SCM
Software for Chemistry & Materials

AMS Manual

Amsterdam Modeling Suite 2019

www.scm.com

Apr 18, 2019

CONTENTS

CHAPTER

ONE

GENERAL

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 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 AMS2019 driver?

- *Intrinsic Reaction Coordinate (IRC) Scan* (page 28) in now available in the AMS driver for molecular and periodic systems.
- Support for the *Grand Canonical Monte Carlo (GCMC)* (page 79) method has been added in the AMS driver.
- Molecular composition analysis for molecular dynamics simulations (see tutorial)
- *Collective Variable-driven Hyperdynamics (CVHD)* (page 51)
- *Molecule gun* (page 45) and *molecule sink* (page 48) for molecular dynamics
- *PLUMED library support* (page 50) 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 91) (assuming an ideal gas) are automatically computed after normal modes calculations.
- *Partial vibrational density of states (PVDOS)* (page 93) 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* (page 57):
  1. *Mode Scanning* (page 58) (aka ADF's ScanFreq)
  2. *Mode Refinement* (page 62) (aka "Frequency range selection")
  3. *Mode Tracking* (page 66) (experimental)

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 inhomogenity 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 are 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 BAND or 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 101) 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 is a big reorganization of the entire suite, which is not complete yet. The long-term goal is to integrate all programs in the suite fully into the AMS driver, but as of the 2018 release, only BAND, DFTB, MOPAC and UFF have been fully integrated and removed as separate programs. ReaxFF is fully usable from within AMS, but not all features of the standalone ReaxFF program (e.g. the bond boost method) have been ported to AMS yet. Therefore, ReaxFF is in the 2018 release both available as an AMS engine and as the familiar standalone program. ADF can be used both through AMS and as the standalone program known from previous releases. QuantumEspresso has not yet been integrated into AMS. External programs can be hooked into the AMS driver through a *thin scripting layer* (page 101). While the transition to AMS is not complete yet, we believe that AMS in its current state already offers significant benefits over the 2017 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.

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 *Input syntax* (page 107) 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:

```
$ADFBIN/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
    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 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 99), 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 $ADFBIN/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 157) 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. Conceptually there are two different kinds of these binary files in the result folder:

- 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 contains 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.
- Additionally there might be a binary output file for every point on the potential energy surface that was visited during the calculation. They contain all information tied to a specific point on the potential energy surface. We call these files the engine output files, because they are not written by the AMS driver, but by the specific engine used for the calculation. As such they contain engine specific information (e.g. orbitals for quantum mechanical engines), which might be written in an engine specific format. The engine files are basically the same as the main output file the standalone programs produced for single point calculations prior to the 2018 release of the suite: The BAND engine writes engine output files that are basically the same as the `RUNKF` file that the BAND program wrote. The engine output files of the DFTB engine correspond to the `dftb.rkf` file that the DFTB program used to write. The 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, so by default the engine results are only kept for special points, e.g. the final geometry in a geometry optimization.

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 all properties documented in the *PES point properties section* (page 87) of this manual. If the information depends on the entire trajectory over the PES, it is found in the main `ams.rkf` written by the AMS driver.

CHAPTER

TWO

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 [True | False]
   GeometryFile string
   LatticeStrain float_list
   SuperCell integer_list
   AtomMasses # Non-standard block. See details.
      ...
   End
   Charge float
   BondOrders # Non-standard block. See details.
      ...
   End
End
```

2.1 System geometry

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 False

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
```

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
      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 `[` and `]`). 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 157) is supported.

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

`System`

`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). The integer numbers represent the supercell transformation; (i.e. 1 number for 1D, 4 numbers for 2D and 9 numbers for 3D periodic systems). The order is as you read the matrix (horizontal reading), unless you are used to vertical reading.

`RandomizeCoordinates`

Type Float

Default value 0.0

Unit Angstrom

Description Apply a random noise to the atomic coordinates. This can be useful if you want to deviate from an ideal symmetric geometry.

RandomizeStrain

Type Float

Default value 0.0

Description Apply a random strain to the system. 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.

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 14) block. Note that the randomization of the coordinates is applied after a potential supercell creation.

2.2 Additional system properties

AMS allows to set user-defined masses for particular atoms. This can be used to simulate isotopes of different atoms. Masses are specified by tagging the specific atoms in the `Atoms` block and then assigning them a custom mass (in unified atomic mass units) within the `AtomMasses` block. The following input shows the system specification for a heavy water molecule:

```
System
   Atoms
      O     0.0   0.0        0.59372
      H.d   0.0   0.76544   -0.00836
      H.d   0.0  -0.76544   -0.00836
   End
   AtomMasses
      H.d 2.014
   End
End
```

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

Description The system's total charge in atomic units (only for non-periodic systems).

BondOrders

Type Non-standard block

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

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

2.3 Restoring a system from disk

Instead of specifying the system to simulate in the `System` block of the input, it is also possible to restore 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.

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 4) the loaded geometry is just the one for which the engine was invoked when it wrote this file. For the *main result file* (page 4) `ams.rkf` written by the AMS driver, which geometry is loaded depends on the *task* (page 11) 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 11) 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.)

CHAPTER

THREE

EXPLORING THE PES: TASKS

3.1 Single point calculations

A single point calculation is the simplest task available in th AMS driver. It simply runs the *engine* (page 99) 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 87). 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 87) for an overview of which properties can be calculated with the `SinglePoint` task in AMS.

3.2 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 117) 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
   End
   MaxIterations integer
   CalcPropertiesOnlyIfConverged [True | False]
   OptimizeLattice [True | False]
   Pressure float
   KeepIntermediateResults [True | False]
End
```

`GeometryOptimization`

Type Block

Description Configures details of the geometry optimization and transition state searches.

`Convergence`

Type Block

Description Convergence is monitored for two items: the energy and the Cartesian gradients. Convergence criteria can be specified separately for each of these items.

`Energy`

Type Float

Default value 1e-05

Unit Hartree

Description The criterion for changes in the energy.

`Gradients`

Type Float

Default value 0.001

Unit Hartree/Angstrom

Description The criterion for changes in the gradients.

`Step`

Type Float

Default value 0.001

Unit Angstrom

Description The maximum Cartesian step allowed for a converged geometry.

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`.
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

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

`CalcPropertiesOnlyIfConverged`

Type Bool

Default value True

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.

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 87) 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.

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

GeometryOptimization

OptimizeLattice

Type Bool

Default value False

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

Pressure

Type Float

Default value 0.0

Description Optimize the structure under pressure (this will only have an effect if you are optimizing the lattice vectors). Currently only working in combination with the Quasi-Newton optimizer. For phase transitions you may consider disabling or breaking the symmetry.

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

GeometryOptimization

KeepIntermediateResults

Type Bool

Default value False

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. ...

3.2.1 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 120)

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

```
Constraints
   Atom integer
   Coordinate integer [x|y|z] float?
   Distance (integer){2} float
   Angle (integer){3} float
   Dihedral (integer){4} float
   BlockAtoms integer_list
   Block string
End
```

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

`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.

`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 blockName` Creates a block constraint (freezes all internal degrees of freedom) for a set of atoms identified by a tagging string `blockName` in the `System%Atoms` block. The tag is attached to element symbol and separated by a dot. Example:

```
System
   Atoms
      C.myblock  0.0  0.0  0.0
      C.myblock  0.0  0.0  1.0
      C          0.0  1.0  0.0
   End
End
Constraints
   Block myblock
End
```

Note that the coordinate, distance, angle, and dihedral constraints do **not** need to be satisfied at the beginning of the optimization.

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 these kinds 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.

3.2.2 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 | ConjugateGradients]
   CoordinateType [Auto | Delocalized | Cartesian]
End
```

`GeometryOptimization`

`Method`

Type Multiple Choice

Default value Auto

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

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), 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]

Description Select the type of coordinates in which to perform the optimization. If 'Auto', delocalized coordinates will be used for molecular systems, while Cartesian coordinates will be used for periodic systems. Optimization in delocalized coordinates [Delocalized] can only be used for geometry optimizations or transition state searches of molecular systems with the Quasi-Newton method. The experimental SCMGO optimizer supports [Delocalized] coordinates for both molecular and periodic systems.

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 compatibilty of the different optimizers and options:

Optimizer	Constraints	Lattice opt.	Coordinate types
Quasi-Newton	Yes	Yes	All (molecules)
			Cartesian (periodic)
FIRE	Fixed atoms and coordinates	Yes	Cartesian
SCMGO	No	Yes	Delocalized
Conjugate gradients	No	No	Cartesian

Furthermore for optimal performance the optimizer should be chosen with the speed of the engine in mind: Faster engine 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 approach [*1-3* (page 163)], using the Hessian for computing changes in the geometry so as to reach a local minimum. The Hessian itself is typically *approximated* (page 17) in the beginning and updated in the process of optimization. For molecules it uses delocalized coordinates by default, based mainly on the work by Marcel Swart [*4* (page 163)]. For periodic systems the optimization is performed in cartesian coordinates instead.

The Quasi-Newton optimizer supports all types of constraints and can be used for both molecular and periodic systems, including lattice optimizations. For molecular systems, where the optimization can be performed in delocalized coordinates, the number of steps taken to reach the local minimum is quite small. For large systems (on the order of hundreds of atoms) and fast compute *engines* (page 99), the overhead of the Quasi-Newton optimizer is likely the bottleneck of the calculation, and more light-weight optimizers like *FIRE* (page 18) will give an overall better performance. We do not recommend using the Quasi-Newton optimizer for systems >1000 atoms. Because of these properties the Quasi-Newton optimizer is the default in AMS for all kinds of optimizations of small and medium sized systems. It is also used as the optimizer backend for the *PES scan task* (page 24), the *transition state search* (page 23) as well as the calculation of the *elastic tensor* (page 93).

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

```
GeometryOptimization
   Quasi-Newton
      MaxGDIISVectors integer
      Step
         TrustRadius float
      End
   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.

The Quasi-Newton optimizer uses the Hessian to compute the next 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]
   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

Description KF file containing the initial Hessian. 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]

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).

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, see *this example* (page 117).

FIRE

The Fast Inertial Relaxation Engine [*5* (page 163)] 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 99). 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 is also supports *fixed atom constraints* (page 14) and *coordinate constraints* (page 14) (as long as the value of the constrained coordinate is already satisfied in the input geometry), as well as lattice optimizations.

FIRE is the default optimizer for systems >1000 atoms. For smaller systems and fast compute engines it is selected 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
      MapAtomsToUnitCell [True | False]
      NMin integer
      RejectEnergyIncrease [True | False]
      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 the the symbols used in the article describing the method, see PRL 97, 170201 (2006).

MapAtomsToUnitCell

Type Bool

Default value False

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.

RejectEnergyIncrease

Type Bool

Default value False

Description Makes the optimizer reject steps that increase the energy. This can speed up convergence, but often causes the optimizer to get stuck on small discontinuities on the potential energy surface. It is therefore disabled by default.

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.

Note that neither the energy change per step, nor the step size are reliable convergence criteria for the FIRE optimizer. Only the gradient convergence criterium (set with the `Converge%Gradients` key) is used by FIRE to determine when the optimization has converged.

SCMGO

The SCMGO optimizer is a new implementation of a Quasi-Newton style optimizer working in delocalized coordinates. In the 2019 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 16) optimizer, which can not use the more efficient delocalized coordinates for periodic systems.

```
GeometryOptimization
   SCMGO
      ContractPrimitives [True | False]
      NumericalBMatrix [True | False]
      Step
         TrustRadius float
         VariableTrustRadius [True | False]
      End
      logSCMGO [True | False]
      testSCMGO [True | False]
   End
End
```

`GeometryOptimization`

`SCMGO`

Type Block

Description Configures details SCMGO.

`ContractPrimitives`

Type Bool

Default value True

Description Form non-redundant linear combinations of primitive coordinates sharing the same central atom

NumericalBMatrix

Type Bool

Default value False

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 True

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

logSCMGO

Type Bool

Default value False

Description Verbose output of SCMGO internal data

testSCMGO

Type Bool

Default value False

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 17).

