeLattice Yes
  Convergence Step=1.0e-3
End

Constraints
  # Keeps first two lattice vectors orthogonal to the third. Also fixes the
  # length of the third vector, keeping the graphene layer compressed.
  FreezeStrain xz yz zz
End

System
  Atoms
    C 1.332002504889882e-05 -0.0005830055256093706 -8.209389319526933e-06

```

(continues on next page)

(continued from previous page)

```
C -1.22799350000696 -0.7102112812520209 2.281155685325205e-06
C -0.0006872840163290542 -0.0003386565731411325 -1.981477647175959
C 1.2274512359848 0.7092866246929653 -1.981478017299119
C 2.455989750017203 -0.000767672446473915 -5.638209535859324e-06
C 1.227983749989149 -0.7105220051279582 3.556077144406634e-06
C 2.45553905980411 -0.0003697961984884611 -1.981476578954899
C 3.68349483597652 0.7093774139714127 -1.981475303736415
C 4.912014119974971 -0.0004697689000645081 8.202057640607653e-06
C 3.68401303002027 -0.7103327188132248 -6.644074866545941e-06
C 4.911561265976663 -0.0002732185613776612 -1.98147535090646
C -3.685503114025999 0.7094747213946447 -1.98147447813657
C -2.457004890026731 -0.0008782302621621878 8.760751751649826e-06
C -3.684994169978904 -0.7103491590560944 -6.913500704937906e-06
C -2.45740142402999 -0.0002120088132086839 -1.981473170030486
C -1.229200584026242 0.709517932531879 -1.98147439816519
C 1.227980230018157 2.127401471357515 -5.950364005944094e-06
C 9.469984377119545e-06 1.417970232416515 5.120417805695729e-06
C 1.227229005981529 2.127790824745807 -1.981476944534885
C 2.45544009594217 2.837313001498961 -1.981464045820237
C 3.683977240012926 2.127396400995821 -4.237131224100653e-06
C 2.456019429974761 1.41770041892015 8.271514976735398e-06
C 3.683520895940616 2.127826615636785 -1.981463536474189
C 4.911484545967099 2.837408990674362 -1.981472216079415
C 6.140019459971655 2.127636216669431 9.289406940173374e-06
C 4.912011129977858 1.417969521782559 7.256699431696856e-06
C 6.139527915931508 2.12792209328836 -1.981460550680031
C -2.457504644023984 2.837506078460876 -1.981475136785154
C -1.229001220032881 2.127025640069692 1.077705178964691e-05
C -2.457025360024441 1.417788944250494 8.010947395781608e-06
C -1.229428944072945 2.128012192586653 -1.981459091806229
C -0.001217694074323372 2.837543459113209 -1.981458639351295
C 2.455982410005773 4.255441598373883 -1.892083560740779e-06
C 1.228003499971814 3.545886142064043 9.237737681677788e-06
C 2.455221785970465 4.255792992279458 -1.981473318340598
C -1.228386974045185 -3.547700260767117 -1.981468190394571
C 4.911976899993052 4.255411828501257 2.27723146438149e-06
C 3.684014579960917 3.545723396055813 1.280915829951697e-05
C 4.911520375955087 4.255828023455356 -1.981468278811
C 1.227655395958869 -3.547614761418386 -1.98146951906497
C -2.457018900008975 4.255512695928259 2.943041159330732e-06
C 6.140026009993287 3.545891232143294 2.20060806891485e-06
C 7.367526315913146 4.255927240986645 -1.981454533470139
C -6.141340994042006 -3.547511474074143 -1.981469232026619
C -0.001002500050462096 4.255387679251578 1.654017565004685e-05
C -1.228981830007242 3.545851372434187 2.37503105142233e-06
C -0.00142595404759982 4.256013860539822 -1.981467396982039
C -3.685044664049419 -3.547477052980626 -1.981466802486946
C -1.227808819999351 -2.12938224692705 -2.127801149456805e-07
C -2.455832350038186 -2.838708610558109 1.251523005803983e-05
C -1.228620264037983 -2.129205540950233 -1.981470550283798
C -0.0004192140835714842 -1.419477849521901 -1.981455609514733
C 1.228193719957573 -2.129406616582517 1.390527520593389e-05
C 0.0001477699611014405 -2.839255138681037 1.274825905530347e-05
C 1.227684425953123 -2.129163359695275 -1.981467635427455
C 2.455626385910209 -1.419413169537493 -1.981453570671329
C -6.140842350045955 -2.129322170430451 1.506226938282425e-05
C 2.456153479955113 -2.839200210115958 1.47101339575357e-05
```

(continues on next page)

(continued from previous page)

```
C 3.683689305925244 -2.129064842197087 -1.981458500475127
C -4.913374384079035 -1.419316522220884 -1.981457095053834
C -3.684843340052955 -2.129350560151249 1.735547419229382e-05
C -4.912808430047692 -2.839071261955975 1.563048016823986e-05
C -3.685268534086676 -2.128978772839927 -1.98145459141338
C -2.45712732409351 -1.419353221465499 -1.981452351546793
C -2.996192032925579e-05 -0.000699242152149526 3.962939886687566
C -1.228003971875175 -0.7103778453492622 3.962925088122617
C -0.000700355908038296 -0.0003332148789394825 1.981452859744668
C 1.227439704045832 0.7092909964964251 1.981467977918086
C 2.455972288079895 -0.000591484995550912 3.962939812711258
C 1.228003298120663 -0.7104549535647978 3.962926452096982
C 2.455542044092204 -0.0003647244468015716 1.981452779807835
C 3.683482354040657 0.7093820057289575 1.981469673526022
C 4.912000348117807 -0.0004686671456845799 3.96292738718769
C 3.683997778083127 -0.7103249572456309 3.962938753391991
C 4.911548884078128 -0.0002680168299863262 1.981457393146928
C -3.685515265933381 0.7094801331342701 1.981461165904004
C -2.457014661901717 -0.0008761383582568633 3.962933787286889
C -3.685007531928472 -0.7103425373777408 3.962942555784025
C -2.457412705931413 -0.0002065770405354501 1.981460520958165
C -1.229211335932976 0.7095228343414065 1.98146103281522
C 1.227967348110556 2.127386193561453 3.962929764672974
C -4.281883530521391e-06 1.417919455416095 3.96292782690195
C 1.227218624090811 2.127796526544795 1.981453236878594
C 2.455427384033809 2.83731889320433 1.981471917810995
C 3.683966438104317 2.127406352683383 3.962931809362267
C 2.456006178071116 1.417740879775001 3.962942690498066
C 3.683508864080636 2.127831077384122 1.981456571921596
C 4.911472874041424 2.837414422441095 1.981469423240322
C 6.140006188100173 2.127640908364754 3.96293316778528
C 4.912000948062944 1.417975193600739 3.962945368689221
C 6.139515384034993 2.127928054994523 1.981471530520008
C -2.457516635928298 2.837511270211092 1.981459501137365
C -1.229012451945974 2.127104470095207 3.962948291371187
C -2.457033631907664 1.417792836176979 3.962935736248542
C -1.229441855971066 2.128017364296635 1.981473516125039
C -0.001231065982844282 2.83754798082405 1.981477377543141
C 2.455970918062962 4.255446730162459 3.962945363741043
C 1.227990978068837 3.545888883850297 3.962943437699987
C 2.455207564067417 4.255798083939164 1.981460905296909
C -1.228397585936881 -3.54769452897796 1.981462310609218
C 4.911964488060532 4.255420650159205 3.962946159605713
C 3.684002968100828 3.545727757822979 3.962932953064215
C 4.911507194030889 4.25583213519047 1.981472875202606
C 1.22764468406568 -3.547609799614339 1.981461473160161
C -2.457027591902453 4.255510887977889 3.962934029953071
C 6.140011718057854 3.545898453762834 3.962947035832434
C 7.367513404018803 4.25593209270268 1.981476837527609
C -6.141352815949912 -3.547506482310592 1.981466583645511
C -0.001013701916043441 4.255385511194097 3.962938483249906
C -1.228995721948309 3.545866003895579 3.962949056786192
C -0.001438315946372004 4.256019262267359 1.98146542401001
C -3.68505504595736 -3.547472521146422 1.981469025022036
C -1.227817711946263 -2.129397354545356 3.962948386486201
C -2.45584760189824 -2.838693919188594 3.962932647309199
C -1.22863031596877 -2.129200899095368 1.981472763926178
```

(continues on next page)

(continued from previous page)

```

C -0.0004312259786934947 -1.419471387806733 1.981476015973903
C 1.228183428085925 -2.129384525179524 3.962937836939656
C 0.0001342880538246494 -2.839225227582055 3.962948356816577
C 1.227671624025677 -2.129158527960262 1.981474583888431
C 2.455614294015247 -1.419407657803005 1.981478001551132
C -6.140855951953014 -2.129314868796188 3.962950599781267
C 2.456139408049645 -2.839212628030558 3.962949726809414
C 3.683677514018658 -2.12905939044384 1.98147688315868
C -4.913386125957436 -1.419311390469788 1.981469050050141
C -3.68485488196171 -2.129348928295441 3.96295344930467
C -4.91282016191732 -2.839063750265042 3.962938899273355
C -3.685281025994509 -2.12897353111013 1.981481200585353
C -2.457141595966529 -1.419347949824808 1.981472029890663

End
Lattice
  9.825000579999999 0.0 0.0
  4.91182904 8.51302256 0.0
  0.0 0.0 8.0
End
End
Engine DFTB
  Model DFTB
  ResourcesDir DFTB.org/mio-1-1
  KSpace
  Quality GammaOnly
End
EndEngine

EOF

# 10. Equalized strain components
# =====

AMS_JOBNAME=equalstrain "$AMSBIN/ams" << EOF

Task GeometryOptimization

GeometryOptimization
  OptimizeLattice Yes
  Convergence Step=1.0e-3
End

Constraints
  # Keep the cell cubic, but allow the size of the cube to vary.
  FreezeStrain xy xz yz
  EqualStrain xx yy zz
End

System
  Atoms
    C -0.132285 3.230196 3.399625
    H 0.67231 2.571995 3.747816
    H -0.546925 3.782407 4.25108
    H -0.921872 2.627955 2.935193
    H 0.267346 3.938428 2.664409

```

(continues on next page)

(continued from previous page)

```
C 2.647972 3.79511 0.161215
H 2.745753 2.707187 0.254494
H 2.108302 4.189352 1.030219
H 2.093026 4.03506 -0.753279
H 3.644808 4.248843 0.113424
C -3.290954 -3.607704 -3.419879
H -4.099867 -4.000479 -4.046956
H -2.386445 -3.482146 -4.026458
H -3.088346 -4.309774 -2.602634
H -3.589157 -2.638419 -3.003466
C -3.900392 1.971446 -2.092972
H -2.821972 1.97396 -1.895882
H -4.303924 2.977719 -1.930439
H -4.394183 1.265761 -1.414725
H -4.081488 1.668344 -3.130841
C -3.143958 -3.520015 3.393796
H -3.128547 -4.088022 2.456262
H -3.994325 -3.845525 4.004334
H -3.24151 -2.450891 3.171766
H -2.211449 -3.695621 3.94282
C -0.31406 -0.626145 3.522914
H -0.044022 0.228271 4.154493
H -1.353702 -0.912447 3.720437
H 0.346904 -1.471041 3.749005
H -0.205421 -0.349364 2.467723
C 3.411151 -3.454122 0.161835
H 2.877462 -2.569463 0.528433
H 4.211866 -3.141886 -0.518755
H 2.711585 -4.107617 -0.372385
H 3.843691 -3.997522 1.010048
H -3.283653 -0.451758 -4.172013
H -1.922139 0.6502520000000001 -3.802207
H -2.586463 -0.412172 -2.523601
C -2.360175 -0.332772 -3.593334
H -1.648446 -1.117408 -3.875514
C 3.046249 -3.33059 3.76859
H 2.414628 -3.18136 2.88506
H 2.465863 -3.831302 4.55235
H 3.39517 -2.358701 4.136729
H 3.909333 -3.950995 3.500222
C -3.086408 3.73574 0.4638
H -2.559805 3.990117 -0.463465
H -2.394025 3.813016 1.310247
H -3.469203 2.710618 0.397221
H -3.922599 4.429208 0.611196
C 3.736451 0.338903 -0.234383
H 4.139844 -0.659226 -0.441391
H 4.286082 0.789262 0.6004350000000001
H 2.675343 0.256879 0.028602
H 3.844535 0.968696 -1.125179
C -0.953217 3.761489 -3.029722
H -0.738671 2.687271 -2.986546
H -2.017033 3.913231 -3.24677
H -0.349572 4.223228 -3.819817
H -0.707592 4.222226 -2.065757
C 3.438238 3.368005 3.536049
H 3.718968 3.030104 2.531632
```

(continues on next page)

(continued from previous page)

```

H 4.305113 3.831685 4.021198
H 3.102844 2.509703 4.129906
H 2.62603 4.10053 3.461459
C -0.093351 2.447961 0.147782
H 0.412783 2.191741 -0.790311
H -1.100739 2.015519 0.149423
H -0.163522 3.53817 0.239205
H 0.478074 2.046413 0.99281
C -0.067378999999999999 -1.067744 -0.644773
H 0.831493 -1.69444 -0.611303
H -0.920384 -1.630791 -0.248288
H -0.271712 -0.77696 -1.681851
H 0.091087 -0.168785 -0.037648
C -3.13266 0.095347 1.684164
H -2.468956 -0.758832 1.506078
H -3.797646 -0.127993 2.526632
H -3.731911 0.285949 0.786162
H -2.532126 0.982263 1.917783
C -3.650862 -2.700373 -0.074687
H -4.155919 -2.467829 0.87013
H -2.740814 -3.276506 0.129574
H -3.385824 -1.767849 -0.58654
H -4.32089 -3.289309 -0.711913
C 3.803884 3.754796 -3.348637
H 3.946713 2.667857 -3.350306
H 2.769436 3.987861 -3.626912
H 4.48894 4.214494 -4.07059
H 4.010448 4.148971 -2.34674
C 2.868209 0.11231 2.894284
H 2.317604 0.914041 2.388417
H 2.406222 -0.090550000000000001 3.867554
H 3.909847 0.420825 3.041412
H 2.839161 -0.795073999999999999 2.279753
C -0.320765 -3.560008 1.887422
H -0.965068 -2.68183 2.011986
H 0.02585 -3.901247 2.869895
H 0.543227 -3.294315 1.26713
H -0.88707 -4.36264 1.400678
C 2.415398 -1.437717 -2.776235
H 1.964383 -1.676188 -3.746573
H 2.445411 -2.340763 -2.155392
H 1.816728 -0.668767 -2.274091
H 3.43507 -1.065149 -2.928883
C -3.625996 2.934989 3.78523
H -4.070333 2.734452 2.803299
H -3.043299 2.064066 4.107404
H -4.421782 3.131813 4.513121
H -2.968572 3.809626 3.717096
C 1.422335 1.538945 -3.931672
H 0.608488 0.8054 -3.894205
H 2.3282 1.060135 -4.321529
H 1.616409 1.921293 -2.92272
H 1.136242 2.368954 -4.588236
C 0.028875 -3.521123 -2.677443
H 0.240436 -2.624091 -3.271089
H -0.857142 -3.347281 -2.055678
H 0.888225 -3.744967 -2.034598

```

(continues on next page)

(continued from previous page)

```
      H -0.156019 -4.368152 -3.348409
    End
  Lattice
    10.0 0.0 0.0
    0.0 10.0 0.0
    0.0 0.0 10.0
  End
End
Engine DFTB
  Model DFTB
  ResourcesDir DFTB.org/mio-1-1
  KSpace
    Quality GammaOnly
  End
EndEngine
EOF
```

11.2 Transition state search

11.2.1 Example: TS search starting from initial Hessian

Download COChainFreqTS.run

```
#!/bin/sh

# This example demonstrates in the first step how to calculate the Hessian.
# The second run uses the pre-calculated Hessian and performs a transition
# state search along the frequency mode with the smallest frequency.

# First run: Calculate Hessian
# =====

AMS_JOBNAME=hessian $AMSBIN/ams << EOF

  Task SinglePoint

  Properties
    Hessian True
  End

  System
    Atoms
      C 0.0 0.0 0.0
      O 1.5 0.5 0.0
    End
    Lattice
      3.2 0.0 0.0
    End
  End

  Engine Band
```

(continues on next page)

(continued from previous page)

```

    Basis Type=DZP
    KSpace Quality=Good
EndEngine

EOF

# Second run: TS search with initial Hessian
# =====

AMS_JOBNAME=TS $AMSBIN/ams << EOF

Task TransitionStateSearch

System
  Atoms
    C 0.0 0.0 0.0
    O 1.5 0.5 0.0
  End
  Lattice
    3.2 0.0 0.0
  End
End

GeometryOptimization
  Convergence Gradients=1.0e-4
  InitialHessian
    # Load the pre-calculated Hessian as the initial Hessian for the
    # transition state search using the Quasi-Newton based optimizer.
    Type FromFile
    File hessian.results/band.rkf
  End
End

Properties
  # Also calculate normal modes in the end, so we can see if we actually
  # found a transition state.
  NormalModes True
End

Engine Band
  Basis Type=DZP
  KSpace Quality=Good
EndEngine

EOF

```

11.2.2 Example: PES scan and TS search for H2 on graphene

Download PESScan_and_TS_H2_on_Graphene.run

```

#!/bin/sh

# First we do a 2D PES scan varying the z-coordinate of the two hydrogen atoms
# In this example we will keep the graphene slab fixed. From a physical/chemical

```

(continues on next page)

(continued from previous page)

```
# standpoint this is not a good approximation. The graphene slab is
# intentionally not perfectly symmetric.
```

```
AMS_JOBNAME=PESScan $AMSBIN/ams << EOF
```

```
Task PESScan
```

```
System
```

```
Atoms
```

H	0.0	1.53633037	1.1
H	0.0	-0.11341359	1.1
C	0.001	1.42028166	0.0
C	1.230	2.13042249	0.0
C	1.230	-0.71014083	0.0
C	2.460	0.00000000	0.0
C	2.460	1.42028167	0.0
C	0.000	0.00000000	0.0

```
End
```

```
Lattice
```

3.69	-2.13042249	0.0
0.00	4.26084499	0.0

```
End
```

```
End
```

```
PESScan
```

```
ScanCoordinate
```

```
nPoints 10
```

```
Coordinate 1 Z 1.1 2.0
```

```
End
```

```
ScanCoordinate
```

```
nPoints 10
```

```
Coordinate 2 Z 1.1 2.0
```

```
End
```

```
End
```

```
GeometryOptimization
```

```
Convergence Step=1.0e-3
```

```
End
```

```
Constraints
```

```
# Fix the entire graphene slab.
```

```
Atom 3
```

```
Atom 4
```

```
Atom 5
```

```
Atom 6
```

```
Atom 7
```

```
Atom 8
```

```
End
```

```
Engine DFTB
```

```
Model DFTB
```

```
ResourcesDir DFTB.org/3ob-3-1
```

```
DispersionCorrection D3-BJ
```

```
KSpace
```

```
Type Symmetric
```

```
Symmetric KInteg=3
```

```
End
```

(continues on next page)

(continued from previous page)

```

EndEngine

EOF

# A human looks at the PES scan and picks a reasonable starting point for the
# TS search. (Normally you would do that in AMSMovie by looking at the PES and
# then exporting the geometry into an xyz file.)

#
#      _
#      ___))      [ / | \
#      ) //o      | | ]
#      _ ( >      | | ]
#      (O) \__<   | | ]
#      [ / ] / \ ) [ ___ / _
#      [ \ ] | ( \  _ / ___ \ ___
#      [ / ] | \ \ _ ___ /
#      [ \ ] | \ ___ E / % % / | ___
#      [ / ] | =====_ ( _____ )

cat << EOF > initial_geometry_for_TS.xyz
8

H    0.4145668856457391    1.72927656037925    1.100000023839768    region=H2
H   -0.05533871972549955   -0.06805093626643093   1.500000013242627    region=H2
C    0.001                1.42028166             0.0
C    1.230                2.13042249            0.0
C    1.230                -0.71014083           0.0
C    2.460                0.00000000            0.0
C    2.460                1.42028167            0.0
C    0.000                0.00000000            0.0
VEC1 3.69 -2.13042249 0.0
VEC2 0.0  4.26084499 0.0
EOF

# Compute the partial initial Hessian to be used in the transition state
# search. (The Hessian will be computed only for the hydrogen atoms.)

AMS_JOBNAME=Hessian $AMSBIN/ams << EOF

Task SinglePoint

System
# Load the geometry we just saved.
GeometryFile initial_geometry_for_TS.xyz
End

Properties
# Calculate the Hessian (implied when calculating normal modes) ...
NormalModes True
# ... but only the part related to the hydrogen atoms.
SelectedRegionForHessian H2
End

Engine DFTB
Model DFTB

```

(continues on next page)

(continued from previous page)

```
ResourcesDir DFTB.org/3ob-3-1
DispersionCorrection D3-BJ
KSpace
  Type Symmetric
  Symmetric KInteg=3
End
EndEngine

EOF

echo "Extract the frequencies from the kf file using amsreport:"
$AMSBIN/amsreport Hessian.results/dftb.rkf -r "Vibrations%Frequencies[cm-1]##1"

# Do a transition state search using the initial Hessian just computed (the
# Graphene slab is constrained). Also compute the final Hessian for the
# hydrogen atoms to validate the TS.

AMS_JOBNAME=TS $AMSBIN/ams << EOF

  Task TransitionStateSearch

  System
    # Load the geometry we just saved.
    GeometryFile initial_geometry_for_TS.xyz
  End

  GeometryOptimization
    Quasi-Newton
      Step TrustRadius=0.05
    End
    Convergence Gradients=1.0e-4
    InitialHessian
      # Load previously calculated Hessian as initial Hessian for a
      # transition state search with the Quasi-Newton optimizer.
      Type FromFile
      File Hessian.results/dftb.rkf
    End
  End

  TransitionStateSearch
    # Follow the mode with the smallest frequency.
    ModeToFollow 1
    # (This is also the default, we wouldn't need to specify this.)
  End

  Constraints
    # Fix the entire graphene slab.
    Atom 3
    Atom 4
    Atom 5
    Atom 6
    Atom 7
    Atom 8
  End

  Properties
```

(continues on next page)

(continued from previous page)

```
NormalModes Yes
SelectedRegionForHessian H2
End

Engine DFTB
Model DFTB
ResourcesDir DFTB.org/3ob-3-1
DispersionCorrection D3-BJ
KSpace
Type Symmetric
Symmetric KInteg=3
End
EndEngine

EOF

echo "Extract energy from the rkf file using amsreport:"
$AMSBIN/amsreport TS.results/dftb.rkf -r "AMSResults%Energy"
```

11.3 Nudged Elastic Band (NEB)

11.3.1 Nudged Elastic Band (NEB) Examples

Here are a few examples showing how the NEB method can be used to obtain the the path and transition state of a reaction.

See also:

Nudged Elastic Band (NEB) Documentation (page 63)

HCN isomerization reaction with NEB

Download `NEB_HCN.run`

H2 dissociation on graphene

Download `NEB_H2_on_graphene.run`

Running multiple NEB calculations using PLAMS

Download `NEB.plms`

This example should be executed using `PLAMS`.

See also:

`PLAMS documentation and tutorial`

11.4 Intrinsic reaction coordinate (IRC)

11.4.1 Example: IRC for HCN

Download IRC_HCN.run

```
#!/bin/sh

# == IRC scan of the reaction path ==

# The IRC calculation is split in two steps to illustrate the Restart feature.

# In the first calculation only a few points are computed along the so-called
# 'forward' path. The definition of which is 'forward' and which is
# 'backward' depends on the sign of the largest component of the normal mode
# corresponding to the reaction coordinate.

# The RKF file from this partial IRC scan serves as restart file
# for the next calculations that will continue the IRC scan.

# The 'MaxPoints' key in the IRC block is used to limit the number of IRC
# points to compute.

AMS_JOBNAME=irc1 $AMSBIN/ams << eor

Task IRC
System
  Atoms
    C      0.000000000000      0.000000000000      0.000000000000
    N      0.000000000000      0.000000000000     -1.182644220000
    H     -1.103250760411      0.000000000000     -0.322462130000
  End
End

IRC
  MaxPoints 5
  Direction Forward
  CoordinateType Cartesian
  InitialHessian
    Type Calculate
  End
End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-3-1
EndEngine

eor

# In the second IRC run, the IRC scan is finished. We start with the RKF file
# from the previous run and omit the MaxPoints from the settings, which means
# that the default 100 will be used. Note that the 100 also includes any points
# computed in the previous calculation. The program starts on
# the forward path, continuing where the first calculation had stopped,
```

(continues on next page)

(continued from previous page)

```

# and completes it. Since we set the Direction to Both
# then AMS proceeds to the backward path. After both paths are finished a summary
# of the path characteristics is printed at the end of the output file.

AMS_JOBNAME=irc2 $AMSBIN/ams << eor

Task IRC
System
  Atoms
    C      0.000000000000      0.000000000000      0.000000000000
    N      0.000000000000      0.000000000000     -1.182644220000
    H     -1.103250760411      0.000000000000     -0.322462130000
  End
End

IRC
  Restart
    File irc1.results/ams.rkf
  End
  ! Change options from the ones found in the restart file
  ! (MaxIRCPoints and MaxPoints will be reset to defaults automatically)
  Direction Both
End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-3-1
EndEngine

eor

```

11.4.2 Example: TS and IRC for Claisen reaction

Download `TS_and_IRC_Claisen.run`

```

#!/bin/sh

# Transition State Search (TS search) followed by Intrinsic Reaction
# Coordinates (IRC) for two similar Claisen rearrangement reactions.