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 slower than *FIRE* (page 18), and supports neither lattice nor constrained optimizations. It is therefore never selected automatically, and we do not recommend using it.

```
GeometryOptimization
   ConjugateGradients
      Step
         MinRadius float
         TrustRadius float
      End
   End
End
```

GeometryOptimization

ConjugateGradients

Type Block

Description Configures details of the conjugate gradients geometry optimizer.

Step

Type Block

Description

MinRadius

Type Float

Default value 0.0

Description Minimum value for the trust radius.

TrustRadius

Type Float

Default value 0.2

Description Initial value of the trust radius.

3.2.3 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 99) 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 15) used. In these cases switching to another method could help with convergence problems. We recommend first trying the *FIRE* (page 18) 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 9) 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.

3.3 Transition state search

A transition state (TS) search is very much like a *geometry optimization* (page 11): 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 117) and the PES scan and transition state search tutorial

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 16) optimizer. As such all the settings and keywords described on the *geometry optimization manual page* (page 11) 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. 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
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).

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 intial geometry that is already close to the transition state. One can use a *potential energy surface scan* (page 24) 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 transitions 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 17) 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 lever of theory) and read that Hessian as the initial Hessian for the transition state search at a higher level of theory. This approach is demonstrated in the *Examples* (page 117) and the PES scan and transition state search tutorial.

3.4 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.

See also:

Examples (page 117) and the PES scan and transition state search tutorial

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}
   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 14) 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.

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 behaviour can be changed with the PESScan%CalcPropertiesAtPESPoints keyword:

PESScan

CalcPropertiesAtPESPoints

Type Bool

Default value False

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 87) selected in `Properties` block.

3.4.1 Troubleshooting

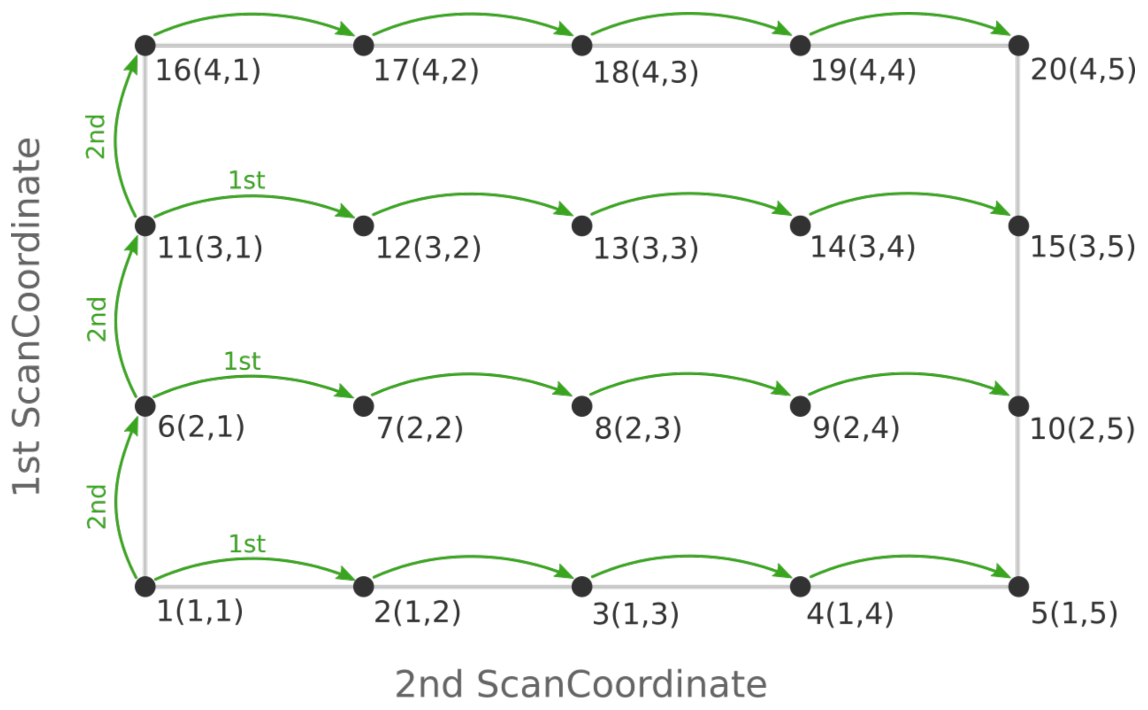
Technically all PES scan calculations are conducted as a series of geometry optimizations with constraints for the scanned coordinates, where the value of the constraint varies slowly through the scanned range. In this way every sampled point on the potential energy surface corresponds to a particular 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 11) in the AMS manual. Note that PES scans always use the *Quasi-Newton* (page 16) optimizer.

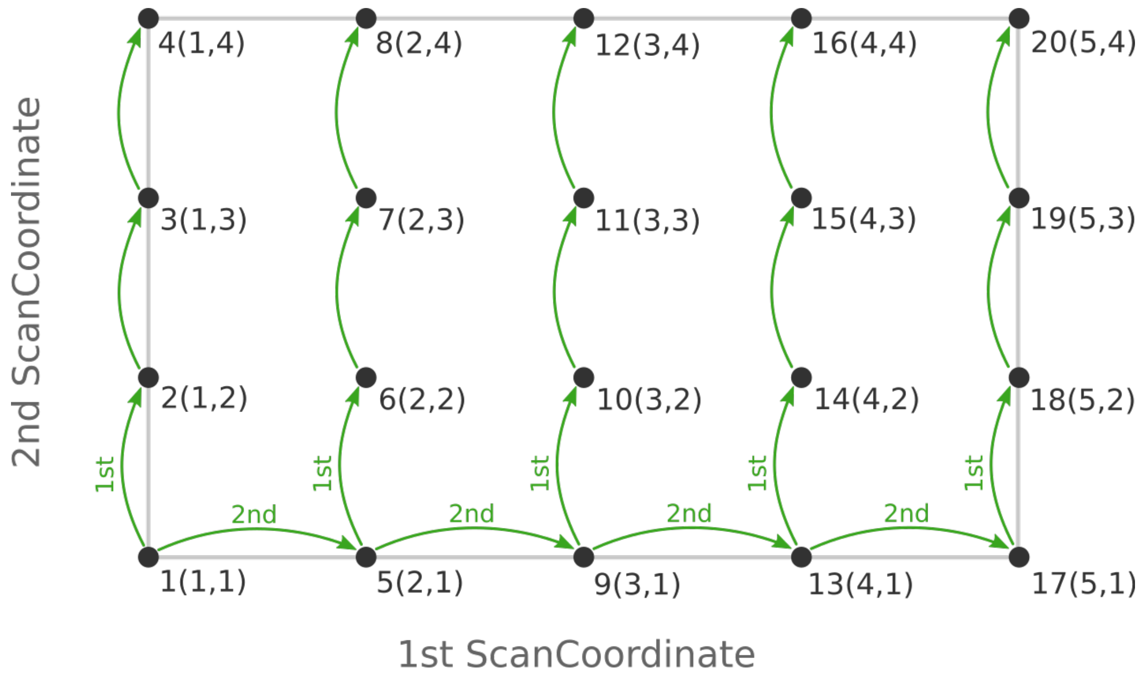
While tweaking the geometry optimizer's settings can sometimes help with convergence problems, these problems can also be easily caused by errors in the user input.

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`

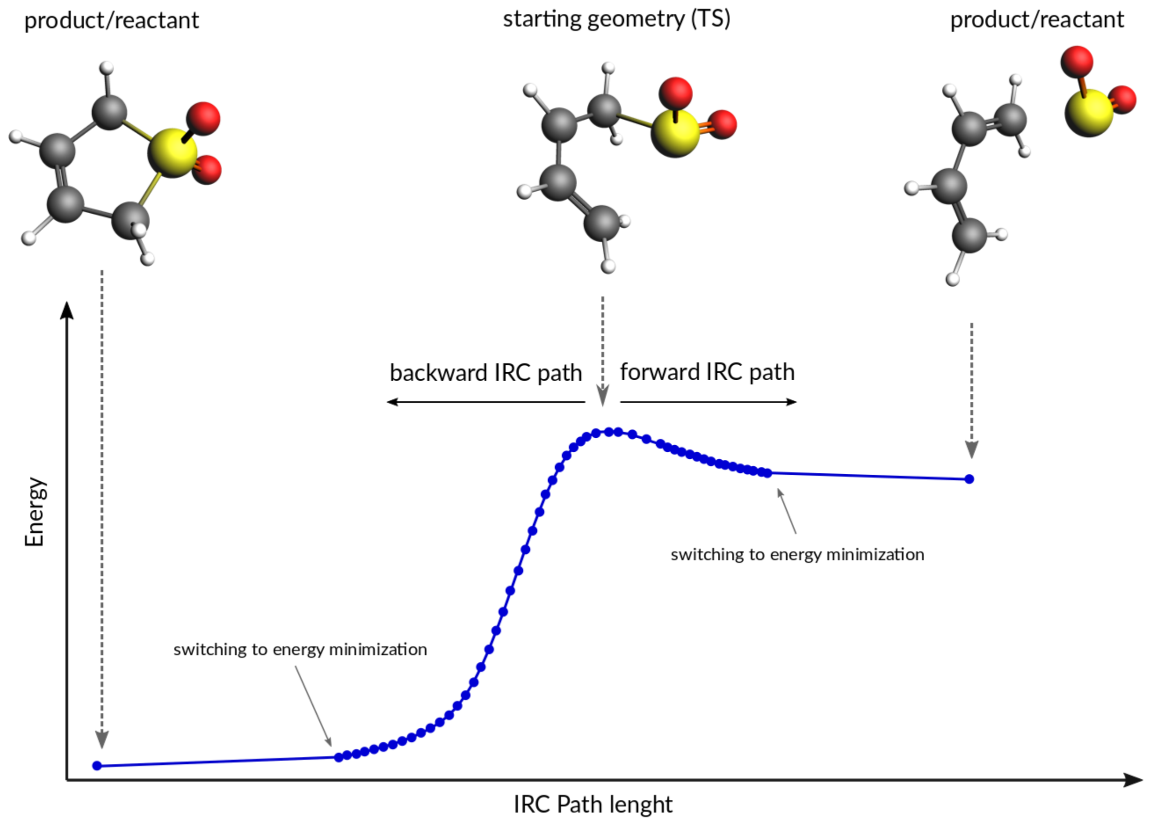
Type Bool

Default value True

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

3.5 Intrinsic Reaction Coordinate (IRC) Scan

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 [*15* (page 163), *16* (page 164)]. **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 coordinates [*17* (page 164)], 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.



See also:

Examples (page 117) and the PES scan, TS and IRC tutorial

3.5.1 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 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 (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 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 12) 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.

3.5.2 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 [True | False]
   MaxIRCSteps integer
   MaxIterations integer
   MaxPoints integer
   MinEnergyProfile [True | False]
   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

Description IRC step size in mass-weighted coordinates, sqrt(amu)*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

Description If 'Type' is set to 'FromFile' then in this key you should specifiy the RKF file containing the initial Hessian. 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]

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

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

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

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

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

`Step`

Type Float

Default value 0.001

Unit Angstrom

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]

Description Select the type of coordinates in which to perform the optimization. Note that the Delocalized option should be considered experimental. Besides, it is not possible to use delocalized coordinates for periodic systems.

`MinEnergyProfile`

Type Bool

Default value False

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

`KeepConvergedResults`

Type Bool

Default value True

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 re-compute 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 'Max-Points' 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

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.

3.5.3 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 ADFMovie 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 Fortrtan 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.

3.6 Molecular dynamics

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

See also:

Examples (page 117)

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
      Coords float_list
      CoordsBox float_list
      CoordsSigma float_list
      Energy float
      EnergySigma float
      FractionalCoords float_list
      FractionalCoordsBox float_list
      FractionalCoordsSigma float_list
      Frequency integer
      MinDistance float
      NumAttempts integer
      Rotate [True | False]
      StartStep integer
      StopStep integer
      System string
      Temperature float
      TemperatureSigma float
      Velocity float
      VelocityDirection float_list
      VelocitySigma float
   End
   Barostat
      BulkModulus float
      ConstantVolume [True | False]
      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
   BondOrderCutoff float
   CVHD
      Bias
         DampingTemp float
         Delta float
         Height float
      End
      ColVarBB
         at1 string
         at2 string
         cutoff float
         p integer
         rmax float
         rmin float
      End
      Frequency integer
      StartStep integer
      StopStep integer
      WaitSteps integer
   End
   CalcPressure [True | False]
   Checkpoint
      Frequency integer
   End
   HeatExchange
      HeatingRate float
      Method [Simple | HEX | eHEX]
      Sink
         Box
            Amax float
            Amin float
            Bmax float
            Bmin float
            Cmax float
            Cmin float
         End
         FirstAtom integer
         LastAtom integer
      End
      Source
         Box
            Amax float
            Amin float
            Bmax float
            Bmin float
            Cmax float
            Cmin float
         End
         FirstAtom integer
         LastAtom integer
      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
   PRD
      BondChange
         ChangeThreshold float
         DissociationThreshold float
         FormationThreshold float
      End
      CorrelatedSteps integer
      DephasingSteps integer
      MolCount
      nReplicas integer
   End
   Plumed
      Input # Non-standard block. See details.
         ...
      End
   End
   Preserve
      AngularMomentum [True | False]
      CenterOfMass [True | False]
      Momentum [True | False]
   End
   Print
      System [True | False]
      Velocities [True | False]
   End
   RemoveMolecules
      Formula string
      Frequency integer
      SafeBox
         Amax float
         Amin float
         Bmax float
         Bmin float
         Cmax float
         Cmin float
      End
      SinkBox
         Amax float
         Amin float
         Bmax float
         Bmin float
         Cmax float
         Cmin float
      End
      StartStep integer
      StopStep integer
   End
   ReplicaExchange
      SwapFrequency integer
      TemperatureFactor float_list
      nReplicas integer
   End
   Restart string
   Thermostat
      BerendsenApply [Local | Global]
      ChainLength integer
      Duration integer_list
      FirstAtom integer
      LastAtom integer
      Tau float
      Temperature float_list
      Type [None | Berendsen | NHC]
   End
   TimeStep float
   Trajectory
      SamplingFreq integer
      TProfileGridPoints integer
   End
End
```

3.6.1 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

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) 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 True

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

`CenterOfMass`

Type Bool

Default value False

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 True

Description Remove overall (linear) momentum of the system.

3.6.2 (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 *System definition* (page 5). 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 `moldyn.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

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. ADFinput will use the thermostat temperature as default.

`Type`

Type Multiple Choice

Default value Random

Options [Zero, Random, FromFile, Input]

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 ADFinput.

`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

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

3.6.3 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 `Tau`. 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 `Tau` 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 `Tau` should thus be set as large as possible to limit these artifacts while still providing sufficient temperature control. Common values of `Tau` 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 `Tau`. It is thus not well suited for systems starting far from equilibrium, because the oscillations may take long to settle. The time constant `Tau` 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 regions of the system at the same time. This is done by specifying the `Thermostat` block multiple times and setting the `FirstAtom` and/or `LastAtom` keys to the desired range of atoms. Care needs to be taken to avoid defining thermostats with overlapping atom ranges.

`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]

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

Description Number of individual thermostats forming the NHC thermostat

`Duration`

Type Integer List

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

`FirstAtom`

Type Integer

Default value 1

Description Index of the first atom to be thermostatted

`LastAtom`

Type Integer

Default value 0

Description Index of the last atom to be thermostatted. A value of zero means the last atom in the system.

`Tau`

Type Float

Unit Femtoseconds

Description The time constant of the thermostat.

`Temperature`

Type Float List

Unit Kelvin

Description The target temperature of the thermostat.

`Type`

Type Multiple Choice

Default value None

Options [None, Berendsen, NHC]

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 `Tau` for the barostat is usually at least several times higher than the corresponding `Tau` 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 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 False

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.

`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

Description Specifies the time constant of the barostat.

`Type`

Type Multiple Choice

Default value None

Options [None, Berendsen, MTK]

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.

3.6.4 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.

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.

`SamplingFreq`

Type Integer

Default value 100

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 is greater than 0 then a temperature profile will be generates along each of the three unit cell axes. By default, no profile is generated.

`Checkpoint`

Type Block

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

`Frequency`

Type Integer

Default value 1000

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

`CalcPressure`

Type Bool

Default value False

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 False

Description Print the chemical system before and after the simulation.

`Velocities`

Type Bool

Default value False

Description Print the atomic velocities before and after the simulation.

3.6.5 Molecule Gun: adding molecules during simulation

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

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. The total momentum of the added molecule is not conserved.

`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 ADFinput, 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.

`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 False

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. ADFinput 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 TemperatureSigma.

`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 ADFinput 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.

3.6.6 Removing molecules during simulation

This feature can be used, for example, to remove reaction products from the system.

`MolecularDynamics`

`RemoveMolecules`

Type Block

Recurring True

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.

`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 a 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 atomic units.

`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.

`SinkBox`

Type Block

Description Part of the simulation box where molecules will be removed. By default, molecules matching the formula will be removed regardless of their location. If this block is present a molecule will 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 atomic units.

`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.

`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.

`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.

3.6.7 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.

`Input`

Type Non-standard block

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

3.6.8 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.

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

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

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

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 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 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

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

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

`rmax`

Type Float

Unit Angstrom

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

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.

3.6.9 Non-equilibrium MD (NEMD): 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. These 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, HEX [*18* (page 164)] and eHEX [*19* (page 164)]. 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.

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

MolecularDynamics

HeatExchange

Type Block

Recurring True

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 transfered 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).

`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. 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.

`FirstAtom`

Type Integer

Description Index of the first atom of the region. This key is ignored if the [Box] block is present.

`LastAtom`

Type Integer

Description Index of the last atom of the region. 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).

`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.

`FirstAtom`

Type Integer

Description Index of the first atom of the region. This key is ignored if the [Box] block is present.

`LastAtom`

Type Integer

Description Index of the last atom of the region. 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 43) section above on how to obtain the temperature profile from which the *grad(T)* can be computed.

3.7 Vibrational analysis

This section documents AMS tasks focused on generating (approximations of) vibrational normal modes, or the calculation of properties of these vibrational modes. We discuss both conventional vibrational analysis and more selective methods, as implemented in AMS.

3.7.1 Full analysis

With vibrational analysis, we focus on the calculation of molecular vibrational modes and their associated properties. These molecular normal modes are typically 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 generalised to multiple nuclear coordinates. The normal modes and their frequencies then become eigenvectors and eigenvalues of a force constant matrix, the Hessian:

$$H_{ij} = \frac{\partial^2 E}{\partial R_i \partial R_j}$$

This Hessian can be requested as a *PES property* (page 87) in AMS.

```
Properties
   Hessian [True | False]
End
```

Properties

Hessian

Type Bool

Default value False

Description Whether or not to calculate the Hessian.

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 99) 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 Hessian requires 6N single points calculation, which can take a considerable amount of time for large systems.

You can also request AMS to calculate the normal modes (and their properties). AMS will obtain the normal modes as the eigenvectors of the mass-weighted Hessian.

```
Properties
   NormalModes [True | False]
End
```

Properties

NormalModes

Type Bool

Default value False

Description Whether or not to calculate the normal modes of vibration (and of molecules the corresponding Ir intensities.)

The frequencies of the normal modes are then obtained from the Hessian eigenvalues, which are the effective force constants of the vibrational modes:

$$\nu = \frac{1}{2\pi c}\sqrt{\frac{\lambda}{\mu_r}}$$

Here ν is the vibrational frequency in cm^{-1}, λ is the Hessian eigenvalue and μ_r is the reduced mass associated with the vibrational mode.

Note that the normal modes also include 3 translational and 3 rotational modes (2 for linear molecules). Together, these are referred to as "rigid modes". While they are not presented as part of the vibrational spectra, they are still stored in the *engine result file* (page 4) in the `Vibrations` section.

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:

$$I_{IR} = \frac{N\pi}{3c^2}\sum_{\alpha}\Big(\sum_{j}\frac{\partial\mu_\alpha}{\partial R_j^m}Q_j^m\Big)^2$$

Where α denotes the x-,y- and z-components of the dipole moment μ, and Q^m is the mass-weighted vibrational normal mode.

Note: To obtain accurate results using this method, the harmonic approximation must hold. Your calculations should thus be done at the system's equilibrium geometry. Since different engines, functionals, parameter sets etc. all yield slightly different PESs, it is recommended to always precede the calculation of the normal modes and/or Hessian with a geometry optimisation at the desired engine settings. One can either run this optimisation first and then get the Hessian/modes from a *PES point* (page 11) calculation, or combine both steps into one job by using the *geometry optimization task* (page 11) together with the `Properties%NormalModes` keyword.

3.7.2 Mode selective analysis

The vibrational analysis tools in AMS also provide three additional methods: *Mode Scanning* (page 58), *Mode Refinement* (page 58) and *Mode Tracking* (page 66). The latter two methods can be used for the calculation of select modes or select regions of the vibrational spectrum, whereas Mode Scanning can be used to obtain more accurate approximations of the vibrational mode properties. These methods are discussed in more detail on their respective pages:

Mode Scanning

Mode Scanning can be used to obtain more accurate approximations for the properties of the vibrational normal modes. 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.

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. This 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 99)):

$$I_{IR} = \frac{N\pi}{3c^2}\sum_{\alpha}\left(\frac{\partial\mu_\alpha}{\partial Q^m}\right)^2$$

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 99).

Calculation setup

A numerical frequency calculation is performed by requesting the `ModeScanning` task:

```
Task ModeScanning

ModeScanning

   ModePath adf.rkf

   # select all modes with imaginary frequencies
   ModeSelect
      ImFreq true
   End

   Displacement 0.001

End
```

The details of the calculation are specified in the `ModeScanning` block. Here, `ModePath` 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 77). 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 66) and *Vibrational Mode Refinement* (page 62) 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.

Overview of input options

Below is the overview for all the keys in the `ModeScanning` block:

```
ModeScanning
   Displacement float
   ModePath string
   ModeSelect
      FreqAndIRRange float_list
      FreqRange float_list
      Full [True | False]
      HighFreq integer
      HighIR integer
      IRRange float_list
      ImFreq [True | False]
      LowFreq integer
      LowFreqNoIm integer
      LowIR integer
      ModeNumber integer_list
   End
End
```

`ModeScanning`

Type Block

Description Input data for the ModeScanning task.

`Displacement`

Type Float

Default value 0.001

Description Step size for finite difference calculation of frequencies and IR intensities.

`ModePath`

Type String

Description Path to a .rkf file containing the modes which are to be scanned. Which modes will be scanned is selected using the criteria from the [ModeSelect] block.)

`ModeSelect`

Type Block

Description Pick which modes to scan from those read from file.

`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 scanned. (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 scanned. (2 numbers: a upper and a lower bound.)

`Full`

Type Bool

Default value False

Description Scan all modes.

`HighFreq`

Type Integer

Description Scan the N modes with the highest frequencies.

`HighIR`

Type Integer

Description Scan 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 scanned. (2 numbers: a upper and a lower bound.)

`ImFreq`

Type Bool

Default value False

Description Scan all modes with imaginary frequencies.

`LowFreq`

Type Integer

Description Scan the N modes with the lowest frequencies. (Includes imaginary modes which are recorded with negative frequencies.)

`LowFreqNoIm`

Type Integer

Description Scan the N modes with the lowest non-negative frequencies. (Imaginary modes have negative frequencies and are thus omitted here.)

LowIR

Type Integer

Description Scan the N modes with the smallest IR intensities.

ModeNumber

Type Integer List

Description Indices of the modes to scan.

Mode Refinement

The vibrational Mode Refinement method not only refines frequencies, but also tries to correct the vibrational modes themselves. 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 [*12* (page 163)].

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 58) 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.

Calculation setup

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 minimise 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 minimise 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 `ModeRefinement` task:

```
Task ModeRefinement

ModeRefinement

   ModePath adf.rkf

   ModeSelect
      ...
   End

   Displacement 0.001

   ScanModes true

End
```

The details of the calculation are specified in the `ModeRefinement` block. Here, `ModePath` 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 77). 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 `ModeRefinement` block can be used to automatically run a *numerical frequencies* (page 58) 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 minimise 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.

Overview of input options

Below is the overview for all the keys in the `ModeRefinement` block:

```
ModeRefinement
   Displacement float
   ModePath string
   ModeSelect
      FreqAndIRRange float_list
      FreqRange float_list
      Full [True | False]
      HighFreq integer
      HighIR integer
      IRRange float_list
      ImFreq [True | False]
      LowFreq integer
      LowFreqNoIm integer
      LowIR integer
      ModeNumber integer_list
   End
   ScanModes [True | False]
End
```

`ModeRefinement`

Type Block

Description Input data for ModeRefinement tasks.

`Displacement`

Type Float

Default value 0.001

Description Step size for finite difference calculation of frequencies and IR intensities.

`ModePath`

Type String

Description Path to a .rkf file containing the modes which are to be scanned. Which modes will be refined is selected using the criteria from the [ModeSelect] block.)

`ModeSelect`

Type Block

Description Pick which modes to refine from those read from file.

`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 refined. (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 refined. (2 numbers: a upper and a lower bound.)

`Full`

Type Bool

Default value False

Description Refine all modes.

`HighFreq`

Type Integer

Description Refine the N modes with the highest frequencies.

`HighIR`

Type Integer

Description Refine 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 refined. (2 numbers: a upper and a lower bound.)

`ImFreq`

Type Bool

Default value False

Description Refine all modes with imaginary frequencies.

`LowFreq`

Type Integer

Description Refine the N modes with the lowest frequencies. (Includes imaginary modes which are recorded with negative frequencies.)

`LowFreqNoIm`

Type Integer

Description Refine the N modes with the lowest non-negative frequencies. (Imaginary modes have negative frequencies and are thus omitted here.)

`LowIR`

Type Integer

Description Refine the N modes with the smallest IR intensities.

`ModeNumber`

Type Integer List

Description Indices of the modes to refine.

`ScanModes`

Type Bool

Default value False

Description 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.

Mode Tracking

The Mode Tracking task is an interface for mode- and intensity-tracking methods, adapted from the MoViPac suite [*6* (page 163)- *10* (page 163)]. 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.

Warning: The Mode Tracking implementation in AMS is new and still rather experimental. One should be extra careful when running Mode Tracking calculation, and initially validate the results against the full vibrational analysis.

Mode Tracking starts with a *numerical frequency* (page 58) 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 minimised 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 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:

$$r = \sum_k c_k \cdot \left[\sigma_k - \lambda \cdot b_k \right]$$

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 69). If the residual of this mode has been minimised 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`:

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. [*11* (page 163)]

vdVorst-Sleijpen-Jacobi-Davidson:

This variant of the Jacobi-Davidson scheme from Sleijpen & vdVorst [*11* (page 163)] 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.

There are 4 methods to obtain the approximate Hessian H_A, used by both update methods. They are set by `HessianGuess`:

- `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. [*11* (page 163)]
- `Inline` will read a Hessian specified in the input file, in the `HessianInline` block. This allows you to use Hessians generated in external programs:

```
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   0.32143213  -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
       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
```

- `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.

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 orthogonalise the new guess mode to the existing basis. An iterative procedure is necessary to account for numerical noise.

- `GramSchmidt` sets whether to perform this Gram-Schmidt orthogonalisation step. It is `on` by default.
- `GramSchmidtTolerance` is 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 $||{b_k^m}^T B^m||$.
- `GramSchmidtIterations` is 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.

Additional input parameters:

- `Displacement` 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.
- `MaxIterations` is 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.

- The `ModeScanning` key can be used to automatically run a *numerical frequencies* (page 58) 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.

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 criterium, 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 2 categories:

Mode Tracking:

The original tracking methods focus on obtaining as accurate as possible a normal mode for the system. This class of tracking methods focusses either on accuracy of the mode, or obtaining modes with particular vibrational character:

- `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 optimisation 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 optimisation. Since we essentially "forget" about earlier iterations, this procedure allows the optimisation 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 prioritises 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 prioritise 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 criterium.
- `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 optimisation 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 focusses 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 optimisation.) 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 117)).

Intensity Tracking:

This class of methods focusses on tracking modes based on their intensity in e.g. the infrared spectrum, rather than focussing on getting a mode with a particular type of vibration.

- `IRInitial` chooses the mode with the IR intensity closest to that of the initial guess. This constrains the optimisation 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 `IRIntitial` or `IRPrevious`, then we could be tracking any of one of these. Conversely, convergence behaviour 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 labelled 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 ADFSpectra in the GUI to visualise 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 optimisation 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 minimisation, 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`.

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 `TrackedMode`:

- `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 77).
- `MassWeighInlineMode` decides whether the initial guess modes need to be mass-weighed. As discussed above, Mode Tracking uses mass-weighted normal modes. In most cases, the normal modes are given in reg-

ular 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-weighing, and you should set `MassWeighInlineMode false`.

Convergence

In order to guide the Mode Tracking procedure, several convergence criteria are used:

- `ToleranceForNorm` 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` 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 prioritise either the total vibration or more localised character.
- `ToleranceForBasis` 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.

Note that the iterative procedure is stopped as soon as any one of these convergence criteria is satisfied. 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`.

Overview of input options

Below is the overview for all the keys in the `ModeTracking` block:

```
ModeTracking
   Displacement float
   HessianGuess [Unit | File | UFF | Inline]
   HessianInline # Non-standard block. See details.
      ...
   End
   HessianPath string
   MassWeighInlineMode [True | False]
   MaxIterations integer
   ModeInline # Non-standard block. See details.
      ...
   End
   ModePath string
   ModeSelect
      FreqAndIRRange float_list
      FreqRange float_list
      Full [True | False]
      HighFreq integer
      HighIR integer
      IRRange float_list
      ImFreq [True | False]
      LowFreq integer
      LowFreqNoIm integer
      LowIR integer
      ModeNumber integer_list
   End
   ScanModes [True | False]
   ToleranceForBasis float
   ToleranceForNorm float
   ToleranceForResidual float
   TrackedMode [Inline | File | Hessian]
   TrackingMethod [...]
   UpdateMethod [JD | D]
End
```

`ModeTracking`

Type Block

Description Input data for ModeTracking task.

`Displacement`

Type Float

Default value 0.01

Description Step size (in Bohr) for finite difference calculation of frequencies and IR intensities during mode tracking iterations.

`HessianGuess`

Type Multiple Choice

Default value UFF

Options [Unit, File, UFF, Inline]

Description Sets how to obtain the guess for the Hessian used in the preconditioner.

`HessianInline`

Type Non-standard block

Description Initial guess for the (non-mass-weighted) Hessian in a 3N x 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].

`MassWeighInlineMode`

Type Bool

Default value True

Description The supplied modes must be mass-weighed. This tells the program to mass-weigh the supplied modes in case this has not yet been done. (True means the supplied modes will be mass-weighed by the program, e.g. the supplied modes are non-mass-weighed.)

`MaxIterations`

Type Integer

Description Maximum number of allowed iterations.

`ModeInline`

Type Non-standard block

Recurring True

Description Coordinates of the mode which will be tracked in a N x 3 block (same as for atoms), used when [TrackedMode] = [Inline]. Rows must be ordered in the same way as in the [System%Atoms] block.

`ModePath`

Type String

Description Path to a .rkf file containing the modes which are to be tracked. Which modes will be refined is selected using the criteria from the [ModeSelect] block.)

`ModeSelect`

Type Block

Description Pick which modes to track from modes generated from Hessian or read from file.

`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 tracked. (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 tracked. (2 numbers: a upper and a lower bound.)

`Full`

Type Bool

Default value False

Description Track all modes.

`HighFreq`

Type Integer

Description Track the N modes with the highest frequencies.

`HighIR`

Type Integer

Description Track 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 tracked. (2 numbers: a upper and a lower bound.)

`ImFreq`

Type Bool

Default value False

Description Track all modes with imaginary frequencies.

`LowFreq`

Type Integer

Description Track the N modes with the lowest frequencies. (Includes imaginary modes which are recorded with negative frequencies.)

`LowFreqNoIm`

Type Integer

Description Track the N modes with the lowest non-negative frequencies. (Imaginary modes have negative frequencies and are thus omitted here.)

`LowIR`

Type Integer

Description Track the N modes with the smallest IR intensities.

`ModeNumber`

Type Integer List

Description Indices of the modes to track.

`ScanModes`

Type Bool

Default value False

Description 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.

`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.

`TrackedMode`

Type Multiple Choice

Default value File

Options [Inline, File, Hessian]

Description Set how the initial guesses for the modes are supplied.

`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.

`UpdateMethod`

Type Multiple Choice

Default value JD

Options [JD, D]

Description Chooses the method for expanding the Krylov subspace: (D) Davidson or (JD) vdVorst-Sleijpen variant of Jacobi-Davidson.

Mode Scanning (page 58) is an extension of the frequency scanning options that were part of ADF and BAND in earlier versions of the Amsterdam Modeling Suite. It is primarily used to identify spurious imaginary modes obtained from the normal modes calculation discussed earlier. Alternatively, you can use it to improve the numerical accuracy of the normal mode properties or to obtain approximations for these properties at higher levels of theory.

Both *Mode Tracking* (page 66) and *Mode Refinement* (page 62) can be used to obtain select vibrational modes without construction of the (full) Hessian. In the work-flow for both of these methods, we start with e.g. a semi-empirical method such as MOPAC to get an initial approximation of the normal modes, which are used to obtain accurate approximations of the normal modes on e.g. DFT-level. While the methods appear similar, it is important to stress their differences. Here, we give our recommendations on which methods you should use. (You can read up on the details of these methods on the respective pages.)

Mode Tracking:

- 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 Refinement:

- Refinement of entire spectral regions, but requires a sufficient number of modes in the basis for sufficient accuracy.
- 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.

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 results 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 minimises 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.

Mode Tracking on the other hand 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.

Which method to use thus primarily depends on the type and number of modes you are refining:

- The advantage of Mode Refinement 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.

Selecting modes

Mode Scanning, Mode Refinement and Mode Tracking all require you to supply a set of modes as input. 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.

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 configuration blocks for each task. You can read how to set up these task-specific blocks for each method on their individual pages. Note that 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.

Below is an overview of all the available options for the ModeSelect block. The options are not mutually exclusive, e.g.:

```
ModeTracking

  ModeSelect
      HighFreq 1
      HighIR 1
  End

  ... other Mode Tracking options ...