# =====
# Claisen rearrangement from C=CCC1C=CC=CC1=O to C=CCOc1cccc1
# =====

AMS_JOBNAME=TS_molecule $AMSBIN/ams << eor

Task TransitionStateSearch

System
  Atoms
    C  -1.6622561642524  -1.4933421191817   0.6484353677288
    C  -2.6070283916282  -1.7718977641902  -0.3933564306530
    C  -2.7546368861548  -3.0534770331072  -0.8757259422474

```

(continues on next page)

(continued from previous page)

```
C -1.9443405437492 -4.1131780924428 -0.3870796280948
C -1.0139402388630 -3.8827189276564 0.5979937257975
C -0.7543606665660 -2.5518266272325 1.0788971265869
H -3.2590342954832 -0.9734836183880 -0.7410179986476
H -3.5074724102535 -3.2690301027846 -1.6324229738243
H -2.1044351565280 -5.1220091436618 -0.7615296311573
H -0.4323033724363 -4.7107876462166 1.0021729248699
H -0.3533176373037 -2.4841001370767 2.0927436180830
O -1.5058234397159 -0.2844936546832 1.1029538316409
C 0.1375150486634 0.3928947854321 0.4626789880484
C 1.0578648498087 -0.6180364119671 0.7737587143345
C 0.8861173890663 -1.8991002496105 0.2125497351161
H 0.5725481135101 -1.9591100578644 -0.8292382400922
H -0.2171264706145 0.4859594211539 -0.5641152540806
H 0.1902646842718 1.3359879065177 1.0025369222419
H 1.5779002347488 -0.5540482019233 1.7307360489935
H 1.6031168776346 -2.6724262749099 0.4842127285858
End
End
Properties NormalModes=Yes
GeometryOptimization
  InitialHessian Type=Calculate
End
Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor

AMS_JOBNAME=IRC_molecule $AMSBIN/ams << eor

Task IRC

IRC
  MaxIterations 1000
  InitialHessian
    Type FromFile
    File TS_molecule.results
  End
end

LoadSystem
  File TS_molecule.results/ams.rkf
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor

# =====
```

(continues on next page)

(continued from previous page)

```
# Claisen rearrangement for a (periodic) polymer containing the same aromatic
# ring of the previous calculation (from C=CCClC=CC=CCl=O to C=CCOc1cccc1)
# =====
```

```
AMS_JOBNAME=TS_polymer $AMSBIN/ams << eor
```

```
Task TransitionStateSearch
```

```
System
```

```
  Atoms
```

C	9.4367476128766	1.6156795441351	0.8542644025030
C	8.6813349262903	0.7302865575002	0.0170374963868
C	9.3238583586638	-0.1438417574104	-0.8314773024306
C	10.741095802442	-0.1538813927018	-0.9297572682512
C	11.499909556383	0.6929842950328	-0.1576524435776
C	10.884234857485	1.6861365363275	0.6817028097694
C	7.1928255650616	0.6847044785103	0.0090333093191
H	8.7472521390038	-0.8479982636647	-1.4293934603813
H	11.224348014063	-0.8771406786637	-1.5831686891584
C	12.930798908912	0.8113031163622	-0.1971191730819
H	11.474366091679	2.0120169601893	1.5412690799783
O	8.8401614254340	2.4539245100299	1.6503819438045
C	9.0152184164720	4.1939147755043	0.9354292967913
C	10.386800460742	4.3070474573655	0.6675488559614
C	10.983233806221	3.4166811386158	-0.2473786535066
H	10.425317882010	3.1661173633779	-1.1491528320010
H	8.3301833043316	4.0024301166114	0.1091130317749
H	8.5863952766270	4.7863410214548	1.7409077918798
H	11.023307058306	4.7274400682699	1.4475913648158
H	12.053474297431	3.5124067242738	-0.4244212846926
C	13.897191373368	-0.0308805104165	-0.7206407708622
C	15.267346128343	0.2274543686942	-0.4348415320624
C	16.387786908031	-0.4341710477973	-0.8962646879782
C	6.2048264493065	-0.0924013581064	-0.6266734325417
C	4.8694123304551	0.1638639419185	-0.3442585289372
C	3.7074226634814	-0.4584344973285	-0.8689206409329
C	2.4653410147781	0.0102007864139	-0.4848760767909
H	16.298210512111	-1.2787892169088	-1.5797017846452
H	3.7984002771976	-1.2806821291979	-1.5787594353897
H	6.8240750625775	1.4435007199212	0.7047818340481
H	13.314194383001	1.6434188966421	0.3984353613768
H	13.626424975736	-0.8770823766787	-1.3528414377622
H	15.451469431625	1.0743039058568	0.2322503424471
H	2.4641684561885	0.8477050600186	0.2182734314177
H	6.4819803798672	-0.8741436365939	-1.3339335062636
H	4.6807561767470	0.9698264886906	0.3703766590249

```
End
```

```
Lattice
```

```
  15.210 0.0 0.0
```

```
End
```

```
End
```

```
Properties
```

```
  NormalModes Yes
```

```
End
```

```
GeometryOptimization
```

(continues on next page)

(continued from previous page)

```
Method Quasi-Newton
InitialHessian Type=Calculate
End

Engine DFTB
Model SCC-DFTB
ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor

AMS_JOBNAME=IRC_polymer $AMSBIN/ams << eor

Task IRC

IRC
MaxIterations 1000
InitialHessian
Type FromFile
File TS_polymer.results
End
Direction Forward
end

LoadSystem
File TS_polymer.results/ams.rkf
End

Engine DFTB
Model SCC-DFTB
ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor
```

11.5 PES scan

11.5.1 Example: Linear transit

Download `LinearTransit.run`

```
#!/bin/sh

echo "====="
echo "HCN isomerization"
echo "====="
echo

AMS_JOBNAME=HCN_isomerization $AMSBIN/ams << EOF

Task PESScan
# (Linear transit is just a PES scan with 1 scan coordinate.)
```

(continues on next page)

(continued from previous page)

```

System
  Atoms
    C      0.00000000    0.00000000    1.04219000
    H      0.00000000    0.00000000   -0.03324000
    N      0.00000000    0.00000000    2.20064000
  End
End

PESScan
  ScanCoordinate
    nPoints 25
    Angle 2 1 3 180.0 0.0
  End
End

Engine DFTB
  Model DFTB0
  ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF

echo
echo "======"
echo "Water angle transit"
echo "======"
echo

AMS_JOBNAME=water_angle $AMSBIN/ams << EOF

Task PESScan

System
  Atoms
    O      0.00000000    0.00000000    0.59372000
    H      0.00000000    0.76544000   -0.00836000
    H      0.00000000   -0.76544000   -0.00836000
  End
End

PESScan
  ScanCoordinate
    nPoints 25
    Angle 2 1 3 80.0 180.0
  End
End

GeometryOptimization
  ! Delocalized coordinates currently have a problem with linear systems.
  ! So we will use cartesian coordinates here.
  CoordinateType Cartesian
End

Engine DFTB
  Model DFTB0

```

(continues on next page)

(continued from previous page)

```

ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF

echo
echo "====="
echo "Hydrocarbon reaction"
echo "====="
echo

AMS_JOBNAME=hydcarb $AMSBIN/ams << EOF

Task PESScan

System
  Atoms
    C      0.14667300   -0.21503500    0.40053800
    C      1.45297400   -0.07836900    0.12424400
    C      2.23119700    1.15868100    0.12912100
    C      1.78331500    2.39701500    0.38779700
    H     -0.48348000    0.63110600    0.67664100
    H     -0.33261900   -1.19332100    0.35411600
    H      2.01546300   -0.97840100   -0.14506700
    H      3.29046200    1.03872500   -0.12139700
    H      2.45728900    3.25301000    0.35150400
    H      0.74193400    2.60120700    0.64028800
    C     -0.75086900    1.37782400   -2.43303700
    C     -0.05392100    2.51281000   -2.41769100
    H     -1.78964800    1.33942600   -2.09651100
    H     -0.30849400    0.43896500   -2.76734700
    H     -0.49177100    3.45043100   -2.06789100
    H      0.98633900    2.54913500   -2.74329400
  End
End

PESScan
  ScanCoordinate
    nPoints 25
    Distance 1 11   3.36  1.538
    Distance 4 12   3.36  1.538
  End
End

Engine DFTB
Model DFTB0
ResourcesDir DFTB.org/mio-1-1
EndEngine

EOF

echo
echo "====="

```

(continues on next page)

(continued from previous page)

```

echo "Retinal trans -> 11-cis isomerization"
echo "===== "
echo

AMS_JOBNAME=retinal_transcis $AMSBIN/ams << EOF

Task PESScan

System
  Atoms
    H      -2.10968473   -1.58238733    0.78224517
    C      -2.10306857   -0.54058322    0.46363503
    C      -0.89436995    0.04807217    0.25528247
    H      -0.85555481    1.05432693   -0.15803658
    C      0.38987539   -0.58661182    0.49038464
    C      1.53213446    0.09657801    0.14394773
    H      1.40518949    1.08783970   -0.29205231
    H      3.05232192   -1.34477492    0.72115301
    C      2.88311454   -0.36358433    0.28105432
    C      3.96024700    0.37378345   -0.12385974
    H      3.77965758    1.35231793   -0.56821856
    C      5.34627719   -0.04025647   -0.02249097
    C      6.32191717    0.80135945   -0.49190463
    H      6.00090638    1.74979100   -0.92101391
    C     -4.46825064   -0.90426552   -0.39585925
    C     -5.87277429   -0.25303564   -0.45007491
    C     -3.41139545    0.06493448    0.19516310
    C     -3.67932839    1.38221399    0.41656971
    C     -5.81598497    1.19032366   -0.92660753
    C     -5.00049358    2.01922634    0.05561242
    C     -4.58391145   -2.18782901    0.46346394
    C     -4.01729542   -1.30039402   -1.82272212
    C     -2.72429960    2.32303313    1.10290124
    C      0.40919453   -1.96244629    1.09501374
    C      5.64155973   -1.38034133    0.59419110
    C      7.76996060    0.56699126   -0.48750226
    O      8.57693167    1.36615612   -0.92976322
    H     -6.51997817   -0.84904979   -1.10100203
    H     -6.32039371   -0.28079023    0.54871092
    H     -5.36159995    1.23817633   -1.92112092
    H     -6.82595442    1.60207678   -1.01946858
    H     -5.58216571    2.18390764    0.97424181
    H     -4.81292271    3.01993001   -0.35246294
    H     -4.74166770   -1.94289144    1.51126095
    H     -5.43008715   -2.78247632    0.12572479
    H     -3.69644845   -2.81116549    0.38705593
    H     -3.02900804   -1.75403268   -1.79820003
    H     -4.71056940   -2.01489741   -2.26202914
    H     -3.97070839   -0.42860260   -2.47090348
    H     -2.16469005    2.92261100    0.38111736
    H     -3.27791517    3.02297911    1.72885233
    H     -2.00470188    1.79865198    1.72726573
    H     -0.13689001   -1.97717074    2.03825359
    H     -0.07664772   -2.68134154    0.43362393
    H      1.41837401   -2.31391556    1.28591185
    H      5.15278730   -2.17622743    0.03222328
    H      6.70436647   -1.59729505    0.62729622

```

(continues on next page)

(continued from previous page)

```

      H      5.25700064      -1.42489613      1.61313095
      H      8.12614442      -0.41441814      -0.04549414
    End
  End

  PESScan
    ScanCoordinate
      nPoints 25
      Dihedral  6 9 10 12  180  0
      Dihedral  8 9 10 11  180  0
    End
  End

  Engine DFTB
    Model DFTB0
    ResourcesDir DFTB.org/mio-1-1
  EndEngine

EOF
```

11.5.2 Example: 2D PES scan

Download PESScan.run

```

#!/bin/sh

echo "====="
echo "Ethane torsion"
echo "====="
echo

AMS_JOBNAME=ethane_torsion $AMSBIN/ams << EOF

  Task PESScan

  System
    Atoms
      C  0.0      0.0      0.76576
      C  0.0      0.0     -0.76576
      H -0.88668938  0.51193036  1.16677
      H  0.88668938  0.51193036  1.16677
      H  0.0      -1.02386071  1.16677
      H  0.0      1.02386071 -1.16677
      H -0.88668938 -0.51193036 -1.16677
      H  0.88668938 -0.51193036 -1.16677
    End
  End

  PESScan
    # First scan coordinate: C--C bond distance
    ScanCoordinate
      nPoints 5
      Distance 1 2 1.3 1.7
    End
```

(continues on next page)

(continued from previous page)

```

# Second scan coordinate: One of the H--C--C--H dihedral angles (others will
↳follow naturally)
  ScanCoordinate
    nPoints 21
    Dihedral 3 1 2 6 60.0 0.0
  End
End

GeometryOptimization
  Convergence Step=1.0e-3
End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-3-1
  DispersionCorrection D3-BJ
EndEngine

EOF

echo "======"
echo "Ethene torsion"
echo "======"
echo

AMS_JOBNAME=ethene_torsion $AMSBIN/ams << EOF

Task PESScan

System
  Atoms
    C 0.0 0.0 0.66687
    C 0.0 0.0 -0.66687
    H 0.0 0.92974 -1.23912
    H 0.0 0.92974 1.23912
    H 0.0 -0.92974 1.23912
    H 0.0 -0.92974 -1.23912
  End
End

PESScan
  # First scan coordinate: C--C bond distance
  ScanCoordinate
    nPoints 5
    Distance 1 2 1.1 1.8
  End
  # Second scan coordinate: Two of the H--C--C--H dihedrals
  ScanCoordinate
    nPoints 21
    Dihedral 4 1 2 3 0.0 60.0
    Dihedral 5 1 2 6 0.0 60.0
  End
End

GeometryOptimization
  Convergence Step=1.0e-3

```

(continues on next page)

(continued from previous page)

```

End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-3-1
  DispersionCorrection D3-BJ
EndEngine

EOF

# Below are more technical examples, demonstrating the PES scan gap filling.
# The QUASINANO2015 parameter set shows some discontinuities in the PES,
# which leads to problems with convergence. The first job leaves the
# non-converged steps as is while the second job instructs AMS to
# attempt a second optimization for non-converged point starting from
# a different initial geometry.

echo "======"
echo "Ethane gap filling test (1/2)"
echo "======"
echo

AMS_JOBNAME=ethane_nofillgaps $AMSBIN/ams << EOF

Task PESScan

System
  Atoms
    C -2.333834610464788 -2.268837915270455 -0.2417723425321957
    C -0.8081611038872945 -2.334371994724881 -0.04271045326758349
    H -0.2505615773096904 -1.473443563856088 -0.38077110593546
    H -0.3249814761083244 -3.235478579439597 -0.3904810245975267
    H -0.583247370537557 -2.349691649662279 1.013499336841977
    H -2.817014238243758 -1.367731330555738 0.1059982287977475
    H -2.891434137042391 -3.129766346139247 0.09628831013568076
    H -2.558748343814525 -2.253518260333056 -1.297982132641757
  End
End

GeometryOptimization
  CoordinateType Cartesian
  Convergence Step=1.0e-3
End

PESScan
  FillUnconvergedGaps False
  CalcPropertiesAtPESPoints True
  ScanCoordinate
    nPoints 10
    Distance 1 2 1.4 1.7
  End
  ScanCoordinate
    nPoints 10
    Distance 7 1 1.0 1.2
    Dihedral 7 1 2 3 60.0 180.0

```

(continues on next page)

(continued from previous page)

```

End
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir QUASINANO2015
EndEngine

EOF

echo "======"
echo "Ethane gap filling test (2/2)"
echo "======"
echo

AMS_JOBNAME=ethane_fillgaps $AMSBIN/ams << EOF

  Task PESScan

  System
    Atoms
      C -2.333834610464788 -2.268837915270455 -0.2417723425321957
      C -0.8081611038872945 -2.334371994724881 -0.04271045326758349
      H -0.2505615773096904 -1.473443563856088 -0.38077110593546
      H -0.3249814761083244 -3.235478579439597 -0.3904810245975267
      H -0.583247370537557 -2.349691649662279 1.013499336841977
      H -2.817014238243758 -1.367731330555738 0.1059982287977475
      H -2.891434137042391 -3.129766346139247 0.09628831013568076
      H -2.558748343814525 -2.253518260333056 -1.297982132641757
    End
  End

  GeometryOptimization
    CoordinateType Cartesian
    Convergence Step=1.0e-3
  End

  PESScan
    FillUnconvergedGaps True
    CalcPropertiesAtPESPoints True
    ScanCoordinate
      nPoints 10
      Distance 1 2 1.4 1.7
    End
    ScanCoordinate
      nPoints 10
      Distance 7 1 1.0 1.2
      Dihedral 7 1 2 3 60.0 180.0
    End
  End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir QUASINANO2015
EndEngine

```

(continues on next page)

(continued from previous page)

EOF

11.6 PES Exploration

11.6.1 Example: Basin Hopping for Ar 13 cluster

Download BasinHopping_Ar13.run

```
#!/bin/sh

# This example shows how the Basin Hopping procedure can be used to find the
# global minimum (and several local minima) of an Argon-13 cluster.
# The LennardJones engine is used for computing energy and gradients.

"$AMSBIN/ams" << EOF

Task PESExploration

System
  Atoms
    Ar      3.19796865788947      1.15934987330252      -1.33239534934505
    Ar      0.77363272043336      0.86005279907579      4.44840472306942
    Ar      6.16557782618638      1.42715657126078      1.00292335487038
    Ar      1.31188973509015      -2.14075518617674     -0.96878076867831
    Ar     -2.41087887202771     -1.38658115989663     -0.47288216008194
    Ar     -0.21272211859964     -2.33529094743978     2.53956949757235
    Ar      0.68146295080450      4.04228262113743     -1.08313269600570
    Ar     -0.30271346683735      0.77172238432013     -2.86122291101125
    Ar     -2.82554186881390      0.46651994777473     2.87593685953916
    Ar      3.25381032839757     -0.76862912920592     2.01893019694943
    Ar     -2.88251373018356      2.43351471156095     -0.54320785580135
    Ar      0.18478571590318      0.93399697546301     0.81886912458718
    Ar      2.90785012175754      3.08435653882379     2.01057298433565
  End
End

PESExploration
  Job BasinHopping
  RandomSeed 10
  Temperature 50.0
  NumExpeditions 5
  NumExplorers 5
  FiniteDifference 1e-7

  BasinHopping
    Displacement 5.0
    Steps 50
    DisplacementDistribution gaussian
    WriteUnique T
  End

  Optimizer
    ConvergedForce 1e-8
```

(continues on next page)

(continued from previous page)

```

        MaxIterations 1000
    End

    StructureComparison
        DistanceDifference 0.5
        NeighborCutoff 7.6
    End
End

#Engine ForceField
#EndEngine
Engine LennardJones
    Rmin 3.81749342630018000 # sigma=3.401*angs, rmin=2**(1.0/6.0)*3.401*angs
    Eps 0.000369915709507261 # eps=116.81*K
    Cutoff 15.00 # 15*angs
EndEngine

EOF

```

11.6.2 Example: PES Exploration, Process Search for alanine with PLAMS

Download ProcessSearch_alanine.plms

```

from tools.plams_test_utils import validate_energy_landscape

config.log.stdout = 0

alanine = from_smiles("CC(C(=O)O)N")

# Let's run a process search job on Alanine:

engine_sett = Settings()
engine_sett.input.ForceField = Settings()

sett = Settings()
sett.input.ams.Task = "PESExploration"
sett.input.ams.PESExploration.Job = "ProcessSearch"
sett.input.ams.PESExploration.NumExpeditions = 32
sett.input.ams.PESExploration.NumExplorers = 16

job = AMSJob(name='alanine_process_search', molecule=alanine, settings=sett + engine_
→sett)
results = job.run()
energy_landscape = results.get_energy_landscape()
print(energy_landscape)

# Let's run single point calculations on the found states to verify the
# correctness of the results:

validate_energy_landscape(energy_landscape, engine_sett)

```

11.6.3 Example: PES Exploration, Binding Sites for O on Pt 111

Download BindingSites_O+Pt111.run

```

#!/bin/sh

export OMP_NUM_THREADS=1 # Reduce ReaxFF engine noise

AMS_JOBNAME=process_search "$AMSBIN/ams" << EOF

Task PESExploration

System
  Atoms
    O      5.54003937    1.60484089    8.92740946    region=adsorbate
    Pt     8.31521714    4.80239136    7.78926430    region=surface
    Pt     5.54298786    0.00160612    7.79128800    region=surface
    Pt     6.92910250    2.40199874    7.79027615    region=surface
    Pt     11.08707571   4.80244586    7.78854885    region=surface
    Pt     8.31484643    0.00166062    7.79057254    region=surface
    Pt     9.70096096    2.40178486    7.79068672    region=surface
    Pt     5.54335868    4.80260524    7.78885374    region=surface
    Pt     2.77112940    0.00182001    7.79087743    region=surface
    Pt     4.15724404    2.40221263    7.78986559    region=surface
    Pt     9.70154331    5.60130339    5.52585419    region=surface
    Pt     6.92931403    0.80051815    5.52787789    region=surface
    Pt     8.31542867    3.20091077    5.52686604    region=surface
    Pt     4.15782639    5.60173116    5.52503306    region=surface
    Pt     1.38559711    0.80094593    5.52705675    region=surface
    Pt     2.77171175    3.20133854    5.52604491    region=surface
    Pt     6.92968485    5.60151727    5.52544363    region=surface
    Pt     4.15745557    0.80073204    5.52746732    region=surface
    Pt     5.54357021    3.20112466    5.52645547    region=surface
    Pt     6.92989639    4.00003668    3.26303536    region=surface
    Pt     8.31601103    6.40042930    3.26202352    region=surface
    Pt     5.54378174    1.59964406    3.26404721    region=surface
    Pt     9.70175484    3.99982280    3.26344593    region=surface
    Pt     11.08786948    6.40021541    3.26243408    region=surface
    Pt     8.31564020    1.59943018    3.26445778    region=surface
    Pt     4.15803792    4.00025057    3.26262480    region=surface
    Pt     5.54415256    6.40064318    3.26161295    region=surface
    Pt     2.77192328    1.59985795    3.26363664    region=surface
    Pt     1.38639089    2.39871548    1.00094199    region=surface
    Pt     5.54436409    4.79916260    0.99921469    region=surface
    Pt     6.93024583    7.19969029    1.00043291    region=surface
    Pt     4.15824945    2.39876998    1.00022653    region=surface
    Pt     8.31622256    4.79894871    0.99962525    region=surface
    Pt     9.70210429    7.19947641    1.00084348    region=surface
    Pt     6.93010792    2.39855609    1.00063710    region=surface
    Pt     2.77250564    4.79937648    0.99880412    region=surface
    Pt     4.15838737    7.19990418    1.00002234    region=surface
  End
  Lattice
    8.31557575    0.00000000    0.00000000
    4.15778787    7.20149984    0.00000000
  End
End

Constraints
  FixedRegion surface
End

```

(continues on next page)

(continued from previous page)

```

PESExploration
  Job ProcessSearch

  RandomSeed 100
  NumExpeditions 50
  NumExplorers 5
  DynamicSeedStates T

  Optimizer
    ConvergedForce 0.0001
  End

  SaddleSearch
    MaxEnergy 2.0
  End

  StructureComparison
    DistanceDifference 0.1
    NeighborCutoff 10.0
    EnergyDifference 0.01
  End
End

Engine ReaxFF
  ForceField CHONSFpTc1Ni.ff
  Charges Solver=Direct # Reduce ReaxFF engine noise
EndEngine

EOF

AMS_JOBNAME=binding_sites "$AMSBIN/ams" << EOF

Task PESExploration

System
  Atoms
    O      5.54003937    1.60484089    8.92740946    region=adsorbate
    Pt     8.31521714    4.80239136    7.78926430    region=surface
    Pt     5.54298786    0.00160612    7.79128800    region=surface
    Pt     6.92910250    2.40199874    7.79027615    region=surface
    Pt    11.08707571    4.80244586    7.78854885    region=surface
    Pt     8.31484643    0.00166062    7.79057254    region=surface
    Pt     9.70096096    2.40178486    7.79068672    region=surface
    Pt     5.54335868    4.80260524    7.78885374    region=surface
    Pt     2.77112940    0.00182001    7.79087743    region=surface
    Pt     4.15724404    2.40221263    7.78986559    region=surface
    Pt     9.70154331    5.60130339    5.52585419    region=surface
    Pt     6.92931403    0.80051815    5.52787789    region=surface
    Pt     8.31542867    3.20091077    5.52686604    region=surface
    Pt     4.15782639    5.60173116    5.52503306    region=surface
    Pt     1.38559711    0.80094593    5.52705675    region=surface
    Pt     2.77171175    3.20133854    5.52604491    region=surface
    Pt     6.92968485    5.60151727    5.52544363    region=surface
    Pt     4.15745557    0.80073204    5.52746732    region=surface
    Pt     5.54357021    3.20112466    5.52645547    region=surface

```

(continues on next page)

(continued from previous page)

```
Pt      6.92989639      4.00003668      3.26303536      region=surface
Pt      8.31601103      6.40042930      3.26202352      region=surface
Pt      5.54378174      1.59964406      3.26404721      region=surface
Pt      9.70175484      3.99982280      3.26344593      region=surface
Pt     11.08786948      6.40021541      3.26243408      region=surface
Pt      8.31564020      1.59943018      3.26445778      region=surface
Pt      4.15803792      4.00025057      3.26262480      region=surface
Pt      5.54415256      6.40064318      3.26161295      region=surface
Pt      2.77192328      1.59985795      3.26363664      region=surface
Pt      1.38639089      2.39871548      1.00094199      region=surface
Pt      5.54436409      4.79916260      0.99921469      region=surface
Pt      6.93024583      7.19969029      1.00043291      region=surface
Pt      4.15824945      2.39876998      1.00022653      region=surface
Pt      8.31622256      4.79894871      0.99962525      region=surface
Pt      9.70210429      7.19947641      1.00084348      region=surface
Pt      6.93010792      2.39855609      1.00063710      region=surface
Pt      2.77250564      4.79937648      0.99880412      region=surface
Pt      4.15838737      7.19990418      1.00002234      region=surface
End
Lattice
      8.31557575      0.00000000
      4.15778787      7.20149984
End
End
Constraints
  FixedRegion surface
End
PESExploration
  Job BindingSites
  LoadEnergyLandscape
    Path process_search.results/ams.rkf
  End
  StatesAlignment
    ReferenceRegion surface
  End
  BindingSites
    AllowDisconnected False
  End
  StructureComparison
    DistanceDifference 0.1
    NeighborCutoff 3.5
    EnergyDifference 0.1
  End
End
Engine ReaxFF
  ForceField CHONSFpTc1Ni.ff
  Charges Solver=Direct # Reduce ReaxFF engine noise
EndEngine
EOF
```

11.7 Molecular dynamics

11.7.1 Example: Simple MD for H2

Download MD_hydrogen_longrun.run

```
#!/bin/sh
$AMSBIN/ams << eor

Task MolecularDynamics

MolecularDynamics
  nSteps 1000
  TimeStep 0.1
  InitialVelocities Type=zero
  Thermostat Type=none
  Trajectory SamplingFreq=100
End

System
  Atoms [Bohr]
    H -2.0 0.0 0.0
    H  2.0 0.0 0.0
  End
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir Dresden
  Occupation Strategy=Fermi Temperature=5
  Repulsion
    forcePolynomial true
  End
  DispersionCorrection Auto
EndEngine

eor
```

11.7.2 Example: MD for a box of water

Download H2O_nreac.run

11.7.3 Example: Lattice deformations in MD

Download MD_Deformation.run

```
#!/bin/sh

# This example consists of three MD runs exercising the Deformation feature in MD:
# 1. Simple deformations of a 2D slab
# 2. Complex sequence of deformations of a 3D box including multiple simultaneous ↵
↵deformations
# 3. Combination of a uniaxial strain with barostatting the remaining directions
```

(continues on next page)

(continued from previous page)

```

# This run first oscillates the lengths of the lattice vectors of the original non-
↳orthogonal
# lattice of an argon slab to 20 and 12 Angstrom respectively and back to the
↳original values,
# with a period of 125 fs. Afterwards, it linearly morphs the lattice to a 5 by 5
↳Angstrom square.
AMS_JOBNAME=Ar-2D "$AMSBIN/ams" <<EOR
# Set the random number seed to ensure reproducible results for testing. Omit this in
↳normal runs.
RNGSeed -2056449389 -1942239902 -422783941 -563204803 439708673 85320298 1328648683 -
↳688288369

Task MolecularDynamics

MolecularDynamics
  nSteps 1000
  TimeStep 0.25
  CalcPressure True
  InitialVelocities Type=Zero
  Trajectory SamplingFreq=10
  Deformation
    StopStep 500
    Type Cosine
    TargetLength 20 12
    Period 125
  End
  Deformation
    StartStep 500
    Type Linear
    TargetLattice
      5 0 0
      0 5 0
    End
  End
End

System
  Atoms
    Ar 0.5 0.6 0.7
    Ar 3.0 3.2 3.4
  End
  Lattice
    10.0 2.0 0.0
    1.0 10.0 0.0
  End
End

Engine LennardJones
  Rmin 3.753962043231411
  Eps 3.98e-4
  Cutoff 12.0
EndEngine
EOR

# Print the evolution of the lattice parameters in the "fort.59" format
# of old standalone reaxff for testing purposes.

```

(continues on next page)

(continued from previous page)

```

echo "BEGIN fort59 nobarostat"
"$AMSBIN/amspython" "$AMSHOME/scripting/standalone/reaxff-ams/fort59.py" Ar-2D.results
echo "END fort59"

# This run applies a sequence of deformations to a box of Lennard-Jones argon atoms.
AMS_JOBNAME=Ar-box "$AMSBIN/ams" <<EOR
# Set the random number seed to ensure reproducible results for testing. Omit this in
↳normal runs.
RNGSeed 1450250278 2029462689 827312839 1518230169 1408825437 1769469410 -406759494 -
↳1006638501

Task MolecularDynamics

MolecularDynamics
  nSteps 10000
  InitialVelocities Type=Random Temperature=300
  Thermostat Type=NHC Temperature=300 Tau=100
  Timestep 1
  Trajectory SamplingFreq=100
  CalcPressure True
  # Compress the a and b vectors to 20 Angstrom and stretch the c vector to 40
↳Angstrom over 2000 steps.
  Deformation
    StartStep 0
    StopStep 2000
    Type Linear
    TargetLength 20 20 40
  End
  # Oscillate the c vector (z axis) from 40 to 30 Angstroms and back again with a
↳period of 1000 fs.
  Deformation
    StartStep 2000
    StopStep 3500
    Type Cosine
    TargetLattice
      20.0  0.0  0.0
      0.0  20.0  0.0
      0.0   0.0 30.0
  End
  Period 1000
End
# Shear the box by tilting the c vector 10 A away from the z axis in the xz plane.
Deformation
  StartStep 3500
  StopStep 4000
  Type Linear
  TargetLattice
    20.0  0.0  0.0
    0.0  20.0  0.0
    10.0  0.0 30.0
  End
End
# The following two blocks apply simultaneous cosine and sine oscillations to the
# x and y components of the c vector. The amplitude of both oscillations is the
↳same,
# indirectly defined by the maximum velocity (velocity at the inflection point).

```

(continues on next page)

(continued from previous page)

```

# The c vector thus traces out a cone around the z axis over 2000 fs.
Deformation
  StartStep 4000
  StopStep 8000
  Type Cosine
  LatticeVelocity
    0.0 0.0 0.0
    0.0 0.0 0.0
    -3e-2 0.0 0.0
  End
  Period 2000
End
Deformation
  StartStep 4000
  StopStep 8000
  Type Sine
  LatticeVelocity
    0.0 0.0 0.0
    0.0 0.0 0.0
    0.0 3e-2 0.0
  End
  Period 2000
End
# This un-shears the box again, returning the c vector to the z axis.
Deformation
  StartStep 8000
  StopStep 8500
  Type Linear
  TargetLattice
    20.0 0.0 0.0
    0.0 20.0 0.0
    0.0 0.0 30.0
  End
End
# Finally, apply an exponential ("true") strain by compressing the c vector by 1/
↪1000th every step.
Deformation
  StartStep 8500
  Type Exponential
  LengthRate 0.0 0.0 -1e-3
End
End

System
  Atoms
    Ar 22.97333 5.533091 19.78068
    Ar 5.255777 10.75370 3.164254
    Ar 8.594419 0.957110 14.00514
    Ar 8.407438 7.306515 2.762099
    Ar 5.335001 18.10475 11.25000
    Ar 13.84870 24.00523 1.485310
    Ar 18.65495 8.812906 24.08580
    Ar 23.50126 5.372569 24.77962
    Ar 20.32336 23.84153 3.669808
    Ar 7.692337 5.389691 10.57909
    Ar 2.407753 7.779618 0.755645
    Ar 1.520220 15.74803 13.75000

```

(continues on next page)

(continued from previous page)

```

    Ar 2.921397 15.34413 20.41981
    Ar 1.660913 18.10666 3.895263
    Ar 19.54458 18.76246 20.66279
    Ar 3.853819 4.537228 14.15265
    Ar 17.57174 15.47554 17.14501
    Ar 8.601399 2.836638 1.872337
    Ar 2.786004 3.601185 22.17957
    Ar 0.047572 18.11282 24.78959
    Ar 0.172965 0.035326 0.223982
    Ar 6.604983 7.254174 21.83257
    Ar 16.63254 15.02432 7.992034
    Ar 7.941898 9.130224 17.89523
    Ar 18.26063 20.70600 5.643385
    Ar 22.63932 15.20789 23.33429
    Ar 1.138343 21.37810 13.97294
    Ar 0.461841 18.99572 6.401096
    Ar 22.67542 22.89935 9.344785
    Ar 7.215648 18.28717 3.411308

End
Lattice
    25.0 0.0 0.0
    0.0 25.0 0.0
    0.0 0.0 25.0

end
end

Engine LennardJones
    Eps 3.67e-4
    RMin 3.4
    Cutoff 12.0
EndEngine
EOR

# Print the evolution of the lattice parameters in the "fort.59" format
# of old standalone reaxff for testing purposes.
echo "BEGIN fort59 nobarostat"
"$AMSBIN/amspython" "$AMSHOME/scripting/standalone/reaxff-ams/fort59.py" Ar-box.
↪results | cut -c 1-86
echo "END fort59"

# This run simulates an experiment to measure the mechanical properties of an argon
↪box
# by stretching the lattice in the z direction by 100 ppm per timestep (1 fs) while
↪simultaneously
# keeping the x and y axes relaxed using a barostat.
AMS_JOBNAME=Ar-box-barostat "$AMSBIN/ams" <<EOR
# Set the random number seed to ensure reproducible results for testing. Omit this in
↪normal runs.
RNGSeed 877404503 1621112889 -559440914 -1857103174 -1082101960 -1842609385 -
↪845245172 -279655974

Task MolecularDynamics

MolecularDynamics
    nSteps 8000
    InitialVelocities Type=Random Temperature=300

```

(continues on next page)

(continued from previous page)

```
Thermostat Type=NHC Temperature=300 Tau=100
Barostat Type=MTK Pressure=1e7 Tau=1000 Scale=XY Equal=XY
Timestep 1
Trajectory SamplingFreq=100
Deformation
  Type Exponential
  StrainRate
    0.0 0.0 0.0
    0.0 0.0 0.0
    0.0 0.0 1e-4
  End
End
End
System
  Atoms
    Ar 22.97333 5.533091 19.78068
    Ar 5.255777 10.75370 3.164254
    Ar 8.594419 0.957110 14.00514
    Ar 8.407438 7.306515 2.762099
    Ar 5.335001 18.10475 11.25000
    Ar 13.84870 24.00523 1.485310
    Ar 18.65495 8.812906 24.08580
    Ar 23.50126 5.372569 24.77962
    Ar 20.32336 23.84153 3.669808
    Ar 7.692337 5.389691 10.57909
    Ar 2.407753 7.779618 0.755645
    Ar 1.520220 15.74803 13.75000
    Ar 2.921397 15.34413 20.41981
    Ar 1.660913 18.10666 3.895263
    Ar 19.54458 18.76246 20.66279
    Ar 3.853819 4.537228 14.15265
    Ar 17.57174 15.47554 17.14501
    Ar 8.601399 2.836638 1.872337
    Ar 2.786004 3.601185 22.17957
    Ar 0.047572 18.11282 24.78959
    Ar 0.172965 0.035326 0.223982
    Ar 6.604983 7.254174 21.83257
    Ar 16.63254 15.02432 7.992034
    Ar 7.941898 9.130224 17.89523
    Ar 18.26063 20.70600 5.643385
    Ar 22.63932 15.20789 23.33429
    Ar 1.138343 21.37810 13.97294
    Ar 0.461841 18.99572 6.401096
    Ar 22.67542 22.89935 9.344785
    Ar 7.215648 18.28717 3.411308
  End
  Lattice
    25.0 0.0 0.0
    0.0 25.0 0.0
    0.0 0.0 25.0
end
end
Engine LennardJones
  Eps 3.67e-4
  RMin 3.4
```

(continues on next page)

(continued from previous page)

```

Cutoff 12.0
EndEngine
EOR

# Print the evolution of the lattice parameters in the "fort.59" format
# of old standalone reaxff for testing purposes.
echo "BEGIN fort59 barostat"
"$AMSBIN/amspython" "$AMSHOME/scripting/standalone/reaxff-ams/fort59.py" Ar-box-
↪barostat.results | cut -c 1-86
echo "END fort59"

```

11.8 Vibrational analysis

11.8.1 Example: Mode Refinement

Download VTools_dydrogesterone.run

```

#!/bin/sh

# This example shows a mode refinement of the band associated with the
# C=O and C=C stretch modes in the dydrogesterone molecule. This was one
# of the example calculations in the original paper on mode refinement:
#
#   J. Phys. Chem. Lett., 2018, 9 (23), pp 6878-6882

# Step 1: Get DFTB modes at the optimized DFT geometry
# -----

AMS_JOBNAME=FREQ_DFTB $AMSBIN/ams << EOF

System
Atoms
# Dydrogesterone geometry already optimized with DFT (BP86/TZP).
C 0.179402320119871 1.1462568499773749 -1.34045805553897
C -1.0397129548582973 1.4038864822738149 -0.42742864655449175
C -1.9187723039939633 0.15407663507305838 -0.2887082902723144
C 5.26339021966562 -0.0803229279006518 0.15901574737297627
C 4.387948120190181 -1.2500901532169464 0.1985138671042922
C 3.0305058270434313 -1.2006020082109579 0.06687676611264043
C 2.2733069275281843 0.12162323554336309 -0.08656132560038762
C 3.224495049036027 1.1562588842598234 -0.7427294996476081
C 4.589580248129175 1.2673670000240371 -0.055599205726051454
C -2.593712600512111 -0.1967276203097069 -1.6287069723510241
C -2.011183672049868 -2.0904848406306247 0.6637924766824627
C -3.24333170765593 -1.3040476226855777 1.1949612943873085
C -2.9910039148987044 0.19289628304945772 0.8647395097635705
C 1.8832501895313913 0.6066618903839885 1.3319251022509289
C -1.021215985785471 -0.9883577256376227 0.2595406535945511
C 0.950572917185341 -2.499497885394989 -0.17845795742396
C 2.2618422613458535 -2.429295894019406 0.12998062227671012
C 0.14931427176016024 -1.3398555308999998 -0.6810919840203331
C 1.0424571714734645 -0.11148243146945544 -1.0381151732514977
C -4.261014768367461 0.9709543041281233 0.5302962418813911

```

(continues on next page)

(continued from previous page)

```
C -4.2770282131623745 2.4457823410117943 0.8829249061723259
O -5.23153866013212 0.4348316478674291 0.006835058476534351
O 6.486629619384615 -0.1787206146841789 0.3101830744372818
H 4.496419281348679 1.7413060958817603 0.9361407723186962
H 4.889108126023183 -2.2103967123783246 0.34783394165927906
H -2.5053803890738937 0.6948315694295174 1.7185731459759712
H 2.747947397670055 2.147017249296297 -0.7608107343500633
H 3.3788188981072294 0.8618896159833123 -1.7937886227370627
H 5.278515387629924 1.9033945262275014 -0.6280820487414189
H -0.313881863508176 -1.6818474813482587 -1.6263008186833716
H 2.7970964757389787 -3.3338033858599037 0.4312075853898926
H 0.4355882565839679 -3.460258453470549 -0.09628540747796212
H 2.7788926695047853 0.7904363143118734 1.939504792825026
H 1.3142558486724543 1.5445870277271823 1.281760102768146
H 1.2817550339094792 -0.13839730232383732 1.865618408504196
H 1.519857325493626 -0.37989780969443004 -1.9963570252289853
H -3.35526145791249 2.9466219355670638 0.5538746959825623
H -0.5844412872674496 -0.5966200898063014 1.193600037988863
H 0.8128688006525261 2.043605920072621 -1.3496094336049678
H -0.18858319204633375 1.0503960522723468 -2.3730012879550473
H -1.6112343483079186 2.2461900431605373 -0.8495116990527544
H -0.710640496667605 1.7174589936204205 0.5760379282354502
H -1.5937002785491352 -2.7672883134736264 1.4205298630102559
H -2.2836000112559893 -2.709635208733019 -0.2046201598348845
H -3.3761548431942434 -1.4386051747789588 2.276540650869438
H -4.174084527825135 -1.628156544022731 0.7132317870749941
H -1.860827847389362 -0.4325042173503644 -2.4102132115597854
H -3.2757258446436426 -1.0505046678063983 -1.5382984747892279
H -3.1932088604642455 0.6502553019056425 -1.9892714020438331
H -4.314908973244162 2.55066722711135 1.9790476680048945
H -5.155478941054673 2.9311753898910595 0.44480850527661253

End
End

Task SinglePoint

Properties
  NormalModes Yes
End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-freq-1-2
EndEngine

EOF

# Step 2: Mode refinement of the DFTB C=O and C=C stretch bands at the DFT level
# -----

AMS_JOBNAME=ModeRefinement $AMSBIN/ams << EOF

LoadSystem
  File FREQ_DFTB.results/dftb.rkf
End
```

(continues on next page)

(continued from previous page)

```

Task VibrationalAnalysis

VibrationalAnalysis
  Type ModeRefinement
  NormalModes
    ModeFile FREQ_DFTB.results/dftb.rkf
    ModeSelect
      FreqRange 1500 1800
    End
  End
End

Engine BAND
  # Settings from the paper:
  XC
    GGA BP86
  End
  Basis
    Type TZP
  End
  Relativity Level=None
  # Just to make this test run faster:
  NumericalQuality Basic
EndEngine

EOF

```

11.8.2 Example: Mode Tracking

Download VATools_cyclohexanone.run

```

#!/bin/sh

# This example demonstrates the usage of the AMS vibrational analysis tools
# on the cyclohexanone molecule.

# 1. Optimization with DFT
# -----

AMS_JOBNAME=DFT $AMSBIN/ams << EOF

System
  Atoms
    C      0.000000   0.000000   0.000000
    C      0.978606   1.950589  -1.251328
    C      0.978606   1.950589   1.251328
    C      0.812569   2.806732   0.000000
    C     -0.039622   0.802151   1.272926
    C     -0.039622   0.802151  -1.272926
    O      0.062619  -1.251283   0.000000
    H      0.123306   0.119861   2.122758
    H     -1.060012   1.229063   1.372200
    H      0.871524   2.560541   2.167037
    H      2.001065   1.521810   1.263746

```

(continues on next page)

(continued from previous page)

```
      H      -0.196785    3.268312    0.000000
      H      1.549313    3.631809    0.000000
      H      0.871524    2.560541   -2.167037
      H      2.001065    1.521810   -1.263746
      H      0.123306    0.119861   -2.122758
      H      -1.060012    1.229063   -1.372200
      End
    End

    Task GeometryOptimization

    GeometryOptimization
      Convergence Gradients=1.0e-4
    End

    Engine BAND
    EndEngine

EOF

# 2. Obtain DFTB hessian and modes as preconditioner and guess
# -----
AMS_JOBNAME=DFTB $AMSBIN/ams << EOF

  LoadSystem
    File DFT.results/ams.rkf
  End

  Task GeometryOptimization

  GeometryOptimization
    Convergence Gradients=1.0e-4
  End

  Properties
    NormalModes Yes
  End

  Engine DFTB
    Model DFTB3
    ResourcesDir DFTB.org/3ob-freq-1-2
  EndEngine

EOF

# 3. ModeScanning of DFTB C=O stretch mode with DFT
# -----
AMS_JOBNAME=ModeScanning $AMSBIN/ams << EOF

  LoadSystem
    File DFTB.results/ams.rkf
  End
  LoadEngine DFT.results/band.rkf
```

(continues on next page)

(continued from previous page)

```

Task VibrationalAnalysis

VibrationalAnalysis
  Type ModeScanning
  NormalModes
    ModeFile DFTB.results/dftb.rkf
    ModeSelect
      HighIR 1 # This should select the C=O stretch
    End
  End
End
End

EOF

# 4. Mode tracking with DFT starting from DFTB C=O stretch mode
# -----

AMS_JOBNAME=ModeTracking $AMSBIN/ams << EOF

LoadSystem
  File DFT.results/ams.rkf
End
LoadEngine DFT.results/band.rkf

Task VibrationalAnalysis

VibrationalAnalysis
  Type ModeTracking
  ModeTracking
    HessianGuess File
    HessianPath DFTB.results
  End
  NormalModes
    ModeInputFormat File
    ModeFile DFTB.results/dftb.rkf
    ModeSelect
      HighIR 1 # This should select the C=O stretch
    End
  End
End
End

EOF

# 5. Mode tracking with DFT starting from a pure C=O stretch
# -----

AMS_JOBNAME=ModeTracking_COSTretch $AMSBIN/ams << EOF

LoadSystem
  File DFT.results/ams.rkf
End
LoadEngine DFT.results/band.rkf

Task VibrationalAnalysis

```

(continues on next page)

(continued from previous page)

```

VibrationalAnalysis
  Type ModeTracking
  NormalModes
    ModeInputFormat Inline
    ModeInline
      0.0 0.0 0.7071 # This is the C attached to the O
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 -0.7071 # This is the O
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
      0.0 0.0 0.0
    End
    ModeSelect
      HighIR 1 # This should select the C=O stretch
    End
  End
  ModeTracking
    HessianGuess File
    HessianPath DFTB.results
    TrackingMethod OverlapPrevious
      #      ^-- Probably better than the default.
      #      Our initial mode is not particularly close yet ...
  End
End
EOF

```

11.8.3 Example: Vibronic-Structure Tracking

Download `VST_pyrene.run`

```

#!/bin/sh

# This example demonstrates the usage of vibronic-structure tracking
# on the pyrene molecule. Excited state calculations are performed within AMS and
# at the DFTB level

# 1. Ground state optimization with DFTB
#-----
AMS_JOBNAME=GroundState $AMSBIN/ams << eor

```

(continues on next page)

(continued from previous page)

```

System
  Atoms
    C      -0.01654981      1.36506537      0.00802668
    C       0.00281590     -0.06808050      0.00051978
    C       1.19835953      2.08859192      0.00127327
    C      -1.25056871      2.05539572      0.02157814
    C       1.23682878     -0.75840671     -0.01375673
    C      -1.21209870     -0.79160461      0.00671225
    C       2.44308841      1.35898462     -0.01256813
    C       1.16450680      3.48665407      0.00772934
    C      -2.47512291      1.29231475      0.02824995
    C      -1.25452039      3.45386389      0.02759561
    C       2.46138768      0.00467142     -0.01984509
    C       1.24076149     -2.15686407     -0.02209661
    C      -2.45682219     -0.06199957      0.02121310
    C      -1.17826305     -2.18965609     -0.00177638
    C      -0.05432386      4.15935390      0.02049748
    C       0.04055488     -2.86234799     -0.01620562
    H       3.36596464      1.93801392     -0.01731234
    H       2.09583666      4.04986513      0.00239299
    H      -3.41330596      1.84611463      0.03902307
    H      -2.20073438      3.99162601      0.03767421
    H       3.39957176     -0.54912830     -0.03053168
    H       2.18696724     -2.69462025     -0.03320657
    H      -3.37969722     -0.64102946      0.02611688
    H      -2.10959908     -2.75286422      0.00274662
    H      -0.06904859      5.24837501      0.02503968
    H       0.05526108     -3.95135857     -0.02284989
  End
End

Task GeometryOptimization

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-freq-1-2
EndEngine

eor

# 2. Get lowest singlet-singlet excitation of pyrene and
# compute the excited state gradient at the optimized GS geometry
#-----

AMS_JOBNAME=ExcitedState $AMSBIN/ams << eor

Task SinglePoint
LoadSystem
  File ./GroundState.results/dftb.rkf
End

Properties
  Gradients yes
End

Engine DFTB
  Model DFTB3

```

(continues on next page)

(continued from previous page)

```
ResourcesDir DFTB.org/3ob-freq-1-2
Properties
  Excitations
    TDDFTB
      Calc Singlet
      Lowest 1
    End
    TDDFTBGradients
      Excitation 1
    End
  End
End
EndEngine

eor

# 3. Vibronic-structure tracking for this excitation
# -----

AMS_JOBNAME=VibronicStructure $AMSBIN/ams << eor

LoadSystem
  File ./GroundState.results/dftb.rkf
End

Task VibrationalAnalysis

VibrationalAnalysis
  Type VibronicStructureTracking
  ExcitationSettings
    ExcitationInputFormat File
    ExcitationFile ./ExcitedState.results/dftb.rkf
    Singlet
    A 1
  End
End
AbsorptionSpectrum
  AbsorptionRange -500.0 4000.0
  LineWidth 100
  FrequencyGridPoints 181
End
End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-freq-1-2
EndEngine

eor

# 4. Restart the VST run from the previous one
# We have changed our mind and we want
# the peaks to be a little sharper
# -----

AMS_JOBNAME=VibronicStructureRestart $AMSBIN/ams << eor
```

(continues on next page)

(continued from previous page)

```

LoadSystem
  File ./GroundState.results/dftb.rkf
End

Task VibrationalAnalysis

VibrationalAnalysis
  Type VibronicStructureTracking
  ExcitationSettings
    ExcitationInputFormat File
    ExcitationFile ./ExcitedState.results/dftb.rkf
    Singlet
      A 1
    End
  End
  VSTRestartFile ./VibronicStructure.results/ams.rkf
  AbsorptionSpectrum
    AbsorptionRange -500.0 4000.0
    LineWidth 50
    FrequencyGridPoints 181
  End
End

Engine DFTB
  Model DFTB3
  ResourcesDir DFTB.org/3ob-freq-1-2
EndEngine

eor

```

11.9 PES point properties

11.9.1 Example: Phonons for graphene

Download Phonons_Graphene.run

```

#!/bin/sh

AMS_JOBNAME=graphene $AMSBIN/ams << EOF

  Task GeometryOptimization

  GeometryOptimization
!   CoordinateType Cartesian
    OptimizeLattice True
    Convergence Gradients=1.0e-5
    Method Quasi-Newton
  End

  Properties
    Phonons True
  End

  NumericalPhonons

```

(continues on next page)

(continued from previous page)

```

SuperCell
  2 0
  0 2
End
End

System
  Atoms
    C      0.000000000   -0.000000000   0.00000
    C      0.000000000   -1.420281662   0.00000
  End

  Lattice
    1.230000000   -2.130422493   0.000000000
    1.230000000    2.130422493   0.000000000
  End
End

Engine DFTB
  ResourcesDir Dresden
  Model DFTB0
  KSpace
    Type Symmetric
    Symmetric KInteg=9
  End
  Technical AnalyticalStressTensor=False # Not yet supported with symmetric k-
↪space grid ...
  EndEngine

EOF

echo ""
echo "Begin TOC of result file"

$AMSBIN/dmpkf -n 1 graphene.results/dftb.rkf --toc

echo "End TOC of result file"

```

11.9.2 Example: Phonons with isotopes

Download Phonons_Isotopes.run

```

#!/bin/sh

# =====
# Phonons with default nuclear masses:
# =====

AMS_JOBNAME=defmasses $AMSBIN/ams << EOF

Task SinglePoint

Properties
  Phonons True

```

(continues on next page)

(continued from previous page)

```

End

NumericalPhonons
  StepSize 0.01
  SuperCell
    4
  End
End

System
  Atoms
    C -2.42906152 -0.3445528299 -0.1353492062
    C -1.146891508 -1.134644249 0.1353492061
    H -2.429062041 0.004468895147 -1.185797304
    H -2.429062011 0.5753101439 0.4803683017
    H -1.146891017 -2.054507222 -0.4803683019
    H -1.146890987 -1.483665974 1.185797304
  End

  Lattice
    2.564338467 0.0 0.0
  End
End

Engine DFTB
  ResourcesDir QUASINANO2015
  Model DFTB0
  KSpace
    Type Symmetric
    Symmetric KInteg=9
  End
EndEngine

EOF

echo ""
echo "Begin TOC of result file"
$AMSBIN/dmpkf -n 1 defmasses.results/dftb.rkf --toc
echo "End TOC of result file"

# =====
# Phonons with two deuterium atoms:
# =====

AMS_JOBNAME=usermasses $AMSBIN/ams << EOF

Task SinglePoint

Properties
  Phonons true
End

NumericalPhonons
  StepSize 0.01
  SuperCell
    4

```

(continues on next page)

(continued from previous page)

```

End
End

System
  Atoms
    C  -2.42906152  -0.3445528299  -0.1353492062
    C  -1.146891508  -1.134644249   0.1353492061
    H  -2.429062041   0.004468895147  -1.185797304
    H  -2.429062011   0.5753101439   0.4803683017
    H  -1.146891017  -2.054507222  -0.4803683019  mass=2.014
    H  -1.146890987  -1.483665974   1.185797304   mass=2.014
  End
  Lattice
    2.564338467  0.0  0.0
  End
End

Engine DFTB
  ResourcesDir QUASINANO2015
  Model DFTB0
  KSpace
    Type Symmetric
    Symmetric KInteg=9
  End
EndEngine

EOF

echo ""
echo "Begin TOC of result file"
$AMSBIN/dmpkf -n 1 usermasses.results/dftb.rkf --toc
echo "End TOC of result file"

```

11.9.3 Example: User-defined Brillouin zone for phonon dispersion

Download Phonons_UserBZPath.run

```

#!/bin/sh

# This example shows how to specify a user-defined path through the
# Brillouin zone (BZ) when computing the phonon dispersion curves.

# Note: when computing the phonons, you should first run a geometry
# optimization (including relaxation of lattice vectors).
# Here we use a pre-optimized structure.