End
```

This will select 2 modes: the one with the highest frequency and the one with the highest IR intensity. If these modes happen to be the same one however (the mode with the highest frequency also has the highest IR intensity), only 1 mode is selected.

ModeSelect

- HighFreq followed by an integer N will select the N modes with the highest frequencies.
- LowFreq followed by an integer N will select the N modes with the lowest frequencies. Imaginary modes are given with negative frequencies in AMS, and are included in this selection.
- LowFreqNoIm is the same as LowFreq except imaginary modes are omitted.
- ImFreq will select all imaginary modes.
- ModeNumber allows you to supply a list of integers. Each integer is the index of the mode in the order that they appear in the file. E.g. benzene has 30 vibrational modes, which are numbered 1-30.
- FreqRange selects all modes whose frequency falls in a specific range. 2 values must be supplied to mark this frequency range. Calculating all modes with e.g. frequencies higher than 3000cm-1 can be achieved by making the upper bound very large: FreqRange 3000 1000000
- IRRange selects all modes whose IR intensity falls in a specific range. 2 values must be supplied to mark this IR intensity range, the same way as for FreqRange.
- FreqAndIRRange combines FreqRange and IRRange. It selects modes in a frequency range whose IR intensity falls into a specified range as well. 4 values must be supplied: the first 2 specify the frequency range, the final 2 specify the IR intensity range.
- HighIR followed by an integer N will select the N modes with the highest IR intensities.
- LowIR followed by an integer N will select the N modes with the lowest IR intensities.
- Full requests a full frequency calculation. This will select all modes. This only make sense for Mode Scanning calculations, as tracking or refining all modes is just an overcomplicated way of doing the *full vibrational analysis* (page 57).

```
[ ModeScanning | ModeRefinement | ModeTracking ]

  ModeSelect
      # select the 2 modes with the highest frequency
      HighFreq 2
      # select the 2 modes with the lowest frequency (including imaginary modes)
      LowFreq 2
      # select the 3 modes with the lowest energy
      LowFreqNoIm 3
      # Select modes #1 #7 & #19
      ModeNumber 1 7 19
      # Select modes with frequencies between 3000 - 3200 cm-1
      FreqRange 3000 3200
      # Select modes with IR intensities between 5 - 10 km/mol
      IRRange 5 10
      # Select modes with frequencies between 1000 - 1500 cm-1 that have
      # IR intensities between 10 - 30 km/mol
      FreqAndIRRange 1000 15000 10 30
      # Select the mode with the highest IR intensity
      HighIR 1
      # Select the 3 modes with the lowest IR intensities
      LowIR 3
      # Request all modes
      Full true
  End

End
```

3.8 Grand Canonical Monte Carlo (GCMC)

Tip: Take a look at the GCMC tutorial and learn how to setup a GCMC calculation.

3.8.1 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 Duin [*13* (page 163), *14* (page 163)]. 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).

3.8.2 Method Details

Overview

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 between 0 and 1. The new volume will be calculated as: V_{new} = exp(random(-1:1)*VolumeChangeMax)*V_{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 reservoir [13 (page 163)]:

$$\mu_O^{MC}(T,P) = \frac{1}{2}\mu_{O2}^{MC}(T,P) = \frac{1}{2}\left[\mu_{O2}^{ref}(T,P_{ref}) + kTln\left(\frac{P}{P_{ref}}\right) - E_{O2}^{diss}\right]$$

where the reference chemical potential $\mu_{O2}^{ref}(T,P_{ref})$ is the experimentally determined chemical potential of O2 at T and P_{ref}, $kTln\left(\frac{P}{P_{ref}}\right)$ is the pressure correction to the free energy, and E_{O2}^{diss} is the dissociation energy of the O_2 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:

$E_{old}^{MC} = E_{old}^{AMS} + \mu^{MC}$ for an Insert move;

$E_{old}^{MC} = E_{old}^{AMS} - \mu^{MC}$ for a Delete move;

$E_{old}^{MC} = E_{old}^{AMS}$ for a Displace move;

$E_{old}^{MC} = E_{old}^{AMS} - P(V_{new} - V_{old}) + N_{inserted} ln\left(\frac{V_{new}^{avail}}{V_{old}^{avail}}\right) kT$ 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.

3.8.3 Input

The GCMC functionality in AMS is triggered using the following Task key:

```
Task GCMC
```

```
GCMC
   AccessibleVolume float
   Ensemble [Mu-VT | Mu-PT]
   Iterations integer
   MapAtomsToOriginalCell [True | False]
   MaxDistance float
   MinDistance float
   Molecule
      ChemicalPotential float
      NoAddRemove [True | False]
      SystemName string
   End
   NonAccessibleVolume float
   NumAttempts integer
   Pressure float
   Removables # Non-standard block. See details.
      ...
   End
   Restart string
   Temperature float
   UseGCPreFactor [True | False]
   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

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 and 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: cmpot = Mu_O(T,P) = 1/2*Mu_O2(T,P) = 1/2 * [Mu_ref(T,P_ref) + kT*Log(P/Pref) - E_diss] where the reference chemical potential [Mu_ref(T,P_ref)] is the experimentally determined chemical potential of O2 at T and Pref; kT*Log(P/Pref) is the pressure correction to the free energy, and E_diss is the dissociation energy of the O2 molecule.

NoAddRemove

Type Bool

Default value False

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

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

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

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

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

`MaxDistance`

Type Float

Default value 3.0

Unit Angstrom

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 True

Description Use the GC pre-exponential factor for probability.

`VolumeOption`

Type Multiple Choice

Default value Free

Options [Free, Total, Accessible, FreeAccessible]

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 = totalVolume - occupiedVolume - NonAccessibleVolume; Total: V = totalVolume; Accessible: V = AccessibleVolume; FreeAccessible: V = AccessibleVolume - 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

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 Vnew = exp(random(-1:1)*VolumeChangeMax)*Vold

`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 True

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 11) section of this manual.

3.8.4 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 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.
- *Accepted* - a Fortran logical value containing the acceptance status of the MC move.
- *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.

CHAPTER

FOUR

PES POINT PROPERTIES

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 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 [True | False]
   StressTensor [True | False]
   Hessian [True | False]
   SelectedAtomsForHessian integer_list
   NormalModes [True | False]
   ElasticTensor [True | False]
   Phonons [True | False]
End
```

This page in the AMS manual describes all the supported properties.

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 4). 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 11) AMS is performing: For a *geometry optimization* (page 11) properties would for example only be calculated at the final, converged geometry. This behaviour can often be modified by keywords special to the particular running task.

4.1 Nuclear gradients and stress tensor

The first derivative with respect to the nuclear coordinates can be requested as follows:

```
Properties
   Gradients [True | False]
End
```

`Properties`

`Gradients`

Type Bool

Default value False

Description Whether or not to calculate the 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 4) 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). 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 [True | False]
End
```

Properties

StressTensor

Type Bool

Default value False

Description Whether or not to 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:

$$\sigma_\alpha = \frac{1}{V_0} \frac{\partial E}{\partial \epsilon_\alpha} \bigg|_{\text{constant atomic fractional coordinates}}$$

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 lenght of the unit cell).

The clamped-ion stress tensor (in Cartesian notation) is written to the engine result file in `AMSResults%StressTensor`.

4.2 Hessian and normal modes of vibration

The calculation of the second derivative of the total energy with respect to the nuclear coordinates is enabled by:

```
Properties
   Hessian [True | False]
End
```

Properties

Hessian

Type Bool

Default value False

Description Whether or not to calculate the Hessian.

The Hessian is not printed to the text output, but is saved in the engine result file as variable `AMSResults%Hessian`. Note that this ist just the plain second derivative (no mass-weighting) of the total energy and that for the order of its $3 \times n_{\text{atoms}}$ columns/rows, the component index increases the quickest: The first column refers to changes in the x-component of atom 1, the second to the y-component, the *fourth* to the x-component of the second atoms, and so on.

It is also possible to request the calculation of the normal modes of vibration:

```
Properties
   NormalModes [True | False]
End
```

Properties

NormalModes

Type Bool

Default value False

Description Whether or not to calculate the normal modes of vibration (and of molecules the corresponding Ir intensities.)

Note: For more information and advanced methods of calculating and analyzing molecular vibrations, see the manual chapter on the *vibrational analysis* (page 57) (mode scanning, refinement and tracking).

This implies the calculation of the Hessian, which is required for calculating normal modes. For engines that are capable of calculating dipole moments, this also enables the calculation of the infrared intensities, so that the IR spectrum can be visualized by opening the engine result file with ADFSpectra. The normal modes of vibration and the IR intensities are saved to the *engine result file* (page 4) 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 11) together with the `Properties%NormalModes` keyword.

Symmetry labels of the normal modes of molecules may be calculated if AMS uses symmetry in the calculation of normal modes (key `NormalModes%UseSymmetry`). If symmetry is used the nomal modes are projected against symmtric displacements for each irrep. If that is not successful the symmetric label is 'MIX'. 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. If that is the case or if key `NormalModes%UseSymmetry` is set to 'False' or if there is no symmetry, then no symmetry labels will be calculated.

```
NormalModes
   UseSymmetry [True | False]
End
```

NormalModes

Type Block

Description Configures details of a normal modes calculation.

UseSymmetry

Type Bool

Default value True

Description Whether or not to exploit the symmetry of the system in the normal modes calculation.

4.2.1 Thermodynamics (ideal gas)

Thermodynamic properties are always calculated whenever normal modes are computed. These properties are: Entropy, Internal Energy, Constant Volume Heat Capacity, Enthalpy and Gibbs free energy. 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).

```
Thermo
   Pressure float
   TMax float
   TMin float
   nSteps integer
End
```

Thermo

Type Block

Description Options for thermodynamic properties (assuming an ideal gas). The properties are computed for 'nSteps' temperatures in the range [TMin,TMax].

Pressure

Type Float

Default value 1.0

Unit atm

Description The pressure at which the thermodynamic properties are computed.

TMax

Type Float

Default value 298.15

Unit Kelvin

Description Maximum value for the temperature range.

TMin

Type Float

Default value 298.15

Unit Kelvin

Description Minimum value for the temperature range.

nSteps

Type Integer

Default value 1

Description The number of temperatures in the range [TMin,TMax].

4.2.2 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 diplacement vector for atom I in normal normal mode n.

Tip: The Partial Vibrational Spectra (PVDOS) can be visualized using the **ADFSpectra** 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.

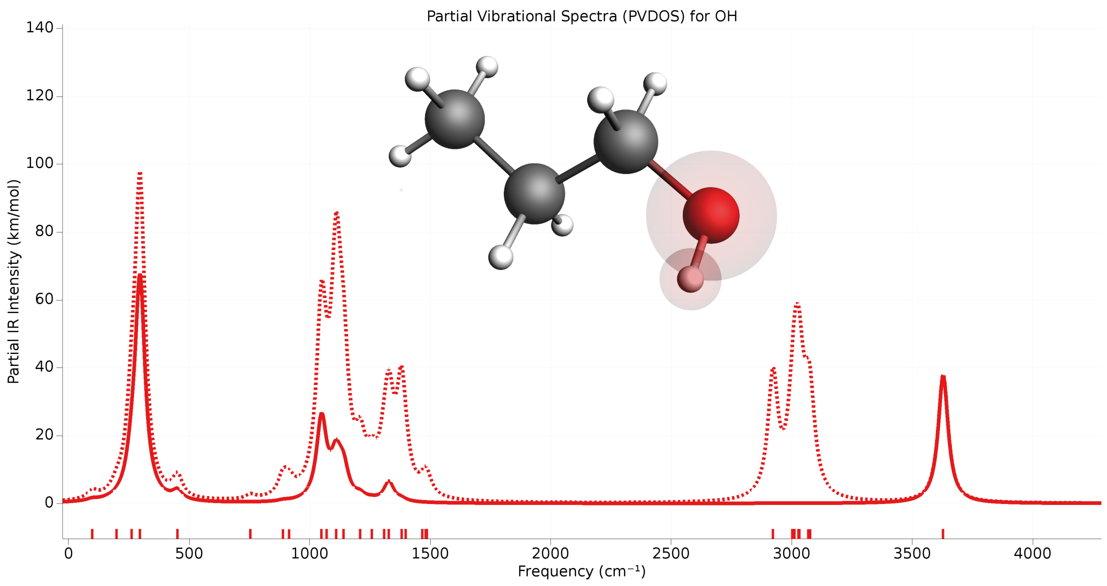


Fig. 4.1: Example of partial vibrational spectrum (PVDOS). The dotted line is the full IR spectrum of 1-propanol. The solid line is the PVDOS-scaled IR spectrum of the OH group (IR spectrum computed using GFN1-xTB).