AMS_JOBNAME=BoronNitride $AMSBIN/ams << eor

Task SinglePoint

System
  Atoms
    B 0.0 0.0 0.0
    N 0.92538708336681 0.92538708336681 0.92538708336681
  End

```

(continues on next page)

(continued from previous page)

```

Lattice
  0.0          1.8507741667336 1.8507741667336
  1.8507741667336 0.0          1.8507741667336
  1.8507741667336 1.8507741667336 0.0
End
End

Properties
  Phonons Yes
End

NumericalPhonons
  SuperCell
    2 0 0
    0 2 0
    0 0 2
  End

  # Disable the automatically generated path though the BZ

  AutomaticBZPath No

  BZPath
    # In each 'Path' block you should define the vertices of the path
    # in fractional coordinates (with respect to the reciprocal lattice
    # vectors)

    # First segment: Gamma-X-W-K
    Path
      0.0  0.0  0.0  G
      0.5  0.0  0.5  X
      0.5  0.25 0.75  W
      0.375 0.375 0.75  K
    End

    # Second segment: Gamma-SomeRandomPoint
    Path
      0.0  0.0  0.0  G
      0.1  0.2  0.3  RandomPoint
    End
  End
End

Engine DFTB
  Model SCC-DFTB
  ResourcesDir DFTB.org/matsci-0-3
  KSpace Quality=Basic
EndEngine

eor

echo 'Content of phonon_curves section'
$AMSBIN/dmpkf BoronNitrade.results/dftb.rkf 'phonon_curves'
echo 'End content of phonon_curves section'

```

11.9.4 Example: Elastic tensor

Download ElasticTensor.run

```
#!/bin/sh

# === Diamond ===

AMS_JOBNAME=Diamond $AMSBIN/ams << EOF

Task GeometryOptimization

Properties
  ElasticTensor Yes
End

System
  Atoms
    C 0.44625 0.44625 2.23125
    C 2.23125 2.23125 2.23125
    C -2.23125 -2.23125 -2.23125
    C -0.44625 -0.44625 -2.23125
    C -0.44625 -2.23125 -0.44625
    C 1.33875 -0.44625 -0.44625
    C -2.23125 -0.44625 -0.44625
    C -0.44625 1.33875 -0.44625
    C -0.44625 -0.44625 1.33875
    C 1.33875 1.33875 1.33875
    C -1.33875 -1.33875 -1.33875
    C 0.44625 0.44625 -1.33875
    C 0.44625 -1.33875 0.44625
    C 2.23125 0.44625 0.44625
    C -1.33875 0.44625 0.44625
    C 0.44625 2.23125 0.44625
  End
  Lattice
    0.0 3.57 3.57
    3.57 0.0 3.57
    3.57 3.57 0.0
  End
End

GeometryOptimization
  OptimizeLattice Yes
  Convergence Gradients=1.0e-5 StressEnergyPerAtom=1.0e-5
End

Symmetry Tolerance=1e-6
ElasticTensor MaxGradientForGeoOpt=1e-5

Engine DFTB
  Model DFTB
  ResourcesDir DFTB.org/mio-1-1
  KSpace
    Type Symmetric
    Symmetric KInteg=3
  End
```

(continues on next page)

(continued from previous page)

```

    Technical AnalyticalStressTensor=False # Not yet supported with symmetric k-
↳space grid.
    EndEngine

EOF

# === Boron-Nitride sheet ===

# 3x3 super-cell, default k-space sampling

AMS_JOBNAME=BN_sheet $AMSBIN/ams << EOF

Task GeometryOptimization

Properties
    ElasticTensor Yes
End

System
    Atoms
        N 3.76095075 0.723795 0.0
        N 5.01460112 2.89518114 0.0
        B -3.76095112 -2.17138614 0.0
        B -2.50730075 0.0 0.0
        B -1.25365038 2.17138614 0.0
        B -1.25365037 -2.17138614 0.0
        B 0.0 0.0 0.0
        B 1.25365037 2.17138614 0.0
        B 1.25365038 -2.17138614 0.0
        B 2.50730075 0.0 0.0
        B 3.76095112 2.17138614 0.0
        N -2.50730112 -1.44759114 0.0
        N -1.25365075 0.723795 0.0
        N -3.8e-07 2.89518114 0.0
        N -3.7e-07 -1.44759114 0.0
        N 1.25365 0.723795 0.0
        N 2.50730037 2.89518114 0.0
        N 2.50730038 -1.44759114 0.0
    End
    Lattice
        7.52190225 0.0
        3.76095111 6.51415842
    End
End

GeometryOptimization
    OptimizeLattice Yes
    Convergence Gradients=1.0e-4
End

Engine DFTB
    Model SCC-DFTB
    ResourcesDir DFTB.org/matsci-0-3
EndEngine

EOF

```

(continues on next page)

(continued from previous page)

```

# === Polyoxyethylene ===

# primitive cell with k-space sampling

AMS_JOBNAME=Polyoxyethylene $AMSBIN/ams << EOF

  Task GeometryOptimization

  Properties
    ElasticTensor Yes
  End

  ElasticTensor
    StrainStepSize 0.002
    MaxGradientForGeoOpt 2.0e-4
  End

  System
    Atoms
      C   -0.279368361  -0.125344097  -0.026221791
      O   0.840592835   -0.919621431  -0.193214154
      H  -0.279527057   0.337014408   0.997733792
      H  -0.281697417   0.707951120  -0.778297849
    End
    Lattice
      2.240292981
    End
  End

  GeometryOptimization
    OptimizeLattice Yes
    Convergence Gradients=1.0e-4
  End

  Engine DFTB
    Model SCC-DFTB
    ResourcesDir DFTB.org/3ob-3-1
    KSpace
      Type Symmetric
      Symmetric KInteg=5
    End
    Technical AnalyticalStressTensor=False # Not yet supported with symmetric k-
↪space grid.
  EndEngine

EOF

# Note: the elastic tensor is also printed to standard output.

echo ""
echo "Extract the elastic tensor of Diamond from the rkf file:"
$AMSBIN/amsreport Diamond.results/dftb.rkf -r "AMSResults%ElasticTensor#12.4f##6"

echo ""

```

(continues on next page)

(continued from previous page)

```

echo "Extract the elastic tensor of Boron-Nitride from the rkf file:"
$AMSBIN/amsreport BN_sheet.results/dftb.rkf -r "AMSResults%ElasticTensor#12.4f##3"

echo ""
echo "Extract the elastic tensor of Polyoxyethylene from the rkf file:"
$AMSBIN/amsreport Polyoxyethylene.results/dftb.rkf -r "AMSResults%ElasticTensor#12.4f#
↪#1"

```

11.10 Pipe interface

11.10.1 Example: ASE calculator as a pipe worker

Download Pipe_ASEWorker.run

```

#!/bin/sh
# This example uses a Python pipe worker based on ASE to evaluate the Lennard-Jones_
↪potential.
# First we do a single-point calculation including the gradients and the stress_
↪tensor and then
# a short MD run similar to the Pipe_AMSonAMS example.

export NSCM=1

AMS_JOBNAME=SP "$AMSBIN/ams" << eor
RNGSeed -538839488 972444872 -468448621 535232319 -953628259 777353319 -1036072_
↪387155213

Task SinglePoint

Properties
  Gradients True
  StressTensor True
End

System
  Atoms
    Ar    0.03374714   -0.02579634   -0.03864485
    Ar   -2.47202773   -2.57754100   -0.02505375
    Ar   -2.54465030   -0.00587811   -2.49637860
    Ar   -4.98237672   -2.60930954   -2.49495517
    Ar   -0.03221746   -2.51967081   -2.47883073
    Ar   -2.52675554   -5.12905072   -2.48152312
    Ar   -2.54445364   -2.45619404   -5.02601013
    Ar   -4.95645981   -5.14903666   -4.95752059
  End
  Lattice
    0.00000000    5.00000000    5.00000000
    5.00000000    0.00000000    5.00000000
    5.00000000    5.20000000    0.00000000
  End
End

Engine Pipe
  WorkerCommand "$AMSBIN/amspython" "$TEST_DIRECTORY/ASE-LJ-worker.py"

```

(continues on next page)

(continued from previous page)

```

EndEngine

eor

AMS_JOBNAME=MD "$AMSBIN/ams" << eor
RNGSeed -538839488 972444872 -468448621 535232319 -953628259 777353319 -1036072_
↪387155213

Task MolecularDynamics

MolecularDynamics
  nSteps 200
  TimeStep 5.0
  Thermostat Type=NHC Temperature=298.15 Tau=100
  Trajectory SamplingFreq=20
  InitialVelocities Type=random Temperature=200
End

System
  Atoms
    Ar 0.0 0.0 0.0
    Ar 1.605 0.9266471820493496 2.605
  End
  Lattice
    3.21 0.0 0.0
    1.605 2.779941546148048 0.0
    0.0 0.0 5.21
  End
  SuperCell 4 4 4
End

Engine Pipe
  WorkerCommand "$AMSBIN/amspython" "$TEST_DIRECTORY/ASE-LJ-worker.py"
EndEngine

eor

```

Download ASE-LJ-worker.py

```

from scm.amspipe import ASEPipeWorker
from ase.calculators.lj import LennardJones

calculator = LennardJones()
calculator.parameters.epsilon = 0.0102996
calculator.parameters.sigma = 3.4
calculator.parameters.rc = 12.0

engine = ASEPipeWorker(calculator=calculator)
engine.run()

```

11.10.2 Example: AMS as a pipe worker

Download Pipe_AMSonAMS.run

```

#!/bin/sh
# This example runs two separate AMS processes communicating over AMSPipe. One of
↳them is the PipeWorker
# calculating Lennard-Jones energies and the other as the PipeMaster drives the MD
↳simulation.

export NSCM=1
export NSCM_AMSEXTERNAL=1

cat - > worker.in << eor
Task Pipe

Engine LennardJones
  Rmin 3.81637
  Eps 3.785e-4
  Cutoff 12.0
EndEngine
eor

"$AMSBIN/ams" << eor
RNGSeed -538839488 972444872 -468448621 535232319 -953628259 777353319 -1036072
↳387155213

Task MolecularDynamics

MolecularDynamics
  nSteps 200
  TimeStep 5.0
  Thermostat Type=NHC Temperature=298.15 Tau=100
  Trajectory SamplingFreq=20
  InitialVelocities Type=random Temperature=200
End

System
  Atoms
    Ar 0.0 0.0 0.0
    Ar 1.605 0.9266471820493496 2.605
  End
  Lattice
    3.21 0.0 0.0
    1.605 2.779941546148048 0.0
    0.0 0.0 5.21
  End
  SuperCell 4 4 4
End

Engine Pipe
  WorkerCommand "$AMSBIN/ams" < ../../worker.in > worker.out 2>&1
EndEngine
eor

echo "Worker output:"
cat ams.results/*/worker.out

```


APPENDICES

12.1 Extended XYZ file format

The `.xyz` file format is a simple text based format for molecular geometries. `.xyz` files have the number of atoms in the first line, followed by a comment line, followed by one line per atom, specifying the element as well as the x, y, and z coordinates of this atom.

However, the standard `.xyz` file format does not include lattice vectors. AMS therefore uses an extended `.xyz` file format which is also suitable for periodic systems. In this extended format the lattice vectors are specified at the end of the `.xyz` file via the keys `VEC1`, `VEC2` and `VEC3`. For 1D periodic systems (chains) only `VEC1` is needed. For 2D periodic systems (slabs) only `VEC1` and `VEC2` are needed. An example extended `.xyz` for graphene looks like this:

```
2
C      0.0  0.0  0.0
C      1.23 0.71014 0.0
VEC1   2.46 0.0  0.0
VEC2   1.23 2.13042 0.0
```

Note that the extended `.xyz` format is also understood by the AMS GUI for importing and exporting geometries from/to `.xyz` files.

12.2 Developer options

```
Print
  Timers [None | Normal | Detail | TooMuchDetail]
End
```

Print

Type Block

Description This block controls the printing of additional information to stdout.

Timers

Type Multiple Choice

Default value None

Options [None, Normal, Detail, TooMuchDetail]

Description Printing timing details to see how much time is spend in which part of the code.

```

EngineDebugging
  CheckInAndOutput Yes/No
  ForceContinousPES Yes/No
  IgnoreGradientsRequest Yes/No
  IgnorePreviousResults Yes/No
  IgnoreStressTensorRequest Yes/No
  NeverQuiet Yes/No
  RandomFailureChance float
  RandomNoiseInEnergy float
  RandomNoiseInGradients float
  RandomStopChance float
End

```

EngineDebugging**Type** Block**Description** This block contains some options useful for debugging the computational engines.**CheckInAndOutput****Type** Bool**Default value** No**Description** Enables some additional checks on the input and output of and engine, e.g. for NaN values.**ForceContinousPES****Type** Bool**Default value** No**Description** If this option is set, the engine will always run in continuous PES mode. For many engines this disables the use of symmetry, as this one always leads to a discontinuous PES around the symmetric points: Basically there is jump in the PES at the point where the symmetry detection starts classifying the system as symmetric. Normally the continuous PES mode of the engine (often disabling the symmetry) is only used when doing numerical derivatives, but this flag forces the engine to continuously run in this mode.**IgnoreGradientsRequest****Type** Bool**Default value** No**Description** If this option is set, the engine will not do analytical gradients if asked for it, so that gradients will have to be evaluated numerically by AMS.**IgnorePreviousResults****Type** Bool**Default value** No**Description** If this option is set, the engine will not receive information from previous calculations. Typically this information is used to restart the self consistent procedure of the engine.**IgnoreStressTensorRequest****Type** Bool**Default value** No

Description If this option is set, the engine will not calculate an analytical stress tensor if asked for it, so that the stress tensor will have to be evaluated numerically by AMS.

NeverQuiet

Type Bool

Default value No

Description Makes the engine ignore the request to work quietly.

RandomFailureChance

Type Float

Default value 0.0

Description Makes the engine randomly report failures, even though the results are actually fine. Useful for testing error handling on the application level.

RandomNoiseInEnergy

Type Float

Default value 0.0

Unit Hartree

Description Adds a random noise to the energy returned by the engine. The random contribution is drawn from $[-r,r]$ where r is the value of this keyword.

RandomNoiseInGradients

Type Float

Default value 0.0

Unit Hartree/Angstrom

Description Adds a random noise to the gradients returned by the engine. A random number in the range $[-r,r]$ (where r is the value of this keyword) is drawn and added separately to each component of the gradient.

RandomStopChance

Type Float

Default value 0.0

Description Makes the engine randomly stop. Can be used to simulate crashes.

12.3 Symmetry

12.3.1 Schönfliess symbols and symmetry labels

A survey of all molecular point groups that are recognized by AMS is given below. The table contains the Schönfliess symbols together with the names of the subspecies of the irreducible representations as they are used by AMS to label normal modes.

Table 12.1: Schönfliess symbols and the labels of the irreducible representations.

Point	Schönfliess	Irreducible representations
Group	Symbol in AMS	
C_1	NOSYM	A
$C_{\infty v}$	C(LIN)	Sigma Pi
$D_{\infty h}$	D(LIN)	Sigma.g Sigma.u Pi.g Pi.u
I	I	A T1 T2 G H
I_h	I(H)	A.g A.u T1.g T1.u T2.g T2.u G.g G.u H.g H.u
O	O	A1 A2 E T1 T2
O_h	O(H)	A1.g A1.u A2.g A2.u E.g E.u T1.g T1.u T2.g T2.u
T	T	A E T
T_h	T(H)	A.g A.u E.g E.u T.g T.u
T_d	T(D)	A1 A2 E T1 T2
C_i	C(I)	A.g A.u
S_n	S(N)	n=4: A B E
		n=6: A.g A.u E.g E.u
		n=8: A B E1 E2 E3
C_n	C(N)	n=2: A B
		n=4: A B E
		even n: A B E1 E2 ...
		n=3: A E
		odd n: A E1 E2 ...
C_{nv}	C(NV)	n=2: A1 A2 B1 B2
		n=4: A1 A2 B1 B2 E
		even n: A1 A2 B1 B2 E1 E2 ...
		n=3: A1 A2 E
		odd n: A1 A2 E1 E2 ...
C_s	C(S)	A' A''
C_{nh}	C(NH)	n=2: A.g A.u B.g B.u
		n=4: A.g A.u B.g B.u E.g E.u
		even n: A.g A.u B.g B.u E1.g E1.u E2.g E2.u ...
		n=3: A' A'' E' E''
		odd n: A' A'' E1' E1'' E2' E2'' ...
D_n	D(N)	n=2: A B1 B2 B3
		n=4: A1 A2 B1 B2 E
		even n: A1 A2 B1 B2 E1 E2 ...
		n=3: A1 A2 E
		odd n: A1 A2 E1 E2 ...
D_{nd}	D(ND)	n=2: A1 A2 B1 B2 E
		even n: A1 A2 B1 B2 E1 E2 ...
		n=3: A1.g A1.u A2.g A2.u E.g E.u
		odd n: A1.g A1.u A2.g A2.u E1.g E1.u E2.g E2.u ...
D_{nh}	D(NH)	n=2: A.g A.u B1.g B1.u B2.g B2.u B3.g B3.u
		n=4: A.g A.u B1.g B1.u B2.g B2.u B3.g B3.u E.g E.u
		even n: A.g A.u B1.g B1.u B2.g B2.u B3.g B3.u E1.g E1.u E2.g E2.u ...
		n=3: A1' A1'' A2' A2'' E' E''
		odd n: A1' A1'' A2' A2'' E1' E1'' E2' E2'' ...

The symmetry labeling may depend on the choice of coordinate system. For instance, B1 and B2 representations in C_{nv} are interchanged when you rotate the system by 90 degrees around the z-axis so that x-axis becomes y-axis and vice-versa (apart from sign).

12.3.2 Molecular orientation requirements

In order that AMS recognizes the (sub)symmetry of a molecule, the molecule has to have a specific orientation in space.

- The origin is a fixed point of the symmetry group.
- The z-axis is the main rotation axis.
- xy is the σ_h -plane (axial groups, C(s)).
- The x-axis is a C_2 axis (D symmetries).
- The xz-plane is a σ_v -plane (C_{nv} symmetries).
- In T_d and O_h the z-axis is a fourfold axis (S_4 and C_4 , respectively) and the (111)-direction is a threefold axis.

If the system is symmetrized (and no symmetry is given in the System block key) the molecular structure is rotranslated into this standard orientation.

REQUIRED CITATIONS

13.1 General references

When you publish results in the scientific literature that were obtained through the AMS driver program, you are required to include a reference to the program package with the appropriate release number:

AMS 2021.1, SCM, Theoretical Chemistry, Vrije Universiteit, Amsterdam, The Netherlands, <http://www.scm.com>. Optionally, you may add the following list of authors and contributors: R. Rüger, M. Franchini, T. Trnka, A. Yakovlev, E. van Lenthe, P. Philipsen, T. van Vuren, B. Klumpers, T. Soini

The engine used for a particular calculation might require you to include other references. Please refer to the specific *engine manuals* (page 211) for required citations.

In addition to these general references, certain AMS features require additional citations, in case you have used them. An overview of these is given in the *Feature references* section below.

Note: If you have used a modified (by yourself, for instance) version of the code, you should mention in the citation that a modified version has been used.

13.2 Feature references

13.2.1 Frequencies, IR Intensities, Raman, VCD

Mode tracking M. Reiher, J. Neugebauer, *A mode-selective quantum chemical method for tracking molecular vibrations applied to functionalized carbon nanotubes*, *Journal of Chemical Physics* 118, 1634 (2003) (<https://doi.org/10.1063/1.1523908>)

M. Reiher, J. Neugebauer, *Convergence characteristics and efficiency of mode-tracking calculations on pre-selected molecular vibrations*, *Physical Chemistry Chemical Physics* 6, 4621 (2004) (<http://dx.doi.org/10.1039/B406134A>)

C. Herrmann, M. Reiher, J. Neugebauer, *Finding a needle in a haystack: direct determination of vibrational signatures in complex systems*, *New Journal of Chemistry* 31, 818 (2007) (<http://dx.doi.org/10.1039/B618769M>)

Mode tracking based on intensities M. Reiher, J. Neugebauer, *A mode-selective quantum chemical method for tracking molecular vibrations applied to functionalized carbon nanotubes*, *Journal of Chemical Physics* 118, 1634 (2003) (<https://doi.org/10.1063/1.1523908>)

S. Luber, J. Neugebauer, M. Reiher, *Intensity tracking for theoretical infrared spectroscopy of large molecules*, *Journal of Chemical Physics* 130, 064105 (2009) (<https://doi.org/10.1063/1.3069834>)

- Mode refinement** T.Q. Teodoro, M.A.J. Koenis, S.E. Galembeck, V.P. Nicu, W.J. Buma, L. Visscher, *A frequency range selection method for vibrational spectra*, *J. Phys. Chem. Lett.*, 9 (23), 6878 (2018) (<https://doi.org/10.1021/acs.jpcllett.8b02963>)
- Mobile Block Hessian (MBH)** A. Ghysels, D. Van Neck, V. Van Speybroeck, T. Verstraelen and M. Waroquier, *Vibrational Modes in partially optimized molecular systems* *Journal of Chemical Physics* 126, 224102 (2007) (<https://doi.org/10.1063/1.2737444>)
- Raman scattering** S.J.A. van Gisbergen, J.G. Snijders and E.J. Baerends, *Application of time-dependent density functional response theory to Raman scattering*, *Chemical Physics Letters* 259, 599 (1996) ([https://doi.org/10.1016/0009-2614\(96\)00858-5](https://doi.org/10.1016/0009-2614(96)00858-5))
- Resonance Raman: excited-state finite lifetime** L. Jensen, L. Zhao, J. Autschbach and G.C. Schatz, *Theory and method for calculating resonance Raman scattering from resonance polarizability derivatives*, *Journal of Chemical Physics* 123, 174110 (2005) (<https://doi.org/10.1063/1.2046670>)
- Resonance Raman: excited-state gradient** J. Neugebauer, E.J. Baerends, E. Efremov, F. Ariese and C. Gooijer, *Combined Theoretical and Experimental Deep-UV Resonance Raman Studies of Substituted Pyrenes*, *Journal of Physical Chemistry A* 109, 2100 (2005) (<https://doi.org/10.1021/jp045360d>)
- VROA: (Resonance) vibrational Raman optical activity** L. Jensen, J. Autschbach, M. Krykunov, and G.C. Schatz, *Resonance vibrational Raman optical activity: A time-dependent density functional theory approach*, *Journal of Chemical Physics* 127, 134101 (2007) (<https://doi.org/10.1063/1.2768533>)
- Vibrational Circular Dichroism (VCD)** V.P. Nicu J. Neugebauer S.K. Wolff and E.J. Baerends, *A vibrational circular dichroism implementation within a Slater-type-orbital based density functional framework and its application to hexa- and hepta-helicenes*, *Theoretical Chemical Accounts* 119, 245 (2008) (<https://doi.org/10.1007/s00214-006-0234-x>)
- VCD analysis: VCDtools** V.P. Nicu, J. Neugebauer and E.J. Baerends, *Effects of Complex Formation on Vibrational Circular Dichroism Spectra*, *Journal of Physical Chemistry A* 112, 6978 (2008) (<https://doi.org/10.1021/jp710201q>)
- M.A.J. Koenis, O. Visser, L. Visscher, W.J. Buma, V.P. Nicu, *GUI Implementation of VCDtools, A Program to Analyze Computed Vibrational Circular Dichroism Spectra*, *J. Chem. Inf. Model* 60, 259 (2020) (<https://pubs.acs.org/doi/abs/10.1021/acs.jcim.9b00956>)
- V.P. Nicu, *Revisiting an old concept: the coupled oscillator model for VCD. Part 1: the generalised coupled oscillator mechanism and its intrinsic connection to the strength of VCD signals*, *Physical Chemistry Chemical Physics* 18, 21202 (2016) (<https://doi.org/10.1039/C6CP01282E>).
- Franck-Condon factors** J.S. Seldenthuis, H.S.J. van der Zant, M.A. Ratner and J.M. Thijssen, *Vibrational Excitations in Weakly Coupled Single-Molecule Junctions: A Computational Analysis*, *ACS Nano* 2, 1445 (2008) (<https://doi.org/10.1021/nn800170h>)

13.2.2 PES Exploration

Samuel T Chill, Matthew Welborn, Rye Terrell, Liang Zhang, Jean-Claude Berthet, Andreas Pedersen, Hannes Jónsson and Graeme Henkelman *EON: software for long time simulations of atomic scale systems*, *Modelling Simul. Mater. Sci. Eng.* 22 055002 (2014) (<https://doi.org/10.1088/0965-0393/22/5/055002>)

EXTERNAL PROGRAMS AND LIBRARIES

[Click here](#) for the list of programs and/or libraries used in the AMS package. On some platforms optimized libraries have been used and/or vendor specific MPI implementations.

KEYWORDS

15.1 Links to manual entries

ams:

- *Constraints* (page 69)
- *ElasticTensor* (page 148)
- *Engine* (page 211)
- *EngineAddons* (page 223)
- *EngineDebugging* (page 314)
- *GCMC* (page 118)
- *GeometryOptimization* (page 43)
- *IRC* (page 72)
- *LoadSystem* (page 37)
- *MolecularDynamics* (page 77)
- *Molecules* (page 209)
- *NEB* (page 65)
- *NormalModes* (page 155)
- *NumericalDifferentiation* (page 150)
- *NumericalPhonons* (page 178)
- *PESExploration* (page 127)
- *PESPointCharacter* (page 146)
- *PESScan* (page 61)
- *Print* (page 313)
- *Properties* (page 8)
- *Raman* (page 181)
- *Restrains* (page 224)
- *Symmetry* (page 33)
- *System* (page 29)
- *System > ElectrostaticEmbedding* (page 36)
- *Task* (page 8)
- *Thermo* (page 175)
- *TransitionStateSearch* (page 58)
- *UseSymmetry* (page 34)
- *VibrationalAnalysis > NormalModes > ModeSelect* (page 171)

ams_interactive:

- *Stop* (page 13)
- *StopProperties* (page 14)

analysis:

- *AutoCorrelation* (page 233)

- *Histogram* (page 231)
- *RadialDistribution* (page 230)
- *Task* (page 230)
- *TrajectoryInfo* (page 229)

pipe:

- *WorkerCommand* (page 26)

15.2 Summary of all keywords

Constraints

Type Block

Description The Constraints block allows geometry optimizations and potential energy surface scans with constraints. The constraints do not have to be satisfied at the start of the calculation.

Angle

Type String

Recurring True

Description Fix the angle between three atoms. Three atom indices followed by an angle in degrees.

Atom

Type Integer

Recurring True

Description Fix the position of an atom. Just one integer referring to the index of the atom in the [System%Atoms] block.

AtomList

Type Integer List

Recurring True

Description Fix positions of the specified atoms. A list of integers referring to indices of atoms in the [System%Atoms] block.

Block

Type String

Recurring True

Description Name of the region to constrain as a rigid block. Regions are specified in the System%Atoms block.

BlockAtoms

Type Integer List

Recurring True

Description List of atom indices for a block constraint, where the internal degrees of freedom are frozen.

Coordinate

Type String

Recurring True

Description Fix a particular coordinate of an atom. Atom index followed by (x|y|z).

DifDist

Type String

Recurring True

Description Four atom indices i j k l followed by the distance in Angstrom. This will constrain the difference $R(ij)-R(kl)$ at the given value.

Dihedral

Type String

Recurring True

Description Fix the dihedral angle between four atoms. Four atom indices followed by an angle in degrees.

Distance

Type String

Recurring True

Description Fix the distance between two atoms. Two atom indices followed by the distance in Angstrom.

EqualStrain

Type String

Description Exclusively for lattice optimizations: Accepts a set of strain components [xx, xy, xz, yy, yz, zz] which are to be kept equal. The applied strain will be determined by the average of the corresponding stress tensors components. In AMSinput just check the corresponding check buttons.

FixedRegion

Type String

Recurring True

Description Fix positions of all atoms in a region.

FreezeStrain

Type String

Description Exclusively for lattice optimizations: Freezes any lattice deformation corresponding to a particular component of the strain tensor. Accepts a set of strain components [xx, xy, xz, yy, yz, zz] to be frozen. In AMSinput just check the corresponding check buttons.

SumDist

Type String

Recurring True

Description Four atom indices i j k l followed by the distance in Angstrom. This will constrain the sum $R(ij)+R(kl)$ at the given value.

ElasticTensor

Type Block

Description Options for numerical evaluation of the elastic tensor.

MaxGradientForGeoOpt

Type Float

Default value 0.0001

Unit Hartree/Angstrom

GUI name Maximum nuclear gradient

Description Maximum nuclear gradient for the relaxation of the internal degrees of freedom of strained systems.

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

StrainStepSize

Type Float

Default value 0.001

Description Step size (relative) of strain deformations used for computing the elastic tensor numerically.

Engine

Type Block

Description The input for the computational engine. The header of the block determines the type of the engine.

EngineAddons**Type** Block**Description** This block configures all the engine add-ons.**AtomEnergies****Type** Non-standard block**Description** Add an element-dependent energy per atom. On each line, give the chemical element followed by the energy (in atomic units).**D3Dispersion****Type** Block**Description** This block configures the add-on that adds the Grimme D3 dispersion correction to the engine's energy, gradients, and stress tensor.**Damping****Type** Multiple Choice**Default value** BJ**Options** [BJ, Zero]**Description** Type of damping: BJ (Becke-Johnson) or Zero. BJ is recommended for most applications.**Enabled****Type** Bool**Default value** No**Description** Enables the D3 dispersion correction add-on.**Functional****Type** String**Default value** PBE**Description** Use the D3 parameterization by Grimme for a given xc-functional. Accepts the same values as the `-func` command line option of the official `dftd3` program. Note: the naming convention is different from elsewhere in the AMS suite. For example, BLYP should be called `b-lyp`.**a1****Type** Float**Description** The `a1` parameter. Only used if Damping is set to BJ. If set, it overwrites the `a1` value for the chosen functional.**a2****Type** Float**Description** The `a2` parameter. Only used if Damping is set to BJ. If set, it overwrites the `a2` value for the chosen functional.**s6****Type** Float

Description The s6 parameter, global scaling parameter. If set, it overwrites the s6 value for the chosen functional.

s8

Type Float

Description The s8 parameter. If set, it overwrites the s8 value for the chosen functional.

sr6

Type Float

Description The sr6 parameter. Only used if Damping is set to Zero. If set, it overwrites the sr6 value for the chosen functional.

D4Dispersion

Type Block

Description This block configures the addon that adds the Grimme D4(EEQ) dispersion correction to the engine's energy, gradients, stress tensor and Hessian.

Enabled

Type Bool

Default value No

Description Enables the D4 dispersion correction addon.

Functional

Type Multiple Choice

Default value PBE

Options [HF, BLYP, BPBE, BP86, BPW, LB94, MPWLYP, MPWPW91, OLYP, OPBE, PBE, RPBE, REVPBE, PW86PBE, RPW86PBE, PW91, PW91P86, XLYP, B97, TPSS, REVTPSS, SCAN, B1LYP, B3LYP, BHLYP, B1P86, B3P86, B1PW91, B3PW91, O3LYP, REVPBE0, REVPBE38, PBE0, PWP1, PW1PW, MPW1PW91, MPW1LYP, PW6B95, TPSSH, TPSS0, X3LYP, M06L, M06, OMEGAB97, OMEGAB97X, CAM-B3LYP, LC-BLYP, LH07TSVWN, LH07SSVWN, LH12CTSSIRPW92, LH12CTSSIFPW92, LH14TCALPBE, B2PLYP, B2GPPLYP, MPW2PLYP, PWPB95, DSDBLYP, DSDPBE, DSDPBEB95, DSDPBEP86, DSDSVWN, DODBLYP, DODPBE, DODPBEB95, DODPBEP86, DODSVWN, PBE02, PBE0DH, DFTB(3ob), DFTB(mio), DFTB(pbc), DFTB(matsci), DFTB(ob2), B1B95, MPWB1K, REVTPSSH, GLYP, REVPBE0DH, REVTPSS0, REVDSDPBEP86, REVDSDPBEPBE, REVDSDBLYP, REVDODPBEP86, B97M, OMEGAB97M, R2SCAN]

Description Use the D4 parameterization by Grimme for a given xc-functional.

Verbosity

Type Multiple Choice

Default value Silent

Options [Silent, Normal, Verbose, VeryVerbose]

Description Controls the verbosity of the dftd4 code. Equivalent to the `-silent` and `-verbose` command line switches of the official dftd4 program.

a1

Type Float

Description The a1 parameter, see D4 article. The physically reasonable range for a1 is [0.0,1.0]. If set, it overwrites the a1 value for the chosen functional.

a2

Type Float

Description The a2 parameter, see D4 article. The physically reasonable range for a2 is [0.0,7.0]. If set, it overwrites the a2 value for the chosen functional.

s6

Type Float

Description The s6 parameter, see D4 article. The physically reasonable range for s6 is [0.0,1.0]. If set, it overwrites the s6 value for the chosen functional.

s8

Type Float

Description The s8 parameter, see D4 article. The physically reasonable range for s8 is [0.0,3.0]. If set, it overwrites the s8 value for the chosen functional.

s9

Type Float

Description The s9 parameter, see D4 article. If set, it overwrites the s9 value for the chosen functional.

ExternalEngine

Type Block

Description External engine as an addon

Execute

Type String

GUI name Execute

Description execute command

ExternalStress

Type Block

Description This block configures the addon that adds external stress term to the engine's energy and stress tensor.

StressTensorVoigt

Type Float List

Unit a.u.

GUI name External stress tensor

Description The elements of the external stress tensor in Voigt notation. One should specify 6 numbers for 3D periodic system (order: xx,yy,zz,yz,xz,xy), 3 numbers for 2D periodic systems (order: xx,yy,xy) or 1 number for 1D periodic systems.

UpdateReferenceCell

Type Bool

Default value No

Description Whether or not the reference cell should be updated every time the system changes (see documentation).

PipeEngine

Type Block

Description Pipe engine as an add-on

WorkerCommand

Type String

GUI name Worker command

Description pipe worker command

Pressure

Type Float

Default value 0.0

Unit GPa

Description Add a hydrostatic pressure term to the engine's energy and stress tensor. Can only be used for 3D periodic boundary conditions.

WallPotential

Type Block

Description This block configures the add-on that adds a spherical wall potential to the engine's energy and gradients.

Enabled

Type Bool

Default value No

Description Enables the wall potential add-on.

Gradient

Type Float

Default value 10.0

Unit 1/Angstrom

Description The radial gradient outside the sphere.

Prefactor

Type Float

Default value 0.01

Unit Hartree

Description The multiplier for the overall strength of the potential.

Radius

Type Float

Default value 30.0

Unit Angstrom

Description The radius of the sphere, wherein the potential is close to zero.

EngineDebugging

Type Block

Description This block contains some options useful for debugging the computational engines.

CheckInAndOutput

Type Bool

Default value No

Description Enables some additional checks on the input and output of and engine, e.g. for NaN values.

ForceContinuousPES

Type Bool

Default value No

Description If this option is set, the engine will always run in continuous PES mode. For many engines this disables the use of symmetry, as this one always leads to a discontinuous PES around the symmetric points: Basically there is jump in the PES at the point where the symmetry detection starts classifying the system as symmetric. Normally the continuous PES mode of the engine (often disabling the symmetry) is only used when doing numerical derivatives, but this flag forces the engine to continuously run in this mode.

IgnoreGradientsRequest

Type Bool

Default value No

Description If this option is set, the engine will not do analytical gradients if asked for it, so that gradients will have to be evaluated numerically by AMS.

IgnorePreviousResults

Type Bool

Default value No

Description If this option is set, the engine will not receive information from previous calculations. Typically this information is used to restart the self consistent procedure of the engine.

IgnoreStressTensorRequest

Type Bool

Default value No

Description If this option is set, the engine will not calculate an analytical stress tensor if asked for it, so that the stress tensor will have to be evaluated numerically by AMS.

NeverQuiet

Type Bool

Default value No

Description Makes the engine ignore the request to work quietly.

RandomFailureChance

Type Float

Default value 0.0

Description Makes the engine randomly report failures, even though the results are actually fine. Useful for testing error handling on the application level.

RandomNoiseInEnergy

Type Float

Default value 0.0

Unit Hartree

Description Adds a random noise to the energy returned by the engine. The random contribution is drawn from $[-r,r]$ where r is the value of this keyword.

RandomNoiseInGradients

Type Float

Default value 0.0

Unit Hartree/Angstrom

Description Adds a random noise to the gradients returned by the engine. A random number in the range $[-r,r]$ (where r is the value of this keyword) is drawn and added separately to each component of the gradient.

RandomStopChance

Type Float

Default value 0.0

Description Makes the engine randomly stop. Can be used to simulate crashes.

EngineRestart

Type String

Description The path to the file from which to restart the engine. Should be a proper engine result file (like `adf.rkf`, `band.rkf` etc), or the name of the results directory containing it.

GCMC

Type Block

Description This block controls the Grand Canonical Monte Carlo (GCMC) task. By default, molecules are added at random positions in the simulation box. The initial position is controlled by

AccessibleVolume

Type Float

Default value 0.0

Description Volume available to GCMC, in cubic Angstroms. `AccessibleVolume` should be specified for “Accessible” and “FreeAccessible” [`VolumeOption`].

Box

Type Block

Description Boundaries of the insertion space, i.e. coordinates of the origin of an inserted molecule (coordinates of an atom of the inserted system may fall outside the box). For a

periodic dimension it is given as a fraction of the simulation box (the full 0 to 1 range by default). For a non-periodic dimension it represents absolute Cartesian coordinates in Angstrom (the system's bounding box extended by the MaxDistance value by default).

Amax

Type Float

Description Coordinate of the upper bound along the first axis.

Amin

Type Float

Description Coordinate of the lower bound along the first axis.

Bmax

Type Float

Description Coordinate of the upper bound along the second axis.

Bmin

Type Float

Description Coordinate of the lower bound along the second axis.

Cmax

Type Float

Description Coordinate of the upper bound along the third axis.

Cmin

Type Float

Description Coordinate of the lower bound along the third axis.

Ensemble

Type Multiple Choice

Default value Mu-VT

Options [Mu-VT, Mu-PT]

Description Select the MC ensemble: Mu-VT for fixed volume or Mu-PT for variable volume. When the Mu-PT ensemble is selected the [Pressure] and [VolumeChangeMax] should also be specified.

Iterations

Type Integer

GUI name Number of GCMC iterations

Description Number of GCMC moves.

MapAtomsToOriginalCell

Type Bool

Default value Yes

Description Keeps the atom (mostly) in the original cell by mapping them back before the geometry optimizations.

MaxDistance

Type Float

Default value 3.0

Unit Angstrom

GUI name Add molecules within

Description The max distance to other atoms of the system when adding the molecule.

MinDistance

Type Float

Default value 0.3

Unit Angstrom

GUI name Add molecules not closer than

Description Keep the minimal distance to other atoms of the system when adding the molecule.

Molecule

Type Block

Recurring True

GUI name Molecules

Description This block defines the molecule (or atom) that can be inserted/moved/deleted with the MC method. The coordinates should form a reasonable structure. The MC code uses these coordinates during the insertion step by giving them a random rotation, followed by a random translation to generate a random position of the molecule inside the box. Currently, there is no check to make sure all atoms of the molecule stay inside the simulation box. The program does check that the MaxDistance/MinDistance conditions are satisfied.

ChemicalPotential

Type Float

Unit Hartree

Description Chemical potential of the molecule (or atom) reservoir. It is used when calculating the Boltzmann accept/reject criteria after a MC move is executed. This value can be derived from first principles using statistical mechanics, or equivalently, it can be determined from thermochemical tables available in literature sources. For example, the proper chemical potential for a GCMC simulation in which single oxygen atoms are exchanged with a reservoir of O₂ gas, should equal 1/2 the chemical potential of O₂ at the temperature and pressure of the reservoir: $\text{cmpot} = \mu_{\text{O}}(T,P) = 1/2 * \mu_{\text{O}_2}(T,P) = 1/2 * [\mu_{\text{ref}}(T,P_{\text{ref}}) + kT * \log(P/P_{\text{ref}}) - E_{\text{diss}}]$ where the reference chemical potential $[\mu_{\text{ref}}(T,P_{\text{ref}})]$ is the experimentally determined chemical potential of O₂ at T and P_{ref} ; $kT * \log(P/P_{\text{ref}})$ is the pressure correction to the free energy, and E_{diss} is the dissociation energy of the O₂ molecule.

NoAddRemove

Type Bool

Default value No

GUI name Fix molecule

Description Set to True to tell the GCMC code to keep the number of molecules/atoms of this type fixed. It will thus disable Insert/Delete moves on this type, meaning it can only do a displacement move, or volume change move (for an NPT ensemble).

SystemName**Type** String**GUI name** Molecule**Description** String ID of a named [System] to be inserted. The lattice specified with this System, if any, is ignored and the main system's lattice is used instead.**NonAccessibleVolume****Type** Float**Default value** 0.0**GUI name** Non-accessible volume**Description** Volume not available to GCMC, in cubic Angstroms. NonAccessibleVolume may be specified for the "Free" [VolumeOption] to reduce the accessible volume.**NumAttempts****Type** Integer**Default value** 1000**GUI name** Max tries**Description** Try inserting/moving the selected molecule up to the specified number of times or until all constraints are satisfied. If all attempts fail a message will be printed and the simulation will stop. If the MaxDistance-MinDistance interval is small this number may have to be large.**Pressure****Type** Float**Default value** 0.0**Unit** Pascal**Description** Pressure used to calculate the energy correction in the Mu-PT ensemble. Set it to zero for incompressible solid systems unless at very high pressures.**Removables****Type** Non-standard block**Description** The Removables can be used to specify a list of molecules that can be removed or moved during this GCMC calculation. Molecules are specified one per line in the format following format: MoleculeName atom1 atom2 ... The MoleculeName must match a name specified in one of the [Molecule] blocks. The atom indices refer to the whole input System and the number of atoms must match that in the specified Molecule. A suitable Removables block is written to the standard output after each accepted MC move. If you do so then you should also replace the initial atomic coordinates with the ones found in the same file. If a [Restart] key is present then the Removables block is ignored.**Restart****Type** String**Description** Name of an RKF restart file. Upon restart, the information about the GCMC input parameters, the initial system (atomic coordinates, lattice, charge, etc.) and the MC molecules (both already inserted and to be inserted) are read from the restart file. The global GCMC input parameters and the MC Molecules can be modified from input. Any parameter not specified in the input will use its value from the restart file (i.e. not the default value).

Molecules found in the restart file do not have to be present as named Systems in the input, however if there is a System present that matches the name of a molecule from restart then the System's geometry will replace that found in the restart file. It is also possible to specify new Molecules in the input, which will be added to the pool of the MC molecules from restart.

Temperature

Type Float

Default value 300.0

Unit Kelvin

Description Temperature of the simulation. Increase the temperature to improve the chance of accepting steps that result in a higher energy.

UseGCPrefactor

Type Bool

Default value Yes

GUI name Use GC prefactor

Description Use the GC pre-exponential factor for probability.

VolumeChangeMax

Type Float

Default value 0.05

Description Fractional value by which logarithm of the volume is allowed to change at each step. The new volume is then calculated as $V_{\text{new}} = \exp(\text{random}(-1:1) * \text{VolumeChangeMax}) * V_{\text{old}}$

VolumeOption

Type Multiple Choice

Default value Free

Options [Free, Total, Accessible, FreeAccessible]

GUI name Volume method

Description Specifies the method to calculate the volume used to calculate the GC pre-exponential factor and the energy correction in the Mu-PT ensemble: Free: $V = \text{totalVolume} - \text{occupiedVolume} - \text{NonAccessibleVolume}$; Total: $V = \text{totalVolume}$; Accessible: $V = \text{AccessibleVolume}$; FreeAccessible: $V = \text{AccessibleVolume} - \text{occupiedVolume}$. The AccessibleVolume and NonAccessibleVolume are specified in the input, the occupiedVolume is calculated as a sum of atomic volumes.

GeometryOptimization

Type Block

Description Configures details of the geometry optimization and transition state searches.

CalcPropertiesOnlyIfConverged

Type Bool

Default value Yes

Description Compute the properties requested in the ‘Properties’ block, e.g. Frequencies or Phonons, only if the optimization (or transition state search) converged. If False, the properties will be computed even if the optimization did not converge.

Convergence

Type Block

Description Convergence is monitored for up to 4 quantities: the energy change, the Cartesian gradients, the Cartesian step size, and for lattice optimizations the stress energy per atom. Convergence criteria can be specified separately for each of these items.

Energy

Type Float

Default value 1e-05

Unit Hartree

GUI name Energy convergence

Description The criterion for changes in the energy. The energy is considered converged when the change in energy is smaller than this threshold times the number of atoms.

Gradients

Type Float

Default value 0.001

Unit Hartree/Angstrom

GUI name Gradient convergence

Description Threshold for nuclear gradients.

Step

Type Float

Default value 0.01

Unit Angstrom

GUI name Step convergence

Description The maximum Cartesian step allowed for a converged geometry.

StressEnergyPerAtom

Type Float

Default value 0.0005

Unit Hartree

Description Threshold used when optimizing the lattice vectors. The stress is considered ‘converged’ when the maximum value of $\text{stress_tensor} * \text{cell_volume} / \text{number_of_atoms}$ is smaller than this threshold (for 2D and 1D systems, the cell_volume is replaced by the cell_area and cell_length respectively).

CoordinateType

Type Multiple Choice

Default value Auto

Options [Auto, Delocalized, Cartesian]

GUI name Optimization space

Description Select the type of coordinates in which to perform the optimization. 'Auto' automatically selects the most appropriate CoordinateType for a given Method. If 'Auto' is selected, Delocalized coordinates will be used for the Quasi-Newton and SCMGO methods, while Cartesian coordinates will be used for all other methods.

EngineAutomations

Type Block

Description The optimizer can change some settings of the engine, based for instance on the error. The idea is to allow the engine to be a bit quicker at the start, and more accurate towards the end. Automations are always engine specific.

Enabled

Type Bool

Default value Yes

Description Whether or not autotions are enabled at all.

Gradient

Type Block

Recurring True

Description A gradient-based automation.

FinalValue

Type Float

Description This value will be used whenever the gradient is less than GradientLow

HighGradient

Type Float

Default value 1.0

Unit Hartree/Angstrom

Description Defines a large gradient. When the actual gradient is between GradientHigh and GradientLow a linear interpolation scheme is used for kT (on a log scale).

InitialValue

Type Float

Description This value will be used at the first geometry, and whenever the gradient is higher than GradientHigh

LowGradient

Type Float

Default value 1.0

Unit Hartree/Angstrom

Description Defines a small gradient, see GradientHigh

UseLogInterpolation

Type Bool

Default value Yes

Description Whether to use interpolation on a log (y) scale or not

Variable

Type String

Default value

Description variable to be tweaked for the engine.

Iteration

Type Block

Recurring True

Description Geometry step based automation.

FinalValue

Type Float

Description

FirstIteration

Type Integer

Default value 1

Description When the actual gradient is between the first and last iteration, a linear interpolation is used.

InitialValue

Type Float

Description This value will be used when the iteration number is smaller or equal to FirstIteration

LastIteration

Type Integer

Default value 10

Description Where the automation should reach the FinalValue

UseLogInterpolation

Type Bool

Default value Yes

Description Whether to use interpolation on a log (y) scale or not

Variable

Type String

Default value

Description variable to be tweaked for the engine.

FIRE

Type Block

Description This block configures the details of the FIRE optimizer. The keywords name correspond to the symbols used in the article describing the method, see PRL 97, 170201 (2006).

AllowOverallRotation

Type Bool

Default value Yes

Description Whether or not the system is allowed to freely rotate during the optimization. This is relevant when optimizing structures in the presence of external fields.

AllowOverallTranslation

Type Bool

Default value No

Description Whether or not the system is allowed to translate during the optimization. This is relevant when optimizing structures in the presence of external fields.