The PVDOS matrix is not printed to the text output, but only saved to the engine binary output (.rkf) in the variable `Vibrations%PVDOS`.

4.3 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 lenght 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 153)

To compute the elastic tensor, request it in the `Properties` input block of AMS:

```
Properties
   ElasticTensor [True | False]
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 11) 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 4) 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
   Parallel
      nCoresPerGroup integer
      nGroups integer
      nNodesPerGroup integer
   End
   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

Description Maximum nuclear gradient for the relaxation of the internal degrees of freedom of strained systems.

`Parallel`

Type Block

Description The evaluation of the elastic tensor via numerical differentiation is an embarrassingly parallel problem. Double parallelization 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

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

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.

4.4 Phonons

Collective oscillations of atoms around theirs 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 150), *Example: Phonons with isotopes* (page 151) and diamond lattice optimization and phonons tutorial

The calculation of phonons is enabled in the `Properties` block.

```
Properties
   Phonons [True | False]
End
```

Note: Phonon calculations should be performed on optimized geometries, **including the lattice vectors**. This can be done by either reading an already optimized system as input, or combining the phonon calculation with the *geometry optimization task* (page 11).

The details of the phonon calculations are configured in the `NumericalPhonons` block:

```
NumericalPhonons
   SuperCell # Non-standard block. See details.
      ...
   End
   StepSize float
   DoubleSided [True | False]
   UseSymmetry [True | False]
   Interpolation integer
   NDosEnergies integer
   Parallel
      nCoresPerGroup integer
      nGroups integer
      nNodesPerGroup integer
   End
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
```

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 True

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.

UseSymmetry

Type Bool

Default value True

Description Whether or not to exploit the symmetry of the system in the phonon calculation.

`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.

Note that the numerical phonon calculation supports AMS' *double parallelism* (page 110), which can perform the calculations for the individual displacements in parallel. This is disabled by default but can be enabled using the keys in the `NumericalPhonons%Parallel` block:

`NumericalPhonons`

`Parallel`

Type Block

Description Computing the phonons via numerical differentiation is an embarrassingly parallel problem. Double parallelization 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. Keep in mind that the displacements for a phonon calculation are done on a super-cell system, so that every task requires more memory than the central point calculated using the primitive cell.

`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

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.

4.5 Numerical differentiation options

The following options apply whenever AMS computes gradients, hessians or stress tensors via numerical differentiation.

```
NumericalDifferentiation
   NuclearStepSize float
   Parallel
      nCoresPerGroup integer
      nGroups integer
      nNodesPerGroup integer
   End
   StrainStepSize float
   UseSymmetry [True | False]
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.

Parallel

Type Block

Description Numerical differentiation is an embarrassingly parallel problem. Double parallelization 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

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

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.

`UseSymmetry`

Type Bool

Default value True

Description Whether or not to exploit the symmetry of the system for numerical differentiations.

Note that the numerical differentiation calculation supports AMS' *double parallelism* (page 110), which can perform the calculations for the individual displacements in parallel. This is disabled by default but can be enabled using the keys in the `NumericalDifferentiation%Parallel` block.

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.

CHAPTER

FIVE

ENGINES

The engines are 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 11) or *molecular dynamics* (page 33) calculation, the engines calculate energies and gradients and in this way *define* the PES on which the driver works.

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.

5.1 Available engines

The following engines are available in the 2019 release of the Amsterdam Modeling Suite:

- **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.
- **UFF** An implementation of the Universal Force Field, a simple non-reactive force field covering the entire periodic table.
- **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.
- **ADF** A wrapper around the standalone ADF program, allowing it to be used as an engine from within the AMS driver.
- ***External* (page 101)** A flexible scripting interface that allows advanced users to use external atomistic modeling programs as engines in AMS.
- ***LennardJones* (page 105)** A simple toy engine implementing a Lennard-Jones potential.

5.2 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. Furthermore the graphical interface of the Amsterdam Modeling Suite can be used to analyze the results of these calculations. 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 `Engine` block for the external engine has just one important keyword, which is 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 $ADFBIN/startpython /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 157) 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,
   "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, the `Engine External` block also needs to contain some information about the capabilities of the external engines:

```
Engine External
   Execute {...}
   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 4). 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 $ADFBIN/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
   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 ADFSpectra 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.

5.3 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.

CHAPTER SIX

TECHNICAL TOPICS

6.1 Input syntax

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:

```
#!/bin/sh

$ADFBIN/ams << EOF

   ... AMS text input goes here:

   Block
      Keywork value
      OtherKeyword value
   End

EOF
```

This section of the AMS manual documents the syntax of the text input.

6.1.1 General remarks on input structure and parsing

- Most keys are optionals. 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:
USESYMMETRY 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
```

6.1.2 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
```

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
```

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

```
KeywordName 0.1 1.0E-2 1.3
```

6.1.3 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).

6.1.4 Units

Some keys have a default unit associated (not all keys have units). For such keys, the default unit is mention 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 [a.u.]
```

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`

6.2 Double parallelism

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 99). Therefore the main parallelization of AMS is inside of the engines, making sure that a good performance is obtained for *tasks* (page 11) such as *molecular dynamics* (page 33) or *geometry optimizations* (page 11), 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**. Double parallelization is used for the calculation of the *PES point properties* (page 87) which are derivatives, if these need to be done numerically:

- Numerical calculation of *forces / nuclear gradients* (page 89). 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 90) 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 90) 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 93). 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 95). 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.

In order to use double parallelization it has to be enabled explicitly in the input. This is done for the above mentioned properties individually, as one might want a different grouping strategy for each property. For each property 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.

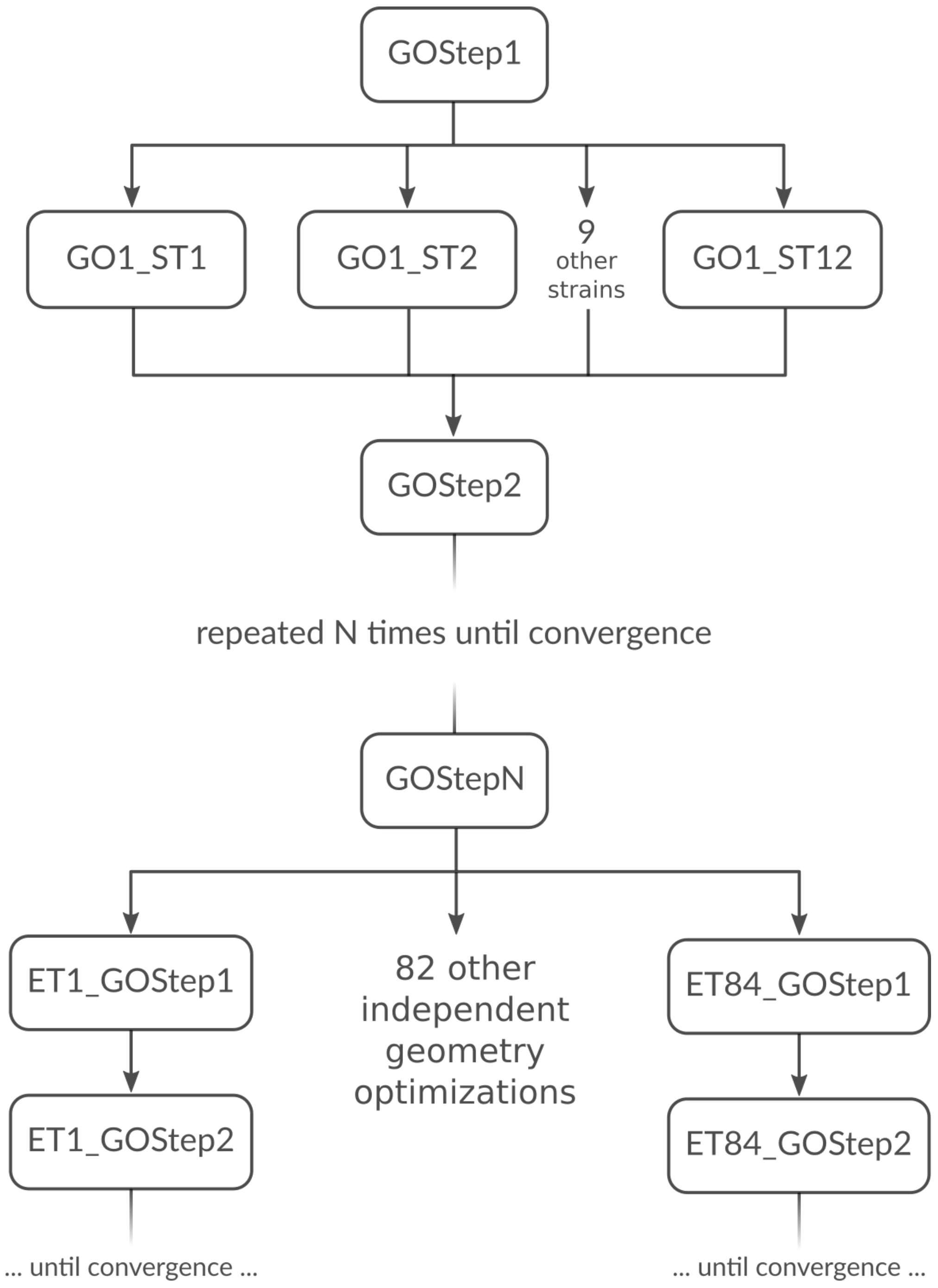
The optimal grouping strategy and number of groups depends on the total number of cores used in the calculation, the amount of independent tasks to be done in parallel, as well as the parallel scalability of the engine itself. In practice it can be a bit tricky. Suppose, as an example, that we want to calculate the elastic properties of a bulk material on a 32 core machine. The calculation of the *elastic tensor* (page 93) 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 90) 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 (which most engines can not calculate 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:


GOStep1
GO1_ST1
GO1_ST2
9 other strains
GO1_ST12
GOStep2
repeated N times until convergence
GOStepN
ET1_GOStep1
82 other independent geometry optimizations
ET84_GOStep1
ET1_GOStep2
ET84_GOStep2
... until convergence ...
... until convergence ...

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 best 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
```

6.3 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

$ADFBIN/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.