MapAtomsToUnitCell

Type Bool

Default value No

Description Map the atoms to the central cell at each geometry step.

NMin

Type Integer

Default value 5

Description Number of steps after stopping before increasing the time step again.

alphaStart

Type Float

Default value 0.1

Description Steering coefficient.

dtMax

Type Float

Default value 1.0

Unit Femtoseconds

Description Maximum time step used for the integration.

dtStart

Type Float

Default value 0.25

Unit Femtoseconds

Description Initial time step for the integration.

fAlpha

Type Float

Default value 0.99

Description Reduction factor for the steering coefficient.

fDec

Type Float

Default value 0.5

Description Reduction factor for reducing the time step in case of uphill movement.

fInc

Type Float

Default value 1.1

Description Growth factor for the integration time step.

strainMass

Type Float

Default value 0.5

Description Fictitious relative mass of the lattice degrees of freedom. This controls the stiffness of the lattice degrees of freedom relative to the atomic degrees of freedom, with smaller values resulting in a more aggressive optimization of the lattice.

HessianFree

Type Block

Description Configures details of the Hessian-free (conjugate gradients or L-BFGS) geometry optimizer.

Step

Type Block

Description

MaxCartesianStep

Type Float

Default value 0.1

Unit Angstrom

Description Limit on a single Cartesian component of the step.

MinRadius

Type Float

Default value 0.0

Unit Angstrom

Description Minimum value for the trust radius.

TrialStep

Type Float

Default value 0.0005

Unit Angstrom

Description Length of the finite-difference step when determining curvature. Should be smaller than the step convergence criterion.

TrustRadius

Type Float

Default value 0.2

Unit Angstrom

Description Initial value of the trust radius.

InitialHessian

Type Block

Description Options for initial model Hessian when optimizing systems with either the Quasi-Newton or the SCMGO method.

File

Type String

GUI name Initial Hessian from

Description KF file containing the initial Hessian (or the results dir. containing it). This can be used to load a Hessian calculated in a previously with the [Properties%Hessian] keyword.

Type

Type Multiple Choice

Default value Auto

Options [Auto, UnitMatrix, Swart, FromFile, Calculate, CalculateWithFastEngine]

GUI name Initial Hessian

Description Select the type of initial Hessian. Auto: let the program pick an initial model Hessian. UnitMatrix: simplest initial model Hessian, just a unit matrix in the optimization coordinates. Swart: model Hessian from M. Swart. FromFile: load the Hessian from the results of a previous calculation (see InitialHessian%File). Calculate: compute the initial Hessian (this may be computationally expensive and it is mostly recommended for TransitionStateSearch calculations). CalculateWithFastEngine: compute the initial Hessian with a faster engine.

KeepIntermediateResults

Type Bool

Default value No

Description Whether the full engine result files of all intermediate steps are stored on disk. By default only the last step is kept, and only if the geometry optimization converged. This can easily lead to huge amounts of data being stored on disk, but it can sometimes be convenient to closely monitor a tricky optimization, e.g. excited state optimizations going through conical intersections, etc. ...

MaxIterations

Type Integer

GUI name Maximum number of iterations

Description The maximum number of geometry iterations allowed to converge to the desired structure.

Method

Type Multiple Choice

Default value Auto

Options [Auto, Quasi-Newton, SCMGO, FIRE, L-BFGS, ConjugateGradients]

GUI name Optimization method

Description Select the optimization algorithm employed for the geometry relaxation. Currently supported are: the Hessian-based Quasi-Newton-type BFGS algorithm, the experimental SCMGO optimizer, the fast inertial relaxation method (FIRE), the limited-memory BFGS method, and the conjugate gradients method. The default is to choose an appropriate method automatically based on the engine's speed, the system size and the supported optimization options.

OptimizeLattice

Type Bool

Default value No

Description Whether to also optimize the lattice for periodic structures. This is currently only supported with the Quasi-Newton, FIRE, L-BFGS and SCMGO optimizers.

PretendConverged

Type Bool

Default value No

Description Normally a non-converged geometry optimization is considered an error. If this keyword is set to True, the optimizer will only produce a warning and still claim that the optimization is converged. (This is mostly useful for scripting applications, where one might want to consider non-converged optimizations still successful jobs.)

Quasi-Newton

Type Block

Description Configures details of the Quasi-Newton geometry optimizer.

MaxGDIISVectors

Type Integer

Default value 0

Description Sets the maximum number of GDIIS vectors. Setting this to a number >0 enables the GDIIS method.

Step

Type Block

Description

TrustRadius

Type Float

Description Initial value of the trust radius.

UpdateTSVectorEveryStep

Type Bool

Default value Yes

GUI name Update TSRC vector every step

Description Whether to update the TS reaction coordinate at each step with the current eigenvector.

SCMGO

Type Block

Description Configures details SCMGO.

ContractPrimitives

Type Bool

Default value Yes

Description Form non-redundant linear combinations of primitive coordinates sharing the same central atom

NumericalBMatrix

Type Bool

Default value No

Description Calculation of the B-matrix, i.e. Jacobian of internal coordinates in terms of numerical differentiations

Step

Type Block

Description

TrustRadius

Type Float

Default value 0.2

Description Initial value of the trust radius.

VariableTrustRadius

Type Bool

Default value Yes

Description Whether or not the trust radius can be updated during the optimization.

logSCMGO

Type Bool

Default value No

Description Verbose output of SCMGO internal data

testSCMGO

Type Bool

Default value No

Description Run SCMGO in test mode.

IRC**Type** Block**Description** Configures details of the Intrinsic Reaction Coordinate optimization.**Convergence****Type** Block**Description** Convergence at each given point is monitored for two items: the Cartesian gradient and the calculated step size. Convergence criteria can be specified separately for each of these items. The same criteria are used both in the inner IRC loop and when performing energy minimization at the path ends.**Gradients****Type** Float**Default value** 0.001**Unit** Hartree/Angstrom**GUI name** Gradient convergence**Description** Convergence criterion for the max component of the residual energy gradient.**Step****Type** Float**Default value** 0.001**Unit** Angstrom**GUI name** Step convergence**Description** Convergence criterion for the max component of the step in the optimization coordinates.**CoordinateType****Type** Multiple Choice**Default value** Cartesian**Options** [Cartesian, Delocalized]**GUI name** Coordinates used for optimization**Description** Select the type of coordinates in which to perform the optimization. Note that the Delocalized option should be considered experimental.**Direction****Type** Multiple Choice**Default value** Both**Options** [Both, Forward, Backward]**Description** Select direction of the IRC path. The difference between the Forward and the Backward directions is determined by the sign of the largest component of the vibrational normal mode corresponding to the reaction coordinate at the transition state geometry. The Forward path correspond to the positive sign of the component. If Both is selected then first the Forward path is computed followed by the Backward one.**InitialHessian**

Type Block

Description Options for initial Hessian at the transition state. The first eigenvalue of the initial Hessian defines direction of the first forward or backward step. This block is ignored when restarting from a previous IRC calculation because the initial Hessian found in the restart file is used.

File

Type String

GUI name File

Description If 'Type' is set to 'FromFile' then in this key you should specify the RKF file containing the initial Hessian (or the ams results dir. containing it). This can be used to load a Hessian calculated previously with the 'Properties%Hessian' keyword. If you want to also use this file for the initial geometry then also specify it in a 'LoadSystem' block.

Type

Type Multiple Choice

Default value Calculate

Options [Calculate, FromFile]

GUI name Initial Hessian

Description Calculate the exact Hessian for the input geometry or load it from the results of a previous calculation.

KeepConvergedResults

Type Bool

Default value Yes

Description Keep the binary RKF result file for every converged IRC point. These files may contain more information than the main ams.rkf result file.

MaxIRCSteps

Type Integer

GUI name Maximum IRC steps

Description Soft limit on the number of IRC points to compute in each direction. After the specified number of IRC steps the program will switch to energy minimization and complete the path. This option should be used when you are interested only in the reaction path area near the transition state. Note that even if the soft limit has been hit and the calculation has completed, the IRC can still be restarted with a 'RedoBackward' or 'RedoForward' option.

MaxIterations

Type Integer

Default value 300

GUI name Maximum iterations

Description The maximum number of geometry iterations allowed to converge the inner IRC loop. If optimization does not converge within the specified number of steps, the calculation is aborted.

MaxPoints

Type Integer

Default value 100

GUI name Maximum points

Description Hard limit on the number of IRC points to compute in each direction. After the specified number of IRC steps the program will stop with the current direction and switch to the next one. If both 'MaxPoints' and 'MaxIRCSteps' are set to the same value then 'MaxPoints' takes precedence, therefore this option should be used to set a limit on the number of IRC steps if you intend to use the results later for a restart.

MinEnergyProfile

Type Bool

Default value No

GUI name Minimum energy profile

Description Calculate minimum energy profile (i.e. no mass-weighting) instead of the IRC.

MinPathLength

Type Float

Default value 0.1

Unit Angstrom

Description Minimum length of the path required before switching to energy minimization. Use this to overcome a small kink or a shoulder on the path.

Restart

Type Block

Description Restart options. Upon restart, the information about the IRC input parameters and the initial system (atomic coordinates, lattice, charge, etc.) is read from the restart file. The IRC input parameters can be modified from input. Except for 'MaxPoints' and 'Direction' all parameters not specified in the input will use their values from the restart file. The 'MaxPoints' and 'Direction' will be reset to their respective default values if not specified in the input. By default, the IRC calculation will continue from the point where it left off. However, the 'RedoForward' and/or 'RedoBackward' option can be used to enforce recalculation of a part of the reaction path, for example, using a different 'Step' value.

File

Type String

GUI name Restart

Description Name of an RKF restart file generated by a previous IRC calculation. Do not use this key to provide an RKF file generated by a TransitionStateSearch or a SinglePoint calculation, use the 'LoadSystem' block instead.

RedoBackward

Type Integer

Default value 0

Description IRC step number to start recalculating the backward path from. By default, if the backward path has not been completed then start after the last completed step. If the backward path has been completed and the 'RedoBackward' is omitted then no point on the backward path will be recomputed.

RedoForward

Type Integer

Default value 0

Description IRC step number to start recalculating the forward path from. By default, if the forward path has not been completed then start after the last completed step. If the forward path has been completed and the 'RedoForward' is omitted then no point on the forward path will be recomputed.

Step

Type Float

Default value 0.2

GUI name Step size

Description IRC step size in mass-weighted coordinates, $\sqrt{\text{amu}} \cdot \text{bohr}$. One may have to increase this value when heavy atoms are involved in the reaction, or decrease it if the reactant or products are very close to the transition state.

LoadEngine

Type String

Description The path to the file from which to load the engine configuration. Replaces the Engine block.

LoadSystem

Type Block

Recurring True

Description Block that controls reading the chemical system from a KF file instead of the [System] block.

File

Type String

Description The path of the KF file from which to load the system. It may also be the results directory containing it.

Section

Type String

Default value Molecule

Description The section on the KF file from which to load the system.

Log

Type Non-standard block

Description Configures the debugging loggers. Syntax: 'Level LoggerName'. Possible Levels: All, Debug, Info, Warning, Error, Fatal.

MolecularDynamics

Type Block

Description Configures molecular dynamics (with the velocity-Verlet algorithm) with and without thermostats. This block allows to specify the details of the molecular dynamics calculation.

AddMolecules

Type Block

Recurring True

GUI name Add molecules

Description This block controls adding molecules to the system (a.k.a. the Molecule Gun). Multiple occurrences of this block are possible. By default, molecules are added at random positions in the simulation box with velocity matching the current system temperature. The initial position can be modified using one of the following keywords: Coords, CoordsBox, FractionalCoords, FractionalCoordsBox. The Coords and FractionalCoords keys can optionally be accompanied by CoordsSigma or FractionalCoordsSigma, respectively.

AtomTemperature

Type Float

Default value 0.0

Unit Kelvin

Description Add random velocity corresponding to the specified temperature to individual atoms of the molecule. This only affects rotational and internal degrees of freedom, not the net translational velocity of the inserted molecule as set by the other options.

ContactDistance

Type Float

Default value 0.0

Unit Angstrom

Description Translate the bullet along the velocity vector until it comes within ContactDistance of any other atom.

Coords

Type Float List

Unit Angstrom

Description Place molecules at or around the specified Cartesian coordinates. This setting takes precedence over other ways to specify initial coordinates of the molecule: [CoordsBox], [FractionalCoords], and [FractionalCoordsBox].

CoordsBox

Type Float List

Unit Angstrom

Description Place molecules at random locations inside the specified box in Cartesian coordinates. Coordinates of the box corners are specified as: Xmin, Xmax, Ymin, Ymax, Zmin, Zmax. This setting is ignored if Coords is used. In AMSinput, if this field is not empty it will be used instead of the default Coords.

CoordsSigma

Type Float List

Unit Angstrom

Description Sigma values (one per Cartesian axis) for a Gauss distribution of the initial coordinates. Can only be used together with Coords.

DeviationAngle

Type Float

Default value 0.0

Unit Degree

Description Randomly tilt the shooting direction up to this angle away from the VelocityDirection vector.

Energy

Type Float

Unit Hartree

Description Initial kinetic energy of the molecule in the shooting direction.

EnergySigma

Type Float

Default value 0.0

Unit Hartree

Description Sigma value for the Gauss distribution of the initial kinetic energy around the specified value. Should only be used together with Energy.

FractionalCoords

Type Float List

Description Place molecules at or around the specified fractional coordinates in the main system's lattice. For non-periodic dimensions a Cartesian value in Angstrom is expected. This setting is ignored if [Coords] or [CoordsBox] is used.

FractionalCoordsBox

Type Float List

Description Place molecules at random locations inside the box specified as fractional coordinates in the main system's lattice. Coordinates of the box corners are specified as: Xmin, Xmax, Ymin, Ymax, Zmin, Zmax. For non-periodic dimensions the Cartesian value in Angstrom is expected. This setting is ignored if [Coords], [CoordsBox], or [FractionalCoords] is used.

FractionalCoordsSigma

Type Float List

Description Sigma values (one per axis) for a Gauss distribution of the initial coordinates. For non-periodic dimensions the Cartesian value in Angstrom is expected. Can only be used together with FractionalCoords.

Frequency

Type Integer

Default value 0

Description A molecule is added every [Frequency] steps after the StartStep. There is never a molecule added at step 0.

MinDistance

Type Float

Default value 0.0

Unit Angstrom

Description Keep the minimal distance to other atoms of the system when adding the molecule.

NumAttempts

Type Integer

Default value 10

Description Try adding the molecule up to the specified number of times or until the MinDistance constraint is satisfied. If all attempts fail a message will be printed and the simulation will continue normally.

Rotate

Type Bool

Default value No

Description Rotate the molecule randomly before adding it to the system.

StartStep

Type Integer

Default value 0

Description Step number when the first molecule should be added. After that, molecules are added every Frequency steps. For example, ff StartStep=99 and Frequency=100 then a molecule will be added at steps 99, 199, 299, etc... No molecule will be added at step 0, so if StartStep=0 the first molecule is added at the step number equal to [Frequency].

StopStep

Type Integer

Description Do not add this molecule after the specified step.

System

Type String

Description String ID of the [System] that will be added with this 'gun'. The lattice specified with this System is ignored and the main system's lattice is used instead. AMSinput adds the system at the coordinates of the System (thus setting Coords to the center of the System).

Temperature

Type Float

Unit Kelvin

Description Initial energy of the molecule in the shooting direction will correspond to the given temperature.

TemperatureSigma

Type Float

Default value 0.0

Unit Kelvin

Description Sigma value for the Gauss distribution of the initial temperature the specified value. Should only be used together with Temperature.

Velocity**Type** Float**Unit** Angstrom/fs**Description** Initial velocity of the molecule in the shooting direction.**VelocityDirection****Type** Float List**Description** Velocity direction vector for aimed shooting. It will be random if not specified. In AMSinput add one or two atoms (which may be dummies). One atom: use vector from center of the system to add to that atom. Two atoms: use vector from the first to the second atom.**VelocitySigma****Type** Float**Default value** 0.0**Unit** Angstrom/fs**Description** Sigma value for the Gauss distribution of the initial velocity around the specified value. Should only be used together with Velocity.**Barostat****Type** Block**Description** This block allows to specify the use of a barostat during the simulation.**BulkModulus****Type** Float**Default value** 2200000000.0**Unit** Pascal**Description** An estimate of the bulk modulus (inverse compressibility) of the system for the Berendsen barostat. This is only used to make Tau correspond to the true observed relaxation time constant. Values are commonly on the order of 10-100 GPa (1e10 to 1e11) for solids and 1 GPa (1e9) for liquids (2.2e9 for water). Use 1e9 to match the behavior of standalone ReaxFF.**ConstantVolume****Type** Bool**Default value** No**Description** Keep the volume constant while allowing the box shape to change. This is currently supported only by the MTK barostat.**Duration****Type** Integer List**Description** Specifies how many steps should a transition from a particular pressure to the next one in sequence take.**Equal****Type** Multiple Choice

Default value None

Options [None, XYZ, XY, YZ, XZ]

Description Enforce equal scaling of the selected set of dimensions. They will be barostatted as one dimension according to the average pressure over the components.

Pressure

Type Float List

Unit Pascal

Description Specifies the target pressure. You can specify multiple pressures (separated by spaces). In that case the Duration field specifies how many steps to use for the transition from one p to the next p (using a linear ramp).

Scale

Type Multiple Choice

Default value XYZ

Options [XYZ, Shape, X, Y, Z, XY, YZ, XZ]

Description Dimensions that should be scaled by the barostat to maintain pressure. Selecting Shape means that all three dimensions and also all the cell angles are allowed to change.

Tau

Type Float

Unit Femtoseconds

GUI name Damping constant

Description Specifies the time constant of the barostat.

Type

Type Multiple Choice

Default value None

Options [None, Berendsen, MTK]

GUI name Barostat

Description Selects the type of the barostat.

BondBoost

Type Block

Recurring True

Description Forced reaction (bond boost) definitions. Multiple BondBoost blocks may be specified, which will be treated independently.

Chain

Type Block

Description Specifications of a chain of atoms. When a chain is detected the distance restraints will be activated. No other chain of this type will be detected while any restraints for this chain is active.

AtomNames

Type String

Description Atom names specifying the chain. An atom name can optionally be followed by '@' and a region name, in this case only atoms of this type from the given region will be matched. A leading '@' followed by a number indicates that this position in the chain must be occupied by the atom found earlier at the specified position in the chain. For example "O H N C @1" indicates that the last atom in the chain of the five atoms must be the first oxygen, thus defining a 4-membered ring. This is the only way to define a ring because implicit rings will not be detected. For example, "O H N C O" does not include rings.

MaxDistances

Type Float List

Unit Angstrom

Description Maximum distances for each pair of atoms in the chain. The number of distances must be one less than the number of AtomNames.

MinDistances

Type Float List

Unit Angstrom

Description Minimum distances for each pair of atoms in the chain. The number of distances must be one less than the number of AtomNames.

DistanceRestraint

Type String

Recurring True

Description Specify two atom indices followed by the distance in Angstrom, the ForceConstant (in a.u.) and, optionally, the profile type and F(Inf) (in a.u.). This restraint will try to keep the distance between the two specified atoms at the given value. For periodic systems this restraint follows the minimum image convention. Each index indicates position of the corresponding atom in the AtomNames key. Currently recognized restraint profile types: Harmonic (default), Hyperbolic, Erf.

NSteps

Type Integer

GUI name Boost lifetime

Description Number of steps the restraints will remain active until removed. Atoms participating in one reaction are not available for the given number of steps.

NumInstances

Type Integer

Default value 1

GUI name Number of instances

Description Number of reactions of this type taking place simultaneously.

CRESTMTD

Type Block

GUI name CREST_MTD

Description Input for CREST metadynamics simulation.

AddEnergy

Type Bool

Default value No

Description Add the bias energy to the potential energy (to match the gradients)

GaussianScaling

Type Block

Description Options for gradual introduction of the Gaussians

ScaleGaussians

Type Bool

Default value Yes

Description Introduce the Gaussians gradually, using a scaling function

ScalingSlope

Type Float

Default value 0.03

Description Slope of the scaling function for the Gaussians with respect to time

Height

Type Float

Unit Hartree

Description The height of the Gaussians added

NGaussiansMax

Type Integer

Description Maximum number of Gaussians stored

NSteps

Type Integer

Description Interval of Gaussian placement

RestartFile

Type String

Description Filename for file from which to read data on Gaussians placed previously.

Width

Type Float

Unit Bohr

Description The width of the Gaussians added in terms of the RMSD

CVHD

Type Block

Recurring True

GUI name CVHD

Description Input for the Collective Variable-driven HyperDynamics (CVHD).

Bias

Type Block

Description The bias is built from a series of Gaussian peaks deposited on the collective variable axis every [Frequency] steps during MD. Each peak is characterized by its (possibly damped) height and the RMS width (standard deviation).

DampingTemp

Type Float

Default value 0.0

Unit Kelvin

GUI name Bias damping T

Description During well-tempered hyperdynamics the height of the added bias is scaled down with an $\exp(-E/kT)$ factor [PhysRevLett 100, 020603 (2008)], where E is the current value of the bias at the given CV value and T is the damping temperature DampingTemp. If DampingTemp is zero then no damping is applied.

Delta

Type Float

Description Standard deviation parameter of the Gaussian bias peak.

Height

Type Float

Unit Hartree

Description Height of the Gaussian bias peak.

ColVarBB

Type Block

Recurring True

GUI name Collective Variable

Description Description of a bond-breaking collective variable (CV) as described in [Bal & Neyts, JCTC, 11 (2015)]. A collective variable may consist of multiple ColVar blocks.

at1

Type Block

Description Specifies the first bonded atom in the collective variable.

region

Type String

Default value *

Description Restrict the selection of bonded atoms to a specific region. If this is not set, atoms anywhere in the system will be selected.

symbol

Type String

Description Atom type name of the first atom of the bond. The name must be as it appears in the System block. That is, if the atom name contains an extension (e.g C.1) then the full name including the extension must be used here.

at2

Type Block

Description Specifies the second bonded atom in the collective variable.

region

Type String

Default value *

Description Restrict the selection of bonded atoms to a specific region. If this is not set, atoms anywhere in the system will be selected.

symbol

Type String

Description Atom type name of the second atom of the bond. The value is allowed to be the same as [at1], in which case bonds between atoms of the same type will be included.

cutoff

Type Float

Default value 0.3

GUI name Bond order cutoff

Description Bond order cutoff. Bonds with BO below this value are ignored when creating the initial bond list for the CV. The bond list does not change during lifetime of the variable even if some bond orders drop below the cutoff.

p

Type Integer

Default value 6

GUI name Exponent p

Description Exponent value p used to calculate the p-norm for this CV.

rmax

Type Float

Unit Angstrom

GUI name R max

Description Max bond distance parameter Rmax used for calculating the CV. It should be close to the transition-state distance for the corresponding bond.

rmin

Type Float

Unit Angstrom

GUI name R min

Description Min bond distance parameter Rmin used for calculating the CV. It should be close to equilibrium distance for the corresponding bond.

Frequency

Type Integer

Description Frequency of adding a new bias peak, in steps. New bias is deposited every [Frequency] steps after [StartStep] if the following conditions are satisfied: the current CV value is less than 0.9 (to avoid creating barriers at the transition state), the step number is greater than or equal to [StartStep], and the step number is less than or equal to [StopStep].

StartStep

Type Integer

Description If this key is specified, the first bias will be deposited at this step. Otherwise, the first bias peak is added at the step number equal to the Frequency parameter. The bias is never deposited at step 0.

StopStep

Type Integer

Description No bias will be deposited after the specified step. The already deposited bias will continue to be applied until the reaction event occurs. After that no new CVHD will be started. By default, the CVHD runs for the whole duration of the MD calculation.

WaitSteps

Type Integer

Description If the CV value becomes equal to 1 and remains at this value for this many steps then the reaction event is considered having taken place. After this, the collective variable will be reset and the bias will be removed.

CalcPressure

Type Bool

Default value No

Description Calculate the pressure in periodic systems. This may be computationally expensive for some engines that require numerical differentiation. Some other engines can calculate the pressure for negligible additional cost and will always do so, even if this option is disabled.

Checkpoint

Type Block

Description Sets the frequency for storing the entire MD state necessary for restarting the calculation.

Frequency

Type Integer

Default value 1000

GUI name Checkpoint frequency

Description Write the MD state and engine-specific data to the respective .rkf files once every N steps.

WriteProperties

Type Bool

Default value No

Description Write the properties from the properties section to the ChecoPoint file once every N steps.

Deformation

Type Block

Recurring True

Description Deform the periodic lattice of the system during the simulation.

LatticeVelocity

Type Non-standard block

Description Velocity of individual lattice vector components in Angstrom/fs. The format is identical to the System%Lattice block. For Type Sine and Cosine, this defines the maximum velocity (at the inflection point).

LengthRate

Type Float List

Default value [0.0, 0.0, 0.0]

Description Relative rate of change of each lattice vector per step.

LengthVelocity

Type Float List

Default value [0.0, 0.0, 0.0]

Unit Angstrom/fs

Description Change the length of each lattice vector with this velocity. With Type=Exponential, LengthVelocity is divided by the current lattice vector lengths on Start-Step to determine a LengthRate, which is then applied on all subsequent steps. For Type Sine and Cosine, this defines the maximum velocity (at the inflection point).

Period

Type Float

Unit Femtoseconds

Description Period of oscillation for Type Sine and Cosine.

ScaleAtoms

Type Bool

Default value Yes

Description Scale the atomic positions together with the lattice vectors. Disable this to deform only the lattice, keeping the coordinates of atoms unchanged.

StartStep

Type Integer

Default value 1

Description First step at which the deformation will be applied.

StopStep

Type Integer

Default value 0

Description Last step at which the deformation will be applied. If unset or zero, nSteps will be used instead.

StrainRate

Type Non-standard block

Description Strain rate matrix to be applied on every step. The format is identical to the System%Lattice block.

TargetLattice

Type Non-standard block

Description Target lattice vectors to be achieved by StopStep. The format is identical to the System%Lattice block.

TargetLength

Type Float List

Default value [0.0, 0.0, 0.0]

Unit Angstrom

Description Target lengths of each lattice vector to be achieved by StopStep. The number of values should equal the periodicity of the system. If a value is zero, the corresponding lattice vector will not be modified.

Type

Type Multiple Choice

Default value Linear

Options [Linear, Exponential, Sine, Cosine]

Description Function defining the time dependence of the deformed lattice parameters. Linear increments the lattice parameters by the same absolute amount every timestep. Exponential multiplies the lattice parameters by the same factor every timestep. Only StrainRate, LengthRate, and LengthVelocity are supported for Type=Exponential. Sine deforms the system from the starting lattice to TargetLattice/TargetLength and then by the same amount to the opposite direction, while Cosine deforms the system from the starting lattice to the target and back.

Gravity

Type Block

Description Apply a constant acceleration in -z.

Acceleration

Type Float

Default value 0.0

Unit Angstrom/fs²

Description Magnitude of the applied acceleration.

HeatExchange

Type Block

Recurring True

GUI name Heat exchange

Description Input for the heat-exchange non-equilibrium MD (T-NEMD).

HeatingRate

Type Float

Unit Hartree/fs

Description Rate at which the energy is added to the Source and removed from the Sink. A heating rate of 1 Hartree/fs equals to about 0.00436 Watt of power being transferred through the system.

Method

Type Multiple Choice

Default value Simple

Options [Simple, HEX, eHEX]

Description Heat exchange method used. Simple: kinetic energy of the atoms of the source and sink regions is modified irrespective of that of the center of mass (CoM) of the region (recommended for solids). HEX: kinetic energy of the atoms of these regions is modified keeping that of the corresponding CoM constant. eHEX: an enhanced version of HEX that conserves the total energy better (recommended for gases and liquids).

Sink

Type Block

Description Defines the heat sink region (where the heat will be removed).

AtomList

Type Integer List

GUI name Sink region

Description The atoms that are part of the sink. This key is ignored if the [Box] block or [Region] key is present.

Box

Type Block

Description Part of the simulation box (in fractional cell coordinates) defining the heat sink. If this block is specified, then by default, the whole box in each of the three dimensions is used, which usually does not make much sense. Normally, you will want to set the bounds along one of the axes.

Amax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the first axis.

Amin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the first axis.

Bmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the second axis.

Bmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the second axis.

Cmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the third axis.

Cmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the third axis.

Region

Type String

GUI name Sink region

Description The region that is the sink. This key is ignored if the [Box] block is present.

Source

Type Block

Description Defines the heat source region (where the heat will be added).

AtomList

Type Integer List

GUI name Source region

Description The atoms that are part of the source. This key is ignored if the [Box] block or [Region] key is present.

Box

Type Block

Description Part of the simulation box (in fractional cell coordinates) defining the heat source. If this block is specified, then by default, the whole box in each of the three dimensions is used, which usually does not make much sense. Normally, you will want to set the bounds along one of the axes. This block is mutually exclusive with the FirstAtom/LastAtom setting.

Amax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the first axis.

Amin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the first axis.

Bmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the second axis.

Bmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the second axis.

Cmax

Type Float

Default value 1.0

Description Coordinate of the upper bound along the third axis.

Cmin

Type Float

Default value 0.0

Description Coordinate of the lower bound along the third axis.

Region

Type String

GUI name Source region

Description The region that is the source. This key is ignored if the [Box] block is present.

StartStep

Type Integer

Default value 0

Description Index of the MD step at which the heat exchange will start.

StopStep

Type Integer

Description Index of the MD step at which the heat exchange will stop.

InitialVelocities

Type Block

Description Sets the frequency for printing to stdout and storing the molecular configuration on the .rkf file.

File

Type String

Description AMS RKF file containing the initial velocities.

Temperature

Type Float

Unit Kelvin

GUI name Initial temperature

Description Sets the temperature for the Maxwell-Boltzmann distribution when the type of the initial velocities is set to random, in which case specifying this key is mandatory. AMSinput will use the first temperature of the first thermostat as default.

Type

Type Multiple Choice

Default value Random

Options [Zero, Random, FromFile, Input]

GUI name Initial velocities

Description Specifies the initial velocities to assign to the atoms. Three methods to assign velocities are available. Zero: All atom are at rest at the beginning of the calculation. Random: Initial atom velocities follow a Maxwell-Boltzmann distribution for the temperature given by the [MolecularDynamics%InitialVelocities%Temperature] keyword. FromFile: Load the velocities from a previous ams result file. Input: Atom's velocities are set to the values specified in the [MolecularDynamics%InitialVelocities%Values] block, which can be accessed via the Expert AMS panel in AMSinput.

Values

Type Non-standard block

Description This block specifies the velocity of each atom, in Angstrom/fs, when [MolecularDynamics%InitialVelocities%Type] is set to Input. Each row must contain three floating point values (corresponding to the x,y,z component of the velocity vector) and a number of rows equal to the number of atoms must be present, given in the same order as the [System%Atoms] block.

NSteps

Type Integer

Default value 1000

GUI name Number of steps

Description The number of steps to be taken in the MD simulation.

Plumed

Type Block

Description Input for PLUMED. The parallel option is still experimental.

Input

Type Non-standard block

Description Input for PLUMED. Contents of this block is passed to PLUMED as is.

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

Preserve

Type Block

Description Periodically remove numerical drift accumulated during the simulation to preserve different whole-system parameters.

AngularMomentum

Type Bool

Default value Yes

GUI name : Angular momentum

Description Remove overall angular momentum of the system. This option is ignored for 2D and 3D-periodic systems.

CenterOfMass

Type Bool

Default value No

GUI name : Center of mass

Description Translate the system to keep its center of mass at the coordinate origin. This option is not very useful for 3D-periodic systems.

Momentum**Type** Bool**Default value** Yes**GUI name** Preserve: Total momentum**Description** Remove overall (linear) momentum of the system.**Print****Type** Block**Description** This block controls the printing of additional information to stdout.**System****Type** Bool**Default value** No**Description** Print the chemical system before and after the simulation.**Velocities****Type** Bool**Default value** No**Description** Print the atomic velocities before and after the simulation.**Remap****Type** Block**Description** Control periodic remapping (backtranslation) of atoms into the PBC box.**Type****Type** Multiple Choice**Default value** Atoms**Options** [None, Atoms]**Description** Select the method used to remap atoms into the unit cell. None: Disable remapping completely. Atoms: Remap any atoms that leave the unit cell.**RemoveMolecules****Type** Block**Recurring** True**GUI name** Remove molecules**Description** This block controls removal of molecules from the system. Multiple occurrences of this block are possible.**Formula****Type** String**Description** Molecular formula of the molecules that should be removed from the system. The order of elements in the formula is very important and the correct order is: C, H, all other elements in the strictly alphabetic order. Element names are case-sensitive, spaces in the formula are not allowed. Digit '1' must be omitted. Valid formula examples: C2H6O, H2O, O2S. Invalid formula examples: C2H5OH, H2O1, OH, SO2. Invalid formulas are

silently ignored. Use * to remove any molecule, which must be combined with SinkBox or SafeBox.

Frequency

Type Integer

Default value 0

Description The specified molecules are removed every so many steps after the StartStep. There is never a molecule removed at step 0.

SafeBox

Type Block

Description Part of the simulation box where molecules may not be removed. Only one of the SinkBox or SafeBox blocks may be present. If this block is present the molecule will not be removed if any of its atoms is within the box. For a periodic dimension it is given as a fraction of the simulation box (the full 0 to 1 range by default). For a non-periodic dimension it represents absolute Cartesian coordinates in Angstrom.

Amax

Type Float

Description Coordinate of the upper bound along the first axis.

Amin

Type Float

Description Coordinate of the lower bound along the first axis.

Bmax

Type Float

Description Coordinate of the upper bound along the second axis.

Bmin

Type Float

Description Coordinate of the lower bound along the second axis.

Cmax

Type Float

Description Coordinate of the upper bound along the third axis.

Cmin

Type Float

Description Coordinate of the lower bound along the third axis.

FractionalCoordsBox

Type Float List

GUI name Safe box

Description Do not remove molecules that are (partly) inside the safe box. Borders of the safe box specified as: Amin, Amax, Bmin, Bmax, Cmin, Cmax. For periodic dimensions fractional coordinates between 0 and 1 and for non-periodic dimensions Cartesian values in Angstrom are expected.

SinkBox**Type** Block**Description** Part of the simulation box where matching molecules will be removed. By default, molecules matching the formula will be removed regardless of their location. If this block is present then such a molecule will only be removed if any of its atoms is within the box. For a periodic dimension it is given as a fraction of the simulation box (the full 0 to 1 range by default). For a non-periodic dimension it represents absolute Cartesian coordinates in Angstrom.**Amax****Type** Float**Description** Coordinate of the upper bound along the first axis.**Amin****Type** Float**Description** Coordinate of the lower bound along the first axis.**Bmax****Type** Float**Description** Coordinate of the upper bound along the second axis.**Bmin****Type** Float**Description** Coordinate of the lower bound along the second axis.**Cmax****Type** Float**Description** Coordinate of the upper bound along the third axis.**Cmin****Type** Float**Description** Coordinate of the lower bound along the third axis.**FractionalCoordsBox****Type** Float List**GUI name** Sink box**Description** Remove molecules that are (partly) inside the sink box. Borders of the sink box specified as: Amin, Amax, Bmin, Bmax, Cmin, Cmax. For periodic dimensions fractional coordinates between 0 and 1 and for non-periodic dimensions Cartesian values in Angstrom are expected.**StartStep****Type** Integer**Default value** 0**Description** Step number when molecules are removed for the first time. After that, molecules are removed every [Frequency] steps. For example, if StartStep=99 and Frequency=100 then molecules will be removed at steps 99, 199, 299, etc... No molecule

will be removed at step 0, so if StartStep=0 the first molecules are removed at the step number equal to [Frequency].

StopStep

Type Integer

Description Do not remove the specified molecules after this step.

ReplicaExchange

Type Block

Description This block is used for (temperature) Replica Exchange MD (Parallel Tempering) simulations.

AllowWrongResults

Type Bool

Default value No

Description Allow combining Replica Exchange with other features when the combination is known to produce physically incorrect results.

EWMALength

Type Integer

Default value 10

Description Length of the exponentially weighted moving average used to smooth swap probabilities for monitoring. This value is equal to the inverse of the EWMA mixing factor.

SwapFrequency

Type Integer

Default value 100

Description Attempt an exchange every N steps.

TemperatureFactors

Type Float List

Description This is the ratio of the temperatures of two successive replicas. The first value sets the temperature of the second replica with respect to the first replica, the second value sets the temperature of the third replica with respect to the second one, and so on. If there are fewer values than nReplicas, the last value of TemperatureFactor is used for all the remaining replicas.

Temperatures

Type Float List

Description List of temperatures for all replicas except for the first one. This is mutually exclusive with TemperatureFactors. Exactly nReplicas-1 temperature values need to be specified, in increasing order. The temperature of the first replica is given by [Thermostat%Temperature].

nReplicas

Type Integer

Default value 1

GUI name Number of replicas

Description Number of replicas to run in parallel.

Restart

Type String

GUI name Restart from

Description The path to the ams.rkf file from which to restart the simulation.

Thermostat

Type Block

Recurring True

Description This block allows to specify the use of a thermostat during the simulation. Depending on the selected thermostat type, different additional options may be needed to characterize the specific thermostat' behavior.

BerendsenApply

Type Multiple Choice

Default value Global

Options [Local, Global]

GUI name Apply Berendsen

Description Select how to apply the scaling correction for the Berendsen thermostat: - per-atom-velocity (Local) - on the molecular system as a whole (Global).

ChainLength

Type Integer

Default value 10

GUI name NHC chain length

Description Number of individual thermostats forming the NHC thermostat

Duration

Type Integer List

GUI name Duration(s)

Description Specifies how many steps should a transition from a particular temperature to the next one in sequence take.

Region

Type String

Default value *

Description The identifier of the region to thermostat. The default '*' applies the thermostat to the entire system. The value can be a plain region name, or a region expression, e.g. '*-myregion' to thermostat all atoms that are not in myregion, or 'regionA+regionB' to thermostat the union of the 'regionA' and 'regionB'. Note that if multiple thermostats are used, their regions may not overlap.

Tau

Type Float

Unit Femtoseconds

GUI name Damping constant

Description The time constant of the thermostat.

Temperature

Type Float List

Unit Kelvin

GUI name Temperature(s)

Description The target temperature of the thermostat. You can specify multiple temperatures (separated by spaces). In that case the Duration field specifies how many steps to use for the transition from one T to the next T (using a linear ramp). For NHC thermostat, the temperature may not be zero.

Type

Type Multiple Choice

Default value None

Options [None, Berendsen, NHC]

GUI name Thermostat

Description Selects the type of the thermostat.

TimeStep

Type Float

Default value 0.25

Unit Femtoseconds

Description The time difference per step.

Trajectory

Type Block

Description Sets the frequency for printing to stdout and storing the molecular configuration on the .rkf file.

PrintFreq

Type Integer

GUI name Printing frequency

Description Print current thermodynamic properties to the output every N steps. By default this is done every SamplingFreq steps.

SamplingFreq

Type Integer

Default value 100

GUI name Sample frequency

Description Write the the molecular geometry (and possibly other properties) to the .rkf file once every N steps.

TProfileGridPoints

Type Integer

Default value 0

Description Number of points in the temperature profile. If TProfileGridPoints > 0, a temperature profile along each of the three unit cell axes will be written to the .rkf file. By default, no profile is generated.

WriteBonds

Type Bool

Default value Yes

Description Write detected bonds to the .rkf file.

WriteCharges

Type Bool

Default value Yes

Description Write current atomic point charges (if available) to the .rkf file. Disable this to reduce trajectory size if you do not need to analyze charges.

WriteGradients

Type Bool

Default value No

Description Write gradients (negative of the atomic forces) to the .rkf file.

WriteMolecules

Type Bool

Default value Yes

Description Write the results of molecule analysis to the .rkf file.

WriteVelocities

Type Bool

Default value Yes

Description Write velocities to the .rkf file. Disable this to reduce trajectory size if you do not need to analyze the velocities.

fbMC

Type Block

Recurring True

GUI name fbMC

Description This block sets up force bias Monte Carlo interleaved with the molecular dynamics simulation.

Frequency

Type Integer

Default value 1

Description Run the fbMC procedure every Frequency MD steps.

MassRoot

Type Float

Default value 2.0

Description Inverse of the exponent used to mass-weight fbMC steps.

NSteps

Type Integer

GUI name Number of steps

Description Number of fbMC steps to perform on every invocation of the procedure.

PrintFreq

Type Integer

GUI name Printing frequency

Description Print current thermodynamic properties to the output every N fbMC steps. This defaults to the PrintFreq set in the Trajectory block. Setting this to zero disables printing fbMC steps.

StartStep

Type Integer

Default value 1

Description First step at which the fbMC procedure may run.

StepLength

Type Float

Default value 0.1

Unit Angstrom

Description Maximum allowed displacement of the lightest atom in the system in each Cartesian coordinate in one fbMC step.

StopStep

Type Integer

Default value 0

Description Last step at which the fbMC procedure may run. If unset or zero, there is no limit.

Temperature

Type Float

Unit Kelvin

Description Temperature used for fbMC.

Molecules

Type Block

Description Configures details of the molecular composition analysis enabled by the Properties%Molecules block.

AdsorptionSupportRegion

Type String

GUI name Adsorption support region

Description Select region that will represent a support for adsorption analysis. Adsorbed molecules will receive an '(ads)' suffix after name of the element bonded to the support. Such elements will be listed separate from atoms of the same element not bonded to the support, for example, HOH(ads) for a water molecule bonded to a surface via one of its H atoms.

BondOrderCutoff

Type Float

Default value 0.5

Description Bond order cutoff for analysis of the molecular composition. Bonds with bond order smaller than this value are neglected when determining the molecular composition.

NEB

Type Block

Description Configures details of the Nudged Elastic Band optimization.

Climbing

Type Bool

Default value Yes

GUI name Climb highest image to TS

Description Use the climbing image algorithm to drive the highest image to the transition state.

ClimbingThreshold

Type Float

Default value 0.0

Unit Hartree/Bohr

GUI name CI force threshold

Description Climbing image force threshold. If ClimbingThreshold > 0 and the max perpendicular force component is above the threshold then no climbing is performed at this step. This entry can be used to get a better approximation for the reaction path before starting the search for the transition state. A typical value is 0.01 Hartree/Bohr.

Images

Type Integer

Default value 8

GUI name Number of images

Description Number of NEB images (not counting the chain ends). Using more images will result in a smoother reaction path and can help with convergence problems, but it will also increase the computation time.

InterpolateInternal

Type Bool

Default value Yes

GUI name Interpolate in Internal coordinates

Description The initial NEB image geometries are calculated by interpolating between the initial and the final state. By default, for non-periodic systems the interpolation is performed in Internal coordinates but the user can choose to do it in the Cartesian ones. For periodic systems the interpolation is always done in Cartesian coordinates.

InterpolateShortest

Type Bool

Default value Yes

GUI name Interpolate across cell boundary

Description Allow interpolation across periodic cell boundaries. Set to false if an atom is intended to move more than half across the simulation box during reaction.

Iterations

Type Integer

GUI name Maximum number of iterations

Description Maximum number of NEB iterations. The default value depends on the number of degrees of freedom (number of images, atoms, periodic dimensions).

Jacobian

Type Float

GUI name Jacobian value

Description Scaling factor used to convert the lattice strain to a NEB coordinate value. Default value: $\sqrt{N} \cdot (V/N)^{1/d}$, where V - lattice volume (area for 2D, length for 1D), N - number of atoms, and d - number of periodic dimensions.

MapAtomsToCell

Type Bool

Default value Yes

GUI name Map atoms to cell

Description Translate atoms to the [-0.5,0.5] cell before every step. This option cannot be disabled for SS-NEB.

OldTangent

Type Bool

Default value No

GUI name Use old tangent

Description Turn on the old central difference tangent.

OptimizeEnds

Type Bool

Default value Yes

GUI name Optimize reactants/products

Description Start the NEB with optimization of the reactant and product geometries.

OptimizeLattice

Type Bool

Default value No

GUI name Optimize lattice

Description Turn on the solid-state NEB (SS-NEB).

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

ReOptimizeEnds

Type Bool

Default value No

GUI name Re-optimize reactants/products

Description Re-optimize reactant and product geometries upon restart.

Restart

Type String

GUI name Restart from

Description Provide an ams.rkf file from a previous NEB calculation to restart from. It can be an unfinished NEB calculation or one performed with different engine parameters.

Skewness

Type Float

Default value 1.0

GUI name Skewness

Description Degree of how much images are shifted towards or away from the TS, which may help tackle problems with a long reaction path (for example involving a loose adsorption complex) without needing too many images. A value greater than 1 will make sure that images are concentrated near the transition state. The optimal value depends on the path length, the number of images (larger [Skewness] may be needed for a longer path and fewer images). Technically [Skewness] is equal to the ratio between the optimized distances to the lower and the higher neighbor image on the path.

Spring

Type Float

Default value 1.0

Unit Hartree/Bohr²

GUI name Spring value

Description Spring force constant in atomic units.

NormalModes

Type Block

Description Configures details of a normal modes calculation.

BlockDisplacements

Type Block

Description Configures details of a Block Normal Modes (a.k.a. Mobile Block Hessian, or MBH) calculation.

AngularDisplacement

Type Float

Default value 0.5

Unit Degree

Description Relative step size for rotational degrees of freedom during Block Normal Modes finite difference calculations. It will be scaled with the characteristic block size.

BlockAtoms

Type Integer List

Recurring True

Description List of atoms belonging to a block. You can have multiple BlockAtoms.

BlockRegion

Type String

Recurring True

Description The region to to be considered a block. You can have multiple BlockRegions, also in combination with BlockAtoms.

Parallel

Type Block

Description Configuration for how the individual displacements are calculated in parallel.

nCoresPerGroup

Type Integer

Description Number of cores in each working group.

nGroups

Type Integer

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Cores per task

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

RadialDisplacement

Type Float

Default value 0.005

Unit Angstrom

Description Step size for translational degrees of freedom during Block Normal Modes finite difference calculations.

Displacements

Type Multiple Choice

Default value Cartesian

Options [Cartesian, Symmetric, Block]

GUI name Displacements

Description Type of displacements. In case of symmetric displacements it is possible to choose only the modes that have non-zero IR or Raman intensity. Block displacements take rigid blocks into account.

Hessian

Type Multiple Choice

Default value Auto

Options [Auto, Analytical, Numerical]

Description Default Auto means that if possible by the engine the Hessian will be calculated analytically, else the Hessian will be calculated numerically by AMS.

ReScanFreqRange

Type Float List

Default value [-10000000.0, 10.0]

Unit cm-1

Recurring True

GUI name Re-scan range

Description Specifies a frequency range within which all modes will be scanned. 2 numbers: an upper and a lower bound.

ReScanModes

Type Bool

Default value Yes

GUI name Re-scan modes

Description Whether or not to scan imaginary modes after normal modes calculation has concluded.

SymmetricDisplacements

Type Block

Description Configures details of the calculation of the frequencies and normal modes of vibration in symmetric displacements.

Type

Type Multiple Choice

Default value All

Options [All, Infrared, Raman, InfraredAndRaman]

GUI name Symm Frequencies

Description For symmetric molecules it is possible to choose only the modes that have non-zero IR or Raman intensity (or either of them) by symmetry. In order to calculate the Raman intensities the Raman property must be requested.

NumericalDifferentiation

Type Block

Description Define options for numerical differentiations, that is the numerical calculation of gradients, Hessian and the stress tensor for periodic systems.

NuclearStepSize

Type Float

Default value 0.005

Unit Bohr

Description Step size for numerical nuclear gradient calculation.

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups**Type** Integer**GUI name** Number of groups**Description** Total number of processor groups. This is the number of tasks that will be executed in parallel.**nNodesPerGroup****Type** Integer**GUI name** Nodes per group**Description** Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.**StrainStepSize****Type** Float**Default value** 0.001**Description** Step size (relative) for numerical stress tensor calculation.**NumericalPhonons****Type** Block**Description** Configures details of a numerical phonons calculation.**AutomaticBZPath****Type** Bool**Default value** Yes**GUI name** Automatic BZ path**Description** If True, compute the phonon dispersion curve for the standard path through the Brillouin zone. If False, you must specify your custom path in the [BZPath] block.**BZPath****Type** Block**Description** If [NumericalPhonons%AutomaticBZPath] is false, the phonon dispersion curve will be computed for the user-defined path in the [BZPath] block. You should define the vertices of your path in fractional coordinates (with respect to the reciprocal lattice vectors) in the [Path] sub-block. If you want to make a jump in your path (i.e. have a discontinuous path), you need to specify a new [Path] sub-block.**Path****Type** Non-standard block**Recurring** True**Description** A section of a k space path. This block should contain multiple lines, and in each line you should specify one vertex of the path in fractional coordinates. Optionally, you can add text labels for your vertices at the end of each line.**BornEffCharge****Type** Float

Default value 0.0

Description Input option to give the Born effective charges of the species.

DielectricConst

Type Float

Default value 1.0

Description Input option to give the static dielectric constant of the species.

DoubleSided

Type Bool

Default value Yes

Description By default a two-sided (or quadratic) numerical differentiation of the nuclear gradients is used. Using a single-sided (or linear) numerical differentiation is computationally faster but much less accurate. Note: In older versions of the program only the single-sided option was available.

Interpolation

Type Integer

Default value 100

Description Use interpolation to generate smooth phonon plots.

NDosEnergies

Type Integer

Default value 1000

Description Nr. of energies used to calculate the phonon DOS used to integrate thermodynamic properties. For fast compute engines this may become time limiting and smaller values can be tried.

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

StepSize

Type Float

Default value 0.04

Unit Angstrom

Description Step size to be taken to obtain the force constants (second derivative) from the analytical gradients numerically.

SuperCell

Type Non-standard block

Description Used for the phonon run. The super lattice is expressed in the lattice vectors. Most people will find a diagonal matrix easiest to understand.

PESExploration

Type Block

Description Configures details of the automated PES exploration methods.

BasinHopping

Type Block

Description Configures the details of the Basin Hopping subtask.

DisplaceAtomsInRegion

Type String

Default value

Description If you specify a region name here, only the atoms belonging to this region will be displaced during the basin hopping procedure. For more details on regions, see the documentation on the System definition.

Displacement

Type Float

Default value 0.5

Unit Angstrom

Description Displacement in each degree of freedom.

PushApartDistance

Type Float

Default value 0.4

Unit Angstrom

Description Push atoms apart until no atoms are closer than this distance. This criterion is enforced for the initial structure and all those generated by random displacements.

Steps**Type** Integer**Default value** 20**Description** Number of displace & optimize Monte-Carlo steps to take.**BindingSites****Type** Block**Description** Options related to the calculation of binding sites.**AllowDisconnected****Type** Bool**Default value** No**Description** Allow disconnected binding sites.**Calculate****Type** Bool**Default value** No**Description** Calculate binding sites at the end of a job. Not needed for Binding Sites job.**DistanceDifference****Type** Float**Default value** -1.0**Unit** Angstrom**Description** If the distance between two mapped binding-sites is larger than this threshold, the binding-sites are considered different. If not specified, its value will set equal to [PE-SExploration%StructureComparison%DistanceDifference]**Debug****Type** Block**Description** ???.**DynamicSeedStates****Type** Bool**Default value** Yes**Description** Whether subsequent expeditions may start from states discovered by previous expeditions. This should lead to a more comprehensive exploration of the potential energy surface. Disabling this will focus the PES exploration around the initial seed states.**Dynamics****Type** Block**Description** ???.**Andersen****Type** Block**Description** ???.

Alpha

Type Float

Default value 1.0

Description ???.

CollisionPeriod

Type Float

Default value 100.0

Description ???.

Langevin

Type Block

Description ???.

Friction

Type Float

Default value 0.01

Description ???.

Nose

Type Block

Description ???.

Mass

Type Float

Default value 1.0

Description ???.

Thermostat

Type Multiple Choice

Default value none

Options [andersen, nose_hoover, langevin, none]

Description ???.

Time

Type Float

Default value 1000.0

Description ???.

TimeStep

Type Float

Default value 1.0

Description ???.

FiniteDifference

Type Float

Default value 0.01

Unit Angstrom

Description The finite difference distance to use for Dimer, Hessian, Lanczos, and optimization methods.

Hessian

Type Block

Description ???.

AtomList

Type String

Default value all

Description ???.

ZeroFreqValue

Type Float

Default value 1e-06

Description ???.

Job

Type Multiple Choice

Options [ProcessSearch, BasinHopping, SaddleSearch, LandscapeRefinement, BindingSites]

Description Specify the PES exploration job to perform.

LoadEnergyLandscape

Type Block

Description Options related to the loading of an Energy Landscape from a previous calculation.

KeepOnly

Type Integer List

GUI name List of states to keep

Description Upon loading the Energy Landscape, only keep the states specified here. The states should be specified via a list of integers referring to the indices of the states you want to keep.

Path

Type String

GUI name Load energy landscape from

Description The path to load an energy landscape from. Accepts either AMS result folders, or .con files in the native EON format (only available through the text input file).

Remove

Type Integer List

GUI name List of states to remove

Description Upon loading the Energy Landscape, remove (i.e. do not load) the states specified here. The states should be specified via a list of integers referring to the indices of the states you want to remove (i.e. the states you don't want to load).

SeedStates

Type Integer List

GUI name List of seed states

Description By default when you start a new PES Exploration from a loaded Energy Landscape, expeditions can start from any of the loaded minima. By using this input option, you can instruct the program to only use some of the states as 'expedition starting point'. The states that serve as 'expedition starting points' should be specified via a list of integers referring to the indices of the states.

NudgedElasticBand

Type Block

Description Options for the Nudged Elastic Band (NEB) method.

ClimbingImageMethod

Type Bool

Default value Yes

Description Use the climbing image algorithm to drive the highest image to the transition state.

ConvergedForce

Type Float

Default value -1.0

Unit eV/Angstrom

Description Convergence threshold for nuclear gradients. Note: Special value of -1.0 means using the same convergence criterion as the PES explorer's geometry optimizer.

Images

Type Integer

Default value 5

Description Number of NEB images between the two endpoints.

MaxIterations

Type Integer

Default value 500

Description Maximum number of NEB iterations.

OldTangent

Type Bool

Default value No

Description Use the old central difference tangent.

Spring

Type Float

Default value 5.0

Unit eV/Ang²

Description Spring force constant.

NumExpeditions

Type Integer

Default value 1

Description Sets the number of subsequent expeditions our job will consist of. Larger values result in a more comprehensive exploration of the potential energy surface, but will take more computational time.

NumExplorers

Type Integer

Default value 1

Description Sets the number of independent PES explorers dispatched as part of each expedition. Larger values will result in a more comprehensive exploration of the potential energy surface, but will take more computational time. By default an appropriate number of explorers are executed in parallel.

OptTSMethod

Type Multiple Choice

Default value SaddleSearch

Options [SaddleSearch, NudgedElasticBand]

Description When the full set of states in the energy landscape are optimized (see PESExploration%Job = GeometryOptimization), transition states can be optimized using either SaddleSearch or NudgedElasticBand methods. SaddleSearch uses information only from the current geometry of the TS; contrary, NudgedElasticBand ignores the current geometry and runs a Nudged-Elastic-Band calculation trying to connect the associated reactants and products if they are available.

Optimizer

Type Block

Description Configures the details of the geometry optimizers used by the PES explorers.

ConvergedForce

Type Float

Default value 0.005

Unit eV/Angstrom

Description Convergence threshold for nuclear gradients.

MaxIterations

Type Integer

Default value 400

Description Maximum number of iterations allowed for optimizations.

Method

Type Multiple Choice

Default value CG

Options [CG, QM, LBFGS, FIRE, SD]

Description Select the method for geometry optimizations.

Parallel

Type Block

Description Options for double parallelization, which allows to split the available processor cores into groups working through all the available tasks in parallel, resulting in a better parallel performance. The keys in this block determine how to split the available processor cores into groups working in parallel.

nCoresPerGroup

Type Integer

GUI name Cores per group

Description Number of cores in each working group.

nGroups

Type Integer

GUI name Number of groups

Description Total number of processor groups. This is the number of tasks that will be executed in parallel.

nNodesPerGroup

Type Integer

GUI name Nodes per group

Description Number of nodes in each group. This option should only be used on homogeneous compute clusters, where all used compute nodes have the same number of processor cores.

ParallelReplica

Type Block

Description ???.

DephaseLoopMax

Type Integer

Default value 5

Description ???.

DephaseLoopStop

Type Bool

Default value No

Description ???.

DephaseTime

Type Float

Default value 1000.0

Description ???.

PostTransitionTime

Type Float

Default value 1000.0

Description ???.

RefineTransition

Type Bool

Default value Yes

Description ???.

StateCheckInterval

Type Float

Default value 1000.0

Description ???.

StateSaveInterval

Type Float

Default value -1.0

Description ???.

StopAfterTransition

Type Bool

Default value No

Description ???.

Prefactor

Type Block

Description ???.

DefaultValue

Type Float

Default value 1000000000000.0

Description ???.

ProcessSearch

Type Block

Description Input options specific to the process search procedure.

MinimizationOffset

Type Float

Default value 0.2

Description After a saddle is found, images are placed on either side of the saddle along the mode and minimized to ensure that the saddle is connected to the original minimum and to locate the product state. `MinimizationOffset` is the distance those images are displaced from the saddle.

SaddleSearch

Type Block

Description Configuration for the Saddle Search procedure (used in `SaddleSearch` and `ProcessSearchJobs`).

DisplaceAtomsInRegion

Type String

Default value

Description A string corresponding to the name of a region. When performing the initial random displacement, only displace atoms in the specified region. This can help the Saddle Search procedure to start off in a desired region of the PES.

DisplaceMagnitude

Type Float

Default value 0.1

Description The standard deviation of the Gaussian displacement in each degree of freedom for the selected atoms.

MaxEnergy

Type Float

Default value 20.0

Unit eV

Description The energy (relative to the starting point of the saddle search) at which a saddle search explorer considers the search bad and terminates it.

MaxIterations

Type Integer

Default value 400

Description Maximum number of iterations for each saddle search run.

MinModeMethod

Type Multiple Choice

Default value dimer

Options [dimer, lanczos]

Description The minimum-mode following method to use.

StatesAlignment

Type Block

Description Configures details of how the energy landscape configurations are aligned respect to the main chemical system [System].

DistanceDifference

Type Float

Default value -1.0

Unit Angstrom

Description If the distance between two mapped atoms is larger than this threshold, the configuration is considered not aligned. If not specified, its value will set equal to [PESExploration%StructureComparison%DistanceDifference]

ReferenceRegion

Type String

Default value

Description Defines the region that is considered as the reference for alignments. Atoms outside this region are ignored in the alignments. TODO: Description of which PES exploration jobs actually use this ...

StructureComparison

Type Block

Description Settings for structure comparison.

CheckRotation

Type Bool

Description Rotates the system optimally before comparing structures. The default is to do this only for molecular systems when there are no fixed atom constraints.

DistanceDifference

Type Float

Default value 0.1

Unit Angstrom

Description If the distance between two mapped atoms is larger than this threshold, the two configurations are considered different structures.

EnergyDifference

Type Float

Default value 0.01

Unit eV

Description If the energy difference between two configurations is larger than this threshold, the two configurations are considered to be different structures.

IndistinguishableAtoms

Type Bool

Default value Yes

Description If yes, the order of the atoms does not affect the structural comparison. Atoms of the same element are then indistinguishable.

NeighborCutoff

Type Float

Default value 3.3

Unit Angstrom

Description Atoms within this distance of each other are considered neighbors.

RemoveTranslation

Type Bool

Description Translates the system optimally before comparing structures. The default is to do this only when there are no fixed atom constraints.

Temperature

Type Float

Default value 300.0

Unit Kelvin

Description The temperature that the job will run at. This may be used in different ways depending on the job, e.g. acceptance probabilities for Monte-Carlo based jobs, thermostating for dynamics based jobs, kinetic prefactors for jobs that find transition states. Some jobs may not use this temperature at all.

WriteHistory

Type Multiple Choice

Default value Converged

Options [None, Converged, All]

Description When to write the molecular geometry (and possibly other properties) to the history on the ams.rkf file. The default is to only write the converged geometries to the history. Changed to write either all frames or not frames at all to the history. Note that for parallel calculations, only the first group of processes writes to ams.rkf.

PESPointCharacter

Type Block

Description Options for the characterization of PES points.

Displacement

Type Float

Default value 0.04

Description Controls the size of the displacements used for numerical differentiation: The displaced geometries are calculated by taking the original coordinates and adding the mass-weighted mode times the reduced mass of the mode times the value of this keyword.

NegativeFrequenciesTolerance

Type Float

Default value -10.0

Unit cm-1