6.4 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

CHAPTER SEVEN

EXAMPLES

7.1 Geometry optimization

7.1.1 Example: Simple geometry optimization

Download GO_formaldehyde_noSCC.run

```
#!/bin/sh

$ADFBIN/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
```

7.1.2 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
# 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 $ADFBIN/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

   # Parallelize the calculation of the displacements used for the numerical
   # calculation of the Hessian. Aspirin is much too small for the DFTB engine
   # to parallelize efficiently internally, so parallelization at the driver
   # level will give better performance.
   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   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 $ADFBIN/ams << EOF

   # 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/dftb.rkf
      End
   End

   # Settings for the BAND engine:
   Engine BAND
      Basis Type=TZP
      XC GGA=PBE
   EndEngine

EOF
```

7.1.3 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 $ADFBIN/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
       Pressure $P
       Convergence Gradients=1.0e-5
   End

   NumericalDifferentiation
       # Parallelize the calculation of the strain displacements, necessary for
       # numerically calculating the stress tensor during the lattice optimization.
       Parallel nGroups=2
   End

   Properties
       # Request the calculation of phonons at the optimized geometry.
       Phonons Yes
   End

   NumericalPhonons
       Parallel nGroups=2
       SuperCell
           2 0 0
           0 2 0
           0 0 2
       End
   End

   Engine DFTB
       ResourcesDir DFTB.org/mio-1-1
       KSpace
           Type Symmetric
           Symmetric KInteg=5
       End
   EndEngine

EOF
done
```

7.1.4 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 $ADFBIN/ams << EOF
```

```
   Task GeometryOptimization

   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
       ResourcesDir Dresden
       DispersionCorrection Auto
   EndEngine

EOF


# 2. Distance constraints
# =======================

AMS_JOBNAME=dist $ADFBIN/ams << EOF

   Task GeometryOptimization

   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
       ResourcesDir Dresden
       DispersionCorrection Auto
   EndEngine

EOF


# 3. Dihedral angle constraint
# ============================

AMS_JOBNAME=dihed $ADFBIN/ams << EOF

   Task GeometryOptimization
   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
   End

   Engine DFTB
       ResourcesDir Dresden
       DispersionCorrection Auto
   EndEngine

EOF


# 4. Fixed atom constraint
# ========================

AMS_JOBNAME=atom $ADFBIN/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
      ResourcesDir DFTB.org/mio-1-1
   EndEngine
```

```
EOF


# 5. Fixed coordinate constraint
# ==============================

AMS_JOBNAME=coord $ADFBIN/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 the x-coordinate of all atoms.
      Coordinate 1 x
      Coordinate 2 x
      Coordinate 3 x
      Coordinate 4 x
   End

   Engine DFTB
      ResourcesDir DFTB.org/mio-1-1
   EndEngine

EOF


# 6. Fixed atom constraint (in periodic system)
# =============================================

AMS_JOBNAME=pbcatom $ADFBIN/ams << EOF

   Task GeometryOptimization

   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
      ResourcesDir DFTB.org/mio-1-1
   EndEngine

EOF


# 7. Block constraints (with listing the atoms in a block)
# ========================================================

AMS_JOBNAME=block_list $ADFBIN/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
            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 $ADFBIN/ams << EOF

    Task GeometryOptimization

    System
        Atoms
            C           0.5584839616765542   0.5023705181144142  -0.4625483159356394
            C.phenyl1   1.07173137896726     0.2125484528111251  -1.892767990599312
            C.phenyl1   1.699248504588085   -1.006061067555322   -2.191856791501442
            C.phenyl1   2.242484629452111   -1.236470028363516   -3.455616615521399
            C.phenyl1   2.18874580207099    -0.2444337131062739  -4.435483595049287
            C.phenyl1   1.604409798904145    0.9866950282217637  -4.135465239465763
            C.phenyl1   1.061086793296828    1.217355116664161   -2.871773146851866
```

```
          H.phenyl1   1.763625603740592   -1.780903563899969   -1.431707209662057
          H.phenyl1   2.716038261390732   -2.190869049673275   -3.672115451399807
          H.phenyl1   2.611833078693977   -0.4241619800888815  -5.420308290235123
          H.phenyl1   1.578029796368043    1.774138556616255   -4.884624561698751
          H.phenyl1   0.6247213391616491   2.187200330357715   -2.64521108544713
          C.phenyl2   1.303528070245188   -0.1416812092038768   0.7303699949711653
          C.phenyl2   0.8164830922475474  -1.314631142230651    1.326337082260565
          C.phenyl2   1.531799364672407   -1.947399963062604    2.342825210379356
          C.phenyl2   2.757684862125068   -1.432061688813837    2.765634667957531
          C.phenyl2   3.271640455523863   -0.2897364031184506   2.150731553729188
          C.phenyl2   2.556535912403799    0.3432056352653093   1.134221563049466
          H.phenyl2  -0.128925843064934   -1.7366201913903      0.9939642396630857
          H.phenyl2   1.133600273086767   -2.849990046242235    2.799740694330775
          H.phenyl2   3.31486005979636    -1.925049398411132    3.557912279830031
          H.phenyl2   4.236604921323707    0.1064455961800578   2.457138367063388
          H.phenyl2   2.976510069814392    1.222131876866508    0.6510413538003352
          C.phenyl3  -0.930165749820548    0.9153412637395284  -0.5420710991631585
          C.phenyl3  -1.791729737216814    0.6892660986048864   0.5418285200469819
          C.phenyl3  -3.111373625199894    1.139542032267652    0.5090625363459357
          C.phenyl3  -3.586568528476239    1.843983986018719   -0.5977864609101087
          C.phenyl3  -2.726152821786783    2.111108432452229   -1.663369105880468
          C.phenyl3  -1.406454626777386    1.660929752085611   -1.63085383469072
          H.phenyl3  -1.428888457076976    0.1571120160719108   1.417905619994904
          H.phenyl3  -3.76723983501283     0.9462006794587581   1.35432032282366
          H.phenyl3  -4.614972346570283    2.194578435055282   -0.6233521468909432
          H.phenyl3  -3.080200905921361    2.678981846821393   -2.520207901691867
          H.phenyl3  -0.7413545301831963   1.891248563160919   -2.459672151335554
          C.phenyl4   1.235557647765805    1.735720249011045    0.1803884343948648
          C.phenyl4   1.377191890012647    1.826646222422494    1.573181692925026
          C.phenyl4   1.905898822116255    2.975086608901246    2.16214311213053
          C.phenyl4   2.280792642899383    4.061906342938987    1.371311861877147
          C.phenyl4   2.105006642447361    3.998471351380415   -0.0115253875199488
          C.phenyl4   1.576317094651283    2.850163227898022   -0.6007264381779673
          H.phenyl4   1.072424817958776    0.9937816064904853   2.202306496283991
          H.phenyl4   2.017471491684088    3.023369029562452    3.242524256706377
          H.phenyl4   2.693031233132915    4.956641734238467    1.830324484771476
          H.phenyl4   2.372569859099136    4.8485771293401     -0.6342066225733602
          H.phenyl4   1.427765851939196    2.820397327218896   -1.677480576376967
          #  ^---- Element symbols augmented with a tag that we will use in the␣
↪Constraints block
      End
  End

  GeometryOptimization
      Convergence
          Energy 1.0e-6
          Gradients 1.0e-4
          Step 1.0e-4
      End
  End

  Constraints
     # Use the tag 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
```

7.2 Transition state search

7.2.1 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 $ADFBIN/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
      Basis Type=DZP
      KSpace Quality=Good
   EndEngine

EOF


# Second run: TS search with initial Hessian
# ==========================================

AMS_JOBNAME=TS $ADFBIN/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
```

7.2.2 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
# standpoint this is not a good approximation. The graphene slab is
# intentionally not perfectly symmetric.

AMS_JOBNAME=PESScan $ADFBIN/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

   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
   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 ADFMovie 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
H   -0.05533871972549955    -0.06805093626643093   1.500000013242627
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 $ADFBIN/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.
      SelectedAtomsForHessian 1 2
   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 the frequencies from the kf file using adfreport:"
$ADFBIN/adfreport 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 $ADFBIN/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
      NormalModes Yes
      SelectedAtomsForHessian 1 2
   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 adfreport:"
$ADFBIN/adfreport TS.results/dftb.rkf -r "AMSResults%Energy"
```

7.3 Intrinsic reaction coordinate (IRC)

7.3.1 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 $ADFBIN/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,
# 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 $ADFBIN/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
```

7.3.2 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=CCOc1ccccc1
# ============================================================

AMS_JOBNAME=TS_molecule $ADFBIN/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
      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

NumericalDifferentiation
   Parallel nCoresPerGroup=1
End

Engine DFTB
    ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor


AMS_JOBNAME=IRC_molecule $ADFBIN/ams << eor

Task IRC

IRC
   MaxIterations 1000
   InitialHessian
      Type FromFile
      File TS_molecule.results/dftb.rkf
   End
end

LoadSystem
    File TS_molecule.results/ams.rkf
End

Engine DFTB
    ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor


# ==========================================================================
# Claisen rearrangement for a (periodic) polymer containing the same aromatic
# ring of the previous calculation (from C=CCC1C=CC=CC1=O to C=CCOc1ccccc1)
# ============================================================================

AMS_JOBNAME=TS_polymer $ADFBIN/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
    Method Quasi-Newton
    InitialHessian Type=Calculate
End

NumericalDifferentiation
   Parallel nCoresPerGroup=1
End

Engine DFTB
    ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor


AMS_JOBNAME=IRC_polymer $ADFBIN/ams << eor

Task IRC

IRC
   MaxIterations 1000
   InitialHessian
      Type FromFile
      File TS_polymer.results/dftb.rkf
   End
   Direction Forward
end

LoadSystem
    File TS_polymer.results/ams.rkf
End

Engine DFTB
    ResourcesDir DFTB.org/3ob-3-1
EndEngine
eor
```

7.4 PES scan

7.4.1 Example: Linear transit

Download LinearTransit.run

```
#!/bin/sh



echo "================="
echo "HCN isomerization"
echo "================="
echo

AMS_JOBNAME=HCN_isomerization $ADFBIN/ams << EOF

   Task PESScan
   # (Linear transit is just a PES scan with 1 scan coordinate.)

   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 $ADFBIN/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
      ResourcesDir DFTB.org/mio-1-1
   EndEngine

EOF



echo
echo "===================="
echo "Hydrocarbon reaction"
echo "===================="
echo

AMS_JOBNAME=hydcarb $ADFBIN/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 "====================================="
echo "Retinal trans -> 11-cis isomerization"
echo "====================================="
```

```
echo

AMS_JOBNAME=retinal_transcis $ADFBIN/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
         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
```

7.4.2 Example: 2D PES scan

Download PESScan.run

```
#!/bin/sh


echo "=============="
echo "Ethane torsion"
echo "=============="
echo

AMS_JOBNAME=ethane_torsion $ADFBIN/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
      # 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

   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 $ADFBIN/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

   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.
```

```
echo "=============================="
echo "Ethane gap filling test (1/2)"
echo "=============================="
echo

AMS_JOBNAME=ethane_nofillgaps $ADFBIN/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
   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
      End
   End

   Engine DFTB
      ResourcesDir QUASINANO2015
   EndEngine

EOF


echo "=============================="
echo "Ethane gap filling test (2/2)"
echo "=============================="
echo

AMS_JOBNAME=ethane_fillgaps $ADFBIN/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
   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
      ResourcesDir QUASINANO2015
   EndEngine

EOF
```

7.5 Molecular dynamics

7.5.1 Example: Simple MD for H2

Download MD_hydrogen_longrun.run

```
#!/bin/sh

$ADFBIN/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
   ResourcesDir Dresden
   Occupation Strategy=Fermi Temperature=5
   Repulsion
       forcePolynomial true
   End
   DispersionCorrection Auto
EndEngine

eor
```

7.5.2 Example: MD for a box of water

Download H2O_nreac.run

7.6 Vibrational analysis

7.6.1 Example: Mode Refinement

Download VATools_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 $ADFBIN/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
         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

   # Do displacements for the numerical Hessian in parallel.
```

```
   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   End

EOF


# Step 2: Mode refinement of the DFTB C=O and C=C stretch bands at the DFT level
# ------------------------------------------------------------------------------

AMS_JOBNAME=ModeRefinement $ADFBIN/ams << EOF

   LoadSystem
      File FREQ_DFTB.results/dftb.rkf
   End

   Task ModeRefinement

   ModeRefinement
      ModePath FREQ_DFTB.results/dftb.rkf
      ModeSelect
         FreqRange 1500 1800
      End
   End

   Engine BAND
      # Settings from the paper:
      XC
         GGA BP86
      End
      Basis
         Type TZP
      End
      # Just to make this test run faster:
      NumericalQuality Basic
   EndEngine

   NumericalDifferentiation
      # Do +/- displacements along the mode in parallel.
      Parallel nCoresPerGroup=1
   End

EOF
```

7.6.2 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 $ADFBIN/ams << EOF

   System
      Atoms
         C       0.00000000      -0.00000000       0.00000000
         C       0.97860571       1.25132756       1.95058924
         C       0.97860571      -1.25132756       1.95058924
         C       0.81256878      -0.00000000       2.80673224
         C      -0.03962209      -1.27292583       0.80215060
         C      -0.03962209       1.27292583       0.80215060
         O       0.06261908       0.00000000      -1.25128260
         H       0.12330579      -2.12275790       0.11986078
         H      -1.06001189      -1.37219989       1.22906269
         H       0.87152406      -2.16703679       2.56054100
         H       2.00106532      -1.26374568       1.52181021
         H      -0.19678477       0.00000000       3.26831156
         H       1.54931293      -0.00000000       3.63180857
         H       0.87152406       2.16703679       2.56054100
         H       2.00106532       1.26374568       1.52181021
         H       0.12330579       2.12275790       0.11986078
         H      -1.06001189       1.37219989       1.22906269
      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 $ADFBIN/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