Description The threshold in frequency below which a mode is considered imaginary, i.e. indicating a transition state. This is a small negative number, as very small negative frequencies are normally due to numerical noise on an essentially flat PES and do not indicate true transition states.

NumberOfModes

Type Integer

Default value 2

Description The number of (lowest) eigenvalues that should be checked.

Tolerance

Type Float

Default value 0.016

Description Convergence tolerance for residual in iterative Davidson diagonalization.

PESScan

Type Block

Description Configures the details of the potential energy surface scanning task.

CalcPropertiesAtPESPoints

Type Bool

Default value No

Description Whether to perform an additional calculation with properties on all the sampled points of the PES. If this option is enabled AMS will produce a separate engine output file for every sampled PES point.

FillUnconvergedGaps

Type Bool

Default value Yes

Description After the initial pass over the PES, restart the unconverged points from converged neighboring points.

ScanCoordinate

Type Block

Recurring True

Description Specifies a coordinate along which the potential energy surface is scanned. If this block contains multiple entries, these coordinates will be varied and scanned together as if they were one.

Angle

Type String

Recurring True

Description Scan the angle between three atoms. Three atom indices followed by two real numbers delimiting the transit range in degrees.

Coordinate

Type String

Recurring True

Description Scan a particular coordinate of an atom. Atom index followed by (xlylz) followed by two real numbers delimiting the transit range.

DifDist

Type String

Recurring True

Description Scan the difference distance between two pairs of atoms, R(12)-R(34). Four atom indices followed by two real numbers delimiting the transit range in Angstrom.

Dihedral

Type String

Recurring True

Description Scan the dihedral angle between four atoms. Four atom indices followed by two real numbers delimiting the transit angle in degrees.

Distance

Type String

Recurring True

Description Scan the distance between two atoms. Two atom indices followed by two real numbers delimiting the transit distance in Angstrom.

SumDist

Type String

Recurring True

Description Scan the sum of distances between two pairs of atoms, R(12)+R(34). Four atom indices followed by two real numbers delimiting the transit range in Angstrom.

nPoints

Type Integer

Default value 10

Description The number of points along the scanned coordinate. Must be greater or equal 2.

Print

Type Block

Description This block controls the printing of additional information to stdout.

Timers

Type Multiple Choice

Default value None

Options [None, Normal, Detail, TooMuchDetail]

Description Printing timing details to see how much time is spend in which part of the code.

Properties

Type Block

Description Configures which AMS level properties to calculate for SinglePoint calculations or other important geometries (e.g. at the end of an optimization).

BondOrders

Type Bool

Default value No

Description Requests the engine to calculate bond orders. For MM engines these might just be the defined bond orders that go into the force-field, while for QM engines, this might trigger a bond order analysis based on the electronic structure.

Charges

Type Bool

Default value No

Description Requests the engine to calculate the atomic charges.

DipoleGradients

Type Bool

Default value No

Description Requests the engine to calculate the nuclear gradients of the electric dipole moment of the molecule. This can only be requested for non-periodic systems.

DipoleMoment

Type Bool

Default value No

Description Requests the engine to calculate the electric dipole moment of the molecule. This can only be requested for non-periodic systems.

ElasticTensor

Type Bool

Default value No

Description Calculate the elastic tensor.

Gradients

Type Bool

Default value No

GUI name Nuclear gradients

Description Calculate the nuclear gradients.

Hessian

Type Bool

Default value No

Description Whether or not to calculate the Hessian.

Molecules

Type Bool

Default value No

Description Requests an analysis of the molecular components of a system, based on the bond orders calculated by the engine.

NormalModes

Type Bool

Default value No

GUI name Frequencies

Description Calculate the frequencies and normal modes of vibration, and for molecules also the corresponding IR intensities if the engine supports the calculation of dipole moments.

Other

Type Bool

Default value Yes

Description Other (engine specific) properties. Details are configured in the engine block.

PESPointCharacter

Type Bool

Default value No

GUI name Characterize PES point

Description Determine whether the sampled PES point is a minimum or saddle point. Note that for large systems this does not entail the calculation of the full Hessian and can therefore be used to quickly confirm the success of a geometry optimization or transition state search.

Phonons

Type Bool

Default value No

Description Calculate the phonons (for periodic systems).

Polarizability

Type Bool

Default value No

Description Requests the engine to calculate the polarizability tensor of the system.

Raman

Type Bool

Default value No

Description Requests calculation of Raman intensities for vibrational normal modes.

SelectedRegionForHessian

Type String

GUI name Hessian only for

Description Compute the Hessian matrix elements only for the atoms in a particular region. If not specified, the Hessian will be computed for all atoms.

StressTensor

Type Bool

Default value No

GUI name Stress tensor

Description Calculate the stress tensor.

VCD

Type Bool

Default value No

Description Requests calculation of VCD for vibrational normal modes.

VROA

Type Bool

Default value No

Description Requests calculation of VROA for vibrational normal modes.

Raman

Type Block

Description Configures details of the Raman or VROA calculation.

FreqRange

Type Float List

Unit cm-1

Recurring True

GUI name Frequency range

Description Specifies a frequency range within which all modes will be scanned. 2 numbers: an upper and a lower bound.

IncidentFrequency

Type Float

Default value 0.0

Unit eV

Description Frequency of incident light.

LifeTime

Type Float

Default value 0.0

Unit hartree

Description Specify the resonance peak width (damping) in Hartree units. Typically the lifetime of the excited states is approximated with a common phenomenological damping parameter. Values are best obtained by fitting absorption data for the molecule, however, the values do not vary a lot between similar molecules, so it is not hard to estimate values. A typical value is 0.004 Hartree.

Restraints

Type Block

Description The Restraints block allows to add soft constraints to the system. A restraint is a potential energy function (a spring) attached to a certain coordinate, for example, an interatomic distance, with its minimum at the specified optimal value. A restraint is defined using one or two parameters: the ForceConstant and, for some types, the F(Inf) value. The ForceConstant parameter corresponds to second derivative of the restraint potential energy $d^2V(x)/dx^2$ for any x (harmonic restraints) or only at $x=0$ (other restraints). Here, x is a deviation from the restraint's optimal value.

Angle**Type** String**Recurring** True

Description Specify three atom indices *i j k* followed by an angle in degrees and, optionally, by the ForceConstant (default is 0.3 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the *i-j-k* angle at the given value. For periodic systems this restraint follows the minimum image convention.

DifDist**Type** String**Recurring** True

Description Specify four atom indices *i j k l* followed by the distance in Angstrom and, optionally, by the ForceConstant (default is 1.0 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the difference $R(ij)-R(kl)$ at the given value. For periodic systems this restraint follows the minimum image convention.

Dihedral**Type** String**Recurring** True

Description Specify four atom indices *i j k l* followed by an angle in degrees and, optionally, by the ForceConstant (default is 0.1 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the *i-j-k-l* dihedral angle at the given value. For periodic systems this restraint follows the minimum image convention.

Distance**Type** String**Recurring** True

Description Specify two atom indices followed by the distance in Angstrom and, optionally, by the ForceConstant (default is 1.0 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the distance between the two specified atoms at the given value. For periodic systems this restraint follows the minimum image convention.

FInfinity**Type** Float**Default value** 1.0**GUI name** Default F(inf)

Description Specify the default asymptotic value for the restraint force for the Hyperbolic and Erf profiles, in Hartree/Bohr or Hartree/radian. A per-restraint value can be specified after the profile type on the corresponding restraint line.

Profile**Type** Multiple Choice**Default value** Harmonic**Options** [Harmonic, Hyperbolic, Erf]**GUI name** Default restraint profile

Description Select the default type of restraint profile. The harmonic profile is most suitable for geometry optimizations but may result in very large forces that can be problematic in molecular dynamics. For MD simulations the Hyperbolic or Erf may be more suitable because the restraint force is bounded by a user-defined value. A per-restraint profile type can be specified after the ForceConstant value on the corresponding restraint line.

SumDist

Type String

Recurring True

Description Specify four atom indices i j k l followed by the distance in Angstrom and, optionally, by the ForceConstant (default is 1.0 in a.u.), profile type and F(Inf) (in a.u.). This restraint will try to keep the sum $R(ij)+R(kl)$ at the given value. For periodic systems this restraint follows the minimum image convention.

RigidMotions

Type Block

Description Specify which rigid motions of the total system are allowed. An external field is not considered part of the system. Normally the automatic option is doing what you want. However this feature can be used as a means of geometry constraint.

AllowRotations

Type Multiple Choice

Default value Auto

Options [Auto, None, All, X, Y, Z, XY, XZ, YZ]

Description Which overall rotations of the system are allowed

AllowTranslations

Type Multiple Choice

Default value Auto

Options [Auto, None, All, X, Y, Z, XY, XZ, YZ]

Description Which overall translations of the system are allowed

Tolerance

Type Float

Default value 1e-06

Description Tolerance for detecting linear molecules. A large value means larger deviation from linearity is permitted.

RNGSeed

Type Integer List

Description Initial seed for the (pseudo)random number generator. This should be omitted in most calculations to avoid introducing bias into the results. If this is unset, the generator will be seeded randomly from external sources of entropy. If you want to exactly reproduce an older calculation, set this to the numbers printed in its output.

SCMatrix

Type Block

Description Technical settings for programs using the AMT matrix system. Currently this is only used by DFTB

DistributedMatrix

Type Block

Description Technical settings for Distributed matrices

ColBlockSize

Type Integer

Default value 64

Description See comment of RowBlockSize.

RowBlockSize

Type Integer

Default value 64

Description The matrix is divided into blocks of size RowBlockSize x ColBlockSize. The smaller the blocks the better the distribution, but at the expense of increased communication overhead

Type

Type Multiple Choice

Default value Elpa

Options [Auto, Reference, ScaLapack, Elpa]

Description Determines which implementation is used to support the AbstractMatrixType.

Symmetry

Type Block

Description Specifying details about the details of symmetry detection and usage.

SymmetrizeTolerance

Type Float

Default value 0.05

Description Tolerance used to detect symmetry in case symmetrize is requested.

Tolerance

Type Float

Default value 1e-07

Description Tolerance used to detect symmetry in the system.

System

Type Block

Recurring True

Description Specification of the chemical system. For some applications more than one system may be present in the input. In this case, all systems except one must have a non-empty string ID specified after the System keyword. The system without an ID is considered the main one.

AllowCloseAtoms

Type Bool

Default value No

Description If AllowCloseAtoms is set to False, the AMS driver will stop with an error if it detects almost-coinciding atomic coordinates. If set to True, the AMS driver will try to carry on with the calculation.

Atoms

Type Non-standard block

Description The atom types and coordinates. Unit can be specified in the header. Default unit is Angstrom.

BondOrders

Type Non-standard block

Description Defined bond orders. May be used by MM engines.

Charge

Type Float

Default value 0.0

GUI name Total charge

Description The system's total charge in atomic units.

ElectrostaticEmbedding

Type Block

Description Container for electrostatic embedding options, which can be combined.

ElectricField

Type Float List

Unit V/Angstrom

Description External homogeneous electric field with three Cartesian components: e_x , e_y , e_z , the default unit being V/. In atomic units: $\text{Hartree}/(e \text{ bohr}) = 27.211 \text{ V/bohr}$; the relation to SI units is: $1 \text{ Hartree}/(e \text{ bohr}) = 5.14 \dots \times 10^{11} \text{ V/m}$. Supported by the engines `adf`, `band`, `dftb` and `mopac`. For periodic systems the field may only have nonzero components orthogonal to the direction(s) of periodicity (i.e. for 1D periodic system the x-component of the electric field should be zero, while for 2D periodic systems both the x and y components should be zero. This options cannot be used for 3D periodic systems.

MultipolePotential

Type Block

Description External point charges (and dipoles).

ChargeModel

Type Multiple Choice

Default value Point

Options [Point, Gaussian]

Description The charges may be represented as simple points (with a singular potential at the charge location) or may be represented by a spherical Gaussian distribution.

ChargeWidth**Type** Float**Default value** -1.0**Description** A width parameter in a.u. in case a Gaussian charge model is chosen. A negative value means that the width will be chosen automatically.**Coordinates****Type** Non-standard block**Description** Positions (in) and values of multipole charges, one per line. Each line describes a single point charge like: x y z q, or x y z q py pz px. Here x, y, z are the coordinates, q is the charge (in atomic units of charge) and py, pz, px are the (optional) dipole components (in atomic units, i.e. e/Bohr). Periodic systems are not supported.**FractionalCoords****Type** Bool**Default value** No**Description** Whether the atomic coordinates in the Atoms block are given in fractional coordinates of the lattice vectors. Requires the presence of the Lattice block.**GeometryFile****Type** String**Description** Read the geometry from a file (instead of from Atoms and Lattice blocks). Supported formats: .xyz**GuessBonds****Type** Bool**Default value** No**Description** Whether or not UFF bonds should be guessed.**Lattice****Type** Non-standard block**Description** Up to three lattice vectors. Unit can be specified in the header. Default unit is Angstrom.**LatticeStrain****Type** Float List**Description** Deform the input system by the specified strain. The strain elements are in Voigt notation, so one should specify 6 numbers for 3D periodic system (order: xx,yy,zz,yz,xz,xy), 3 numbers for 2D periodic systems (order: xx,yy,xy) or 1 number for 1D periodic systems.**LoadForceFieldAtomTypes****Type** Block**Description** This is a mechanism to set the ForceField.Type attribute in the input. This information is currently only used by the ForceField engine.**File****Type** String

Description Name of the (kf) file. It needs to be the result of a forcefield calculation.

LoadForceFieldCharges

Type Block

Recurring True

Description This is a mechanism to set the ForceField.Charge attribute in the input. This information is currently only used by the ForceField engine.

CheckGeometryRMSD

Type Bool

Default value No

Description Whether the geometry RMSD test should be performed, see MaxGeometryRMSD. Otherwise only basic tests are performed, such as number and atom types. Not doing the RMSD test allows you to load molecular charges in a periodic system.

File

Type String

Description Name of the (kf) file

MaxGeometryRMSD

Type Float

Default value 0.1

Unit Angstrom

Description The geometry of the charge producing calculation is compared to the one of the region, and need to be the same within this tolerance.

Region

Type String

Default value *

Description Region for which the charges should be loaded

Section

Type String

Default value AMSResults

Description Section name of the kf file

Variable

Type String

Default value Charges

Description Variable name of the kf file

MapAtomsToUnitCell

Type Bool

Default value No

Description For periodic systems the atoms will be moved to the central cell.

PerturbCoordinates**Type** Float**Default value** 0.0**Unit** Angstrom

Description Perturb the atomic coordinates by adding random numbers between [-PerturbCoordinates,PerturbCoordinates] to each Cartesian component. This can be useful if you want to break the symmetry of your system (e.g. for a geometry optimization).

PerturbLattice**Type** Float**Default value** 0.0

Description Perturb the lattice vectors by applying random strain with matrix elements between [-PerturbLattice,PerturbLattice]. This can be useful if you want to deviate from an ideal symmetric geometry, for example if you look for a phase change due to high pressure.

RandomizeAtomOrder**Type** Bool**Default value** No

Description Whether or not the order of the atoms should be randomly changed. Intended for some technical testing purposes only. Does not work with bond information.

ShiftCoordinates**Type** Float List**Unit** Bohr

Description Translate the atoms by the specified shift (three numbers).

SuperCell**Type** Integer List

Description Create a supercell of the input system (only possible for periodic systems). The integer numbers represent the diagonal elements of the supercell transformation; you should specify as many numbers as lattice vectors (i.e. 1 number for 1D, 2 numbers for 2D and 3 numbers for 3D periodic systems).

SuperCellTrafo**Type** Integer List

Description Create a supercell of the input system (only possible for periodic systems) $\vec{a}'_i = \sum_j T_{ij} \vec{a}_j$. The integer numbers represent the supercell transformation T_{ij} : 1 number for 1D PBC, 4 numbers for 2D PBC corresponding to a 2x2 matrix (order: (1,1),(1,2),(2,1),(2,2)) and 9 numbers for 3D PBC corresponding to a 3x3 matrix (order: (1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)).

Symmetrize**Type** Bool**Default value** No

Description Whether to symmetrize the input structure. This might also rototranslate the structure into a standard orientation. This will symmetrize the atomic coordinates to machine

precision. Useful if the system is almost symmetric or to rototranslate a symmetric molecule into a standard orientation.

Symmetry

Type Multiple Choice

Default value AUTO

Options [AUTO, NOSYM, C(LIN), D(LIN), C(I), C(S), C(2), C(3), C(4), C(5), C(6), C(7), C(8), C(2V), C(3V), C(4V), C(5V), C(6V), C(7V), C(8V), C(2H), C(3H), C(4H), C(5H), C(6H), C(7H), C(8H), D(2), D(3), D(4), D(5), D(6), D(7), D(8), D(2D), D(3D), D(4D), D(5D), D(6D), D(7D), D(8D), D(2H), D(3H), D(4H), D(5H), D(6H), D(7H), D(8H), I, I(H), O, O(H), T, T(D), T(H), S(4), S(6), S(8)]

Description Use (sub)symmetry with this Schoenflies symbol. Can only be used for molecules. Orientation should be correct for the (sub)symmetry. If used icw Symmetrize, the symmetrization will not reorient the molecule.

Task

Type Multiple Choice

Options [SinglePoint, GeometryOptimization, TransitionStateSearch, IRC, PESScan, NEB, VibrationalAnalysis, MolecularDynamics, GCMC, PESExploration]

Description Specify the computational task to perform: • Single Point: keep geometry as is • Geometry Optimization: optimize the geometry • Transition State: search for the transition state • IRC: intrinsic reaction coordinate • PES Scan: scan the potential energy surface • NEB: Nudged elastic band for reaction path optimization • Vibrational Analysis: perform one of the analysis types selected on the options page • Molecular Dynamics: perform MD simulation • GCMC: Grand Canonical Monte Carlo simulation • PES Exploration: automated potential energy surface exploration

Thermo

Type Block

Description Options for thermodynamic properties (assuming an ideal gas). The properties are computed for all specified temperatures.

LowFrequencyCorrector

Type Block

Description Options for the dampener-powered free rotor interpolator that corrects thermodynamic quantities for low frequencies. See DOI:10.1021/jp509921r and DOI:10.1002/chem.201200497.

Alpha

Type Float

Default value 4.0

Description The exponent term used in the dampener.

Frequency

Type Float

Default value 100.0

Unit cm⁻¹

Description The frequency around which the dampener interpolates between harmonic oscillator and free rotor quantities.

MomentOfInertia

Type Float

Default value 1e-44

Unit kg m²

GUI name Averaging Moment of Inertia

Description The moment of inertia used to restrict entropy results for very small frequencies (generally around less than 1 cm⁻¹).

Pressure

Type Float

Default value 1.0

Unit atm

Description The pressure at which the thermodynamic properties are computed.

Temperatures

Type Float List

Default value [298.15]

Unit Kelvin

Description List of temperatures at which the thermodynamic properties will be calculated.

TransitionStateSearch

Type Block

Description Configures some details of the transition state search.

ModeToFollow

Type Integer

Default value 1

Description In case of Transition State Search, here you can specify the index of the normal mode to follow (1 is the mode with the lowest frequency).

ReactionCoordinate

Type Block

Description Specify components of the transition state reaction coordinate (TSRC) as a linear combination of internal coordinates (distances or angles).

Angle

Type String

Recurring True

Description The TSRC contains the valence angle between the given atoms. Three atom indices followed by the weight.

Dihedral

Type String

Recurring True

Description The TSRC contains the dihedral angle between the given atoms. Four atom indices followed by the weight.

Distance

Type String

Recurring True

Description The TSRC contains the distance between the given atoms. Two atom indices followed by the weight.

UseSymmetry

Type Bool

Default value Yes

Description Whether to use the system's symmetry in AMS. Symmetry is recognized within a tolerance as given in the Symmetry key.

VibrationalAnalysis

Type Block

Description Input data for all vibrational analysis utilities in the AMS driver.

AbsorptionSpectrum

Type Block

Description Settings related to the integration of the spectrum for vibronic tasks.

AbsorptionRange

Type Float List

Default value [-200.0, 4000.0]

Unit cm-1

Recurring True

Description Specifies frequency range of the vibronic absorption spectrum to compute. 2 numbers: an upper and a lower bound.

FrequencyGridPoints

Type Integer

Default value 400

Description Number of grid points to use for the spectrum

LineWidth

Type Float

Default value 200.0

Unit cm-1

Description Lorentzian line-width.

SpectrumOffset

Type Multiple Choice

Default value relative

Options [absolute, relative]

Description Specifies whether provided frequency range are absolute frequencies or frequencies relative to computed 0-0 excitation energy.

Displacement

Type Float

Description Step size for finite difference calculations.

ExcitationSettings

Type Block

Description Block that contains settings related to the excitation for vibronic tasks.

EnergyInline

Type Float

Unit hartree

Description Vertical excitation energy, used when [ExcitationInfo] = [Inline].

ExcitationFile

Type String

Description Path to a .rkf/.t21 file containing the excited state information (gradients, transition dipoles and energies).

ExcitationInputFormat

Type Multiple Choice

Default value File

Options [File, Inline]

Description Select how the application should retrieve the excited state information (energy, gradient).

GradientInline

Type Non-standard block

Description Excited state gradient at ground state equilibrium geometry, used when [ExcitationInfo] = [Inline].

Singlet

Type Non-standard block

Description Symmetry labels + integer indices of desired singlet transitions (VG-FC absorption spectra support only 1 at a time)

Triplet

Type Non-standard block

Description Symmetry labels + integer indices of desired triplet transitions (VG-FC absorption spectra support only 1 at a time)

ModeTracking

Type Block

Description Input data for Mode Tracking.

HessianGuess

Type Multiple Choice

Default value CalculateWithFastEngine

Options [Unit, File, CalculateWithFastEngine]

GUI name Guess Hessian

Description Sets how to obtain the guess for the Hessian used in the preconditioner (if one is to be used).

HessianInline

Type Non-standard block

Description Initial guess for the (non-mass-weighted) Hessian in a $3N \times 3N$ block, used when [HessianGuess] = [Inline].

HessianPath

Type String

Description Path to a .rkf file containing the initial guess for the Hessian, used when [HessianGuess] = [File]. It may also be the name of the results folder containing the engine file.

ToleranceForBasis

Type Float

Default value 0.0001

Description Convergence tolerance for the contribution of the newest basis vector to the tracked mode.

ToleranceForNorm

Type Float

Default value 0.0005

Description Convergence tolerance for residual RMS value.

ToleranceForResidual

Type Float

Default value 0.0005

Description Convergence tolerance for the maximum component of the residual vector.

ToleranceForSpectrum

Type Float

Default value 0.01

Description Convergence tolerance for the spectrum in Vibronic Structure Tracking.

TrackingMethod

Type Multiple Choice

Default value OverlapInitial

Options [OverlapInitial, DifferenceInitial, FreqInitial, IRInitial, OverlapPrevious, DifferencePrevious, FreqPrevious, IRPrevious, HighestFreq, HighestIR, LowestFreq, LowestResidual]

Description Set the tracking method that will be used. Vibronic Structure Tracking uses Largest Displacement.

UpdateMethod

Type Multiple Choice

Options [JD, D, I]

Description Chooses the method for expanding the Krylov subspace: (I) No preconditioner (VST default), (D) Davidson or (JD) vdVorst-Slejpen variant of Jacobi-Davidson (Mode tracking default).

NormalModes

Type Block

Description All input related to processing of normal modes. Not available for vibronic structure tracking (as no modes are required there).

MassWeightInlineMode

Type Bool

Default value Yes

Description MODE TRACKING ONLY: The supplied modes must be mass-weighted. This tells the program to mass-weight the supplied modes in case this has not yet been done. (True means the supplied modes will be mass-weighted by the program, e.g. the supplied modes are non-mass-weighted.)

ModeFile

Type String

Description Path to a .rkf or .t21 file containing the modes which are to be scanned. Which modes will be scanned is selected using the criteria from the [ModeSelect] block.) This key is optional for Resonance Raman and Vibronic Structure. These methods can also calculate the modes using the engine.

ModeInline

Type Non-standard block

Recurring True

Description MODE TRACKING ONLY: Coordinates of the mode which will be tracked in a N x 3 block (same as for atoms), used when [ModeInputFormat] = [Inline]. Rows must be ordered in the same way as in the [System%Atoms] block. Mode Tracking only.

ModeInputFormat

Type Multiple Choice

Default value File

Options [File, Inline, Hessian]

GUI name Tracked mode source

Description Set how the initial guesses for the modes are supplied. Only mode tracking supports the Inline and Hessian options.

ModeSelect**Type** Block**Description** Pick which modes to read from file.**DisplacementBound****Type** Float**Description** Vibronic Structure (Refinement), Resonance Raman: Select all modes with a dimensionless oscillator displacement greater than the specified value.**FreqAndIRRange****Type** Float List**Unit** cm-1 and km/mol**Recurring** True**Description** Specifies a combined frequency and IR intensity range within which all modes will be selected. First 2 numbers are the frequency range, last 2 numbers are the IR intensity range.**FreqRange****Type** Float List**Unit** cm-1**Recurring** True**Description** Specifies a frequency range within which all modes will be selected. 2 numbers: an upper and a lower bound. Calculating all modes higher than some frequency can be achieved by making the upper bound very large.**Full****Type** Bool**Default value** No**GUI name** All modes**Description** Select all modes. This only make sense for Mode Scanning calculations.**HighFreq****Type** Integer**GUI name** # High frequencies**Description** Select the N modes with the highest frequencies.**HighIR****Type** Integer**GUI name** # High IR**Description** Select the N modes with the largest IR intensities.**IRRange****Type** Float List**Unit** km/mol**Recurring** True

Description Specifies an IR intensity range within which all modes will be selected. 2 numbers: an upper and a lower bound.

ImFreq

Type Bool

Default value No

GUI name All imaginary frequencies

Description Select all modes with imaginary frequencies.

LargestDisplacement

Type Integer

Description Vibronic Structure (Refinement), Resonance Raman: Select the N modes with the largest VG-FC displacement.

LowFreq

Type Integer

GUI name # Low frequencies

Description Select the N modes with the lowest frequencies. Includes imaginary modes which are recorded with negative frequencies.

LowFreqNoIm

Type Integer

GUI name # Low positive frequencies

Description Select the N modes with the lowest non-negative frequencies. Imaginary modes have negative frequencies and are thus omitted here.

LowIR

Type Integer

GUI name # Low IR

Description Select the N modes with the smallest IR intensities.

ModeNumber

Type Integer List

GUI name Mode numbers

Description Indices of the modes to select.

ScanModes

Type Bool

Default value No

GUI name Scan after refining

Description Supported by: Mode Tracking, Mode Refinement, Vibronic Structure Refinement: If enabled an additional displacement will be performed along the new modes at the end of the calculation to obtain refined frequencies and IR intensities. Equivalent to running the output file of the mode tracking calculation through the AMS ModeScanning task.

ResonanceRaman

Type Block

Description Block that contains settings for the calculation of Resonance Raman calculations

IncidentFrequency

Type Float

Unit cm-1

Description Frequency of incident light. Also used to determine most important excitation in case more than one is provided.

LifeTime

Type Float

Default value 0.00045

Unit hartree

Description Lifetime of Raman excited state.

RamanOrder

Type Integer

Default value 2

Description Order up to which to compute Raman transitions

RamanRange

Type Float List

Default value [0.0, 2000.0]

Unit cm-1

Recurring True

Description Specifies frequency range of the Raman spectrum to compute. 2 numbers: an upper and a lower bound.

Type

Type Multiple Choice

Options [ModeScanning, ModeTracking, ModeRefinement, VibronicStructure, VibronicStructureTracking, VibronicStructureRefinement, ResonanceRaman]

Description Specifies the type of vibrational analysis that should be performed

VSTRestartFile

Type String

Description Path to a .rkf file containing restart information for VST.

A

Add molecules (*molecular dynamics*), 94
 Add-ons, 217
 ADF, 211
 Adiabatic Hessian Franck-Condon, 189
 AMS input file, 9
 ams.log, 16
 ams.rkf, 16
 AMS_JOBNAME, 18
 AMS_RESULTSDIR, 18
 AMS_SWITCH_LOGFILE_AND_STDOUT, 18
 AMSPipe, 22
 Applications, 40
 APT, 187
 Atom energies, 222
 Atomic charges, 206
 Atomic masses, 35
 Atomic polar tensor, 187
 Autocorrelation function, 233
 Automated PES exploration, 124
 Available engines, 211

B

BAND, 211
 Barostats, 86
 Basin Hopping, 135
 Binary output files, 16
 Binding Sites Detection, 138
 Block constraints, 47
 Block normal modes, 156
 Bond Boost method, 108
 Bond energy calculation, 41
 Bond orders, 208
 Bulk modulus, 147

C

Cell optimization, 46
 Charge, 35
 Compute clusters, 14
 Conjugate gradients (*geometry optimizer*), 56
 Constrained optimization, 46
 Constraints, 46

CREST metadynamics, 101
 CVHD, 103

D

D3 dispersion add-on, 218
 D4 dispersion add-on, 218
 D5 dispersion add-on, 218
 Developer options, 313
 DFTB, 211
 Diffusion coefficient, 237
 Dipole moment, 206
 Double group symmetry, 315
 Double parallelism, 18
 Driver level parallelism, 18

E

eHEX method (*NEMD*), 112
 Elastic properties, 147
 Elastic tensor, 147
 Electric field, 35
 Engine add-ons, 217
 Engine input, 211
 Engine output files, 17
 Engines, 209
 Enthalpy, 173
 Entropy, 173
 Environment variables, 17
 EON, 124
 EON Optimizer, 137
 Examples, 240
 Excited state optimizations, 76
 External engines, 213

F

fbMC, 110
 FCF, 189
 FCF module, 189
 FIRE (*geometry optimizer*), 51
 Fixed atoms, 47
 Fluorescence, 189
 Force bias Monte Carlo, 110
 Forces, 143

Fractional coordinates, 31
Franck–Condon factors, 188, 189
Free rotor interpolation corrections,
173

G

GCMC, 116
Geometry, 29
Geometry constraints, 46
Geometry convergence, 45
Geometry optimization, 43
Geometry optimization methods, 48
Geometry relaxation, 43
Gibbs free energy, 173
Grand canonical Monte Carlo, 116

H

Heat capacity, 173
Heat exchange (*molecular dynamics*), 112
Hessian, 144
HEX method (*NEMD*), 112
Histogram (*molecular dynamics*), 231

I

Imaginary modes, 155
IMDHO, 196
Independent mode displaced harmonic
oscillator, 196
Initial Hessian, 51
Input file syntax, 9
Input for AMS, 7
Interface to external programs, 213
Internal energy, 173
Intrinsic reaction coordinate, 70
IR frequencies, 153
IRC, 70
Irreducible representation, 315
Isotopes, 35

J

Job name, 18

L

L-BFGS (*geometry optimizer*), 55
Landscape Refinement, 137
Lattice, 29
Lattice constraints, 48
Lattice deformations, 91
Lattice optimization, 46
Lattice vectors, 30
Lattice vibrations, 177
Lennard–Jones potential, 217
Linear transit, 60
Logfile `ams.log`, 16

M

MBH, 156
MD, 76
MD trajectory analysis, 228
MEP, 70
Minimum energy profile, 70
MLPotential, 211
Mobile block Hessian, 156
Mode intensity tracking, 168
Mode refinement, 160
Mode scanning, 159
Mode selecting, 171
Mode tracking, 162
Molecular dynamics, 76
Molecular dynamics checkpoint, 89
Molecule detection, 208
Molecule gun (*molecular dynamics*), 94
Moments of inertia, 177
MOPAC, 211

N

NEB, 63
NEMD, 112
Non-isotropic stress, 221
Normal modes, 153
Nuclear gradients, 143
Nudged elastic band, 63

O

Output files, 16
Output of AMS, 16

P

Partial vibrational density of states,
177
Partial vibrational spectra, 177
PES point characterization, 144
PES point properties, 141
PES scan, 60
Phonons, 177
Phosphorescence, 189
Pipe interface, 22
Pipe protocol, 22
PLAMS, 22
PLUMED library, 101
Point charges, 35
Polarizability, 206
Pressure, 220
Pressure (*molecular dynamics*), 88
Process Search, 132
Properties, 8
PVDOS, 177
Python, 22

Q

Quasi-Newton (*geometry optimizer*), 49

R

Radial distribution function, 229
Raman, 181
Raman-active vibrations, 156
RDF, 229
ReaxFF, 211
Regions, 34
Remove molecules (*molecular dynamics*), 98
Required citations, 319
Resonance Raman, 182
Resonance VROA, 185
Restart (*geometry*), 37
Restart (*molecular dynamics*), 83
Restrains, 223
Results directory, 15, 16
Run types, 40
Running AMS, 12

S

Saddle Search, 133
Scan coordinate, 61
ScanFreq, 155
Schönflies symbol, 315
SCM_TMPDIR, 15
SCMGO (*geometry optimizer*), 54
Scratch directory, 15
Scripting, 22
Shear modulus, 147
Single point calculation, 41
Starting directory, 14
Strain constraints, 48
Stress tensor, 147
Structure relaxation, 43
Subspecies, 315
Super cell, 31
Symmetric displacements, 155
Symmetrization, 32
Symmetry, 32, 315
Symmetry label, 315

T

T-NEMD, 112
Task farming, 18
Tasks, 8, 40
Temperature (*molecular dynamics*), 88
Temporary directory, 15
Thermodynamics, 173
Thermostat, 84
Trajectory sampling, 88
Transition state search, 57

Two-level parallelism, 18

U

UFF, 211
Utilities, 226

V

VCD, 187
VCDtools module, 238
Vertical gradient Franck-Condon, 196
VG-FC, 196
Vibrational circular dichroism, 187
Vibrational Raman optical activity, 185
Vibrational spectroscopy, 151
Vibrationally resolved electronic spectra, 188
Vibronic-structure excited state, 188
Volume regimes, 91
VROA, 185

W

Wall potential, 225

X

XYZ file format, 313

Y

Young modulus, 147

Z

Z-matrix, 30