   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   End

EOF


# 3. ModeScanning of DFTB C=O stretch mode with DFT
# -------------------------------------------------

AMS_JOBNAME=ModeScanning $ADFBIN/ams << EOF

   LoadSystem
      File DFTB.results/ams.rkf
   End
   LoadEngine DFT.results/band.rkf

   Task ModeScanning

   ModeScanning
      ModePath DFTB.results/dftb.rkf
      ModeSelect
         HighIR 1 # This should select the C=O stretch
      End
   End

   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   End

EOF


# 4. Mode tracking with DFT starting from DFTB C=O stretch mode
# -------------------------------------------------------------

AMS_JOBNAME=ModeTracking $ADFBIN/ams << EOF

   LoadSystem
      File DFT.results/ams.rkf
   End
   LoadEngine DFT.results/band.rkf

   Task ModeTracking

   ModeTracking
      TrackedMode File
      ModePath DFTB.results/dftb.rkf
      HessianGuess File
      HessianPath DFTB.results/dftb.rkf
      ModeSelect
         HighIR 1 # This should select the C=O stretch
      End
   End
   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   End

EOF


# 5. Mode tracking with DFT starting from a pure C=O stretch
# ----------------------------------------------------------

AMS_JOBNAME=ModeTracking_COStretch $ADFBIN/ams << EOF

   LoadSystem
      File DFT.results/ams.rkf
   End
   LoadEngine DFT.results/band.rkf

   Task ModeTracking

   ModeTracking
      TrackedMode 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
      End
      HessianGuess File
      HessianPath DFTB.results/dftb.rkf
      ModeSelect
         HighIR 1 # This should select the C=O stretch
      End
      TrackingMethod OverlapPrevious
                   #     ^-- Probably better than the default.
                   #         Our initial mode is not particularly close yet ...
   End

   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   End

EOF
```

7.7 PES point properties

7.7.1 Example: Phonons for graphene

Download Phonons_Graphene.run

```
#!/bin/sh

AMS_JOBNAME=graphene $ADFBIN/ams << EOF

   Task GeometryOptimization

   GeometryOptimization
     OptimizeLattice True
     Convergence Gradients=1.0e-5
   End

   NumericalDifferentiation
      Parallel nGroups=2
   End

   Properties
      Phonons True
   End

   NumericalPhonons
     SuperCell
       2 0
       0 2
     End
     Parallel nGroups=4
   End

   System
      Atoms
         C    0.0    0.0                 0.0
         C    0.5    0.28867513459481   0.0
      End

      Lattice
         1.0  0.0                 0.0
         0.5  0.86602540378443   0.0
      End
   End

   Engine DFTB
      ResourcesDir Dresden
      Model DFTB0
      KSpace
        Type Symmetric
        Symmetric KInteg=9
      End
   EndEngine

EOF

echo ""
echo "Begin TOC of result file"

$ADFBIN/dmpkf -n 1 graphene.results/dftb.rkf --toc

echo "End TOC of result file"
```

7.7.2 Example: Phonons with isotopes

Download Phonons_Isotopes.run

```
#! /bin/sh


# ====================================
# Phonons with default nuclear masses:
# ====================================

AMS_JOBNAME=defmasses $ADFBIN/ams << EOF

   Task SinglePoint

   Properties
      Phonons True
   End

   NumericalPhonons
      StepSize 0.01
      SuperCell
         4
      End
      Parallel nCoresPerGroup=1
   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"
$ADFBIN/dmpkf -n 1 defmasses.results/dftb.rkf --toc
echo "End TOC of result file"


# =========================================================
# Phonons with two deuterium atoms (via the AtomMasses key)
# =========================================================

AMS_JOBNAME=usermasses $ADFBIN/ams << EOF

   Task SinglePoint

   Properties
      Phonons true
   End

   NumericalPhonons
      StepSize 0.01
      SuperCell
         4
      End
      Parallel nCoresPerGroup=1
   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.d  -1.146891017  -2.054507222     -0.4803683019
           H.d  -1.146890987  -1.483665974      1.185797304
       End

       AtomMasses
           H.d 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"
$ADFBIN/dmpkf -n 1 usermasses.results/dftb.rkf --toc
echo "End TOC of result file"
```

7.7.3 Example: Elastic tensor

Download ElasticTensor.run

```
#! /bin/sh


# === Diamond ===

AMS_JOBNAME=Diamond $ADFBIN/ams << EOF

   Task GeometryOptimization

   Properties
       ElasticTensor Yes
   End

   # Maximum possible parallelism at the driver level
   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   End
   ElasticTensor
      Parallel nCoresPerGroup=1
   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-4
   End

   Engine DFTB
       Model DFTB
       ResourcesDir DFTB.org/mio-1-1
       KSpace
           Type Symmetric
           Symmetric KInteg=3
       End
   EndEngine

EOF


# === Boron-Nitride sheet ===

# 3x3 super-cell, no k-space sampling

AMS_JOBNAME=BN_sheet $ADFBIN/ams << EOF

   Task GeometryOptimization

   Properties
       ElasticTensor Yes
   End

   # Maximum possible parallelism at the driver level
   NumericalDifferentiation
      Parallel nCoresPerGroup=1
   End
   ElasticTensor
      Parallel nCoresPerGroup=1
   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
      ResourcesDir DFTB.org/matsci-0-3
   EndEngine

EOF


# === Polyoxyethylene ===

# primitive cell with k-space sampling

AMS_JOBNAME=Polyoxyethylene $ADFBIN/ams << EOF

   Task GeometryOptimization

   Properties
      ElasticTensor Yes
   End

   ElasticTensor
      StrainStepSize 0.002
      MaxGradientForGeoOpt 2.0e-4
      Parallel nCoresPerGroup=1
   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
      ResourcesDir DFTB.org/3ob-3-1
      KSpace
          Type Symmetric
          Symmetric KInteg=5
      End
   EndEngine

EOF


# Note: the elastic tensor is also printed to standard output.

echo ""
echo "Extract the elastic tensor of Diamond from the rkf file:"
$ADFBIN/adfreport Diamond.results/dftb.rkf -r "AMSResults%ElasticTensor#12.4f##6"

echo ""
echo "Extract the elastic tensor of Boron-Nitride from the rkf file:"
$ADFBIN/adfreport BN_sheet.results/dftb.rkf -r "AMSResults%ElasticTensor#12.4f##3"

echo ""
echo "Extract the elastic tensor of Polyoxyethylene from the rkf file:"
$ADFBIN/adfreport Polyoxyethylene.results/dftb.rkf -r "AMSResults%ElasticTensor#12.4f#
↪#1"
```

CHAPTER

EIGHT

APPENDICES

8.1 Environment variables

The behaviour of AMS 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. Not 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 behaviour 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.

8.2 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.

8.3 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 [True | False]
   ForceContinousPES [True | False]
   IgnoreGradientsRequest [True | False]
   IgnoreStressTensorRequest [True | False]
   RandomFailureChance float
End
```

EngineDebugging

Type Block

Description This block contains some options useful for debugging the computational engines.

CheckInAndOutput

Type Bool

Default value False

Description Enables some additional checks on the input and output of and engine, e.g. for NaN values.

ForceContinousPES

Type Bool

Default value False

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 False

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.

`IgnoreStressTensorRequest`

Type Bool

Default value False

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.

`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.

CHAPTER

NINE

REQUIRED CITATIONS

9.1 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 2019, 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, 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 101) 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.

9.2 Feature references

9.2.1 Vibrational analysis

Mode tracking

The following references must be cited, in addition to the standard engine citations, if you publish results obtained using this Mode Tracking module:

1. 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)
2. 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)
3. 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)

If you ran calculations using the Intensity Tracking options, use the following citations instead:

1. 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)
2. 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

The following reference must be cited, in addition to the standard engine citations, if you publish results obtained using the vibrational mode refinement module:

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*, Submitted

CHAPTER

TEN

REFERENCES

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)
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)
5. E. Bitzek, P. Koskinen, F. Gähler, M. Moseler and P. Gumbsch, *Structural Relaxation Made Simple*, Physical Review Letters 97, 170201 (2006) (https://doi.org/10.1103/PhysRevLett.97.170201)
6. T. Weymuth, M.P. Haag, K. Kiewisch, S. Luber, 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)
7. 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)
8. 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)
9. 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)
10. 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)
11. 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)
12. 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), 6878–6882 (2018) (https://doi.org/10.1021/acs.jpclett.8b02963)
13. 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)
14. 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)

15. L. Deng, T. Ziegler and L. Fan, *A combined density functional and intrinsic reaction coordinate study on the ground state energy surface of H_2 CO*, Journal of Chemical Physics 99, 3823 (1993) (https://doi.org/10.1063/1.466129)
16. 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)
17. (a) 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)
18. (a) 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)
19. (a) 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)

10.1 External programs and libraries

Click here for the list of programs and/or libraries used in the Amsterdam Modeling Suite. On some platforms optimized libraries have been used and/or vendor specific MPI implementations.

CHAPTER

ELEVEN

KEYWORDS

11.1 Links to manual entries

- *MolecularDynamics* (page 34)

11.2 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.

`Block`

Type String

Recurring True

Description Name of the block to constrain as specified in the atom tag within 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).

`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.

`ElasticTensor`

Type Block

Description Options for numerical evaluation of the elastic tensor.

`MaxGradientForGeoOpt`

Type Float

Default value 0.0001

Unit Hartree/Angstrom

Description Maximum nuclear gradient for the relaxation of the internal degrees of freedom of strained systems.

`Parallel`

Type Block

Description The evaluation of the elastic tensor via numerical differentiation is an embarrassingly parallel problem. Double parallelization 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

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

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.

`EngineDebugging`

Type Block

Description This block contains some options useful for debugging the computational engines.

`CheckInAndOutput`

Type Bool

Default value False

Description Enables some additional checks on the input and output of and engine, e.g. for NaN values.

`ForceContinousPES`

Type Bool

Default value False

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 False

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.

`IgnoreStressTensorRequest`

Type Bool

Default value False

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.

`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.

`EngineRestart`

Type String

Description The path to the file from which to restart the engine.

`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

Description Number of GCMC moves.

`MapAtomsToOriginalCell`

Type Bool

Default value True

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

Description The max distance to other atoms of the system when adding the molecule.

`MinDistance`

Type Float

Default value 0.3

Unit Angstrom

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

`Molecule`

Type Block

Recurring True

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: cmpot = Mu_O(T,P) = 1/2*Mu_O2(T,P) = 1/2 * [Mu_ref(T,P_ref) + kT*Log(P/Pref) - E_diss] where the reference chemical potential [Mu_ref(T,P_ref)] is the experimentally determined chemical potential of O2 at T and Pref; kT*Log(P/Pref) is the pressure correction to the free energy, and E_diss is the dissociation energy of the O2 molecule.

`NoAddRemove`

Type Bool

Default value False

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

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

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

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 True

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 Vnew = exp(random(-1:1)*VolumeChangeMax)*Vold

`VolumeOption`

Type Multiple Choice

Default value Free

Options [Free, Total, Accessible, FreeAccessible]

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 = totalVolume - occupiedVolume - NonAccessibleVolume; Total: V = totalVolume; Accessible: V = AccessibleVolume; FreeAccessible: V = AccessibleVolume - 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 True

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.

`ConjugateGradients`

Type Block

Description Configures details of the conjugate gradients geometry optimizer.

`Step`

Type Block

Description

`MinRadius`

Type Float

Default value 0.0

Description Minimum value for the trust radius.

`TrustRadius`

Type Float

Default value 0.2

Description Initial value of the trust radius.

`Convergence`

Type Block

Description Convergence is monitored for two items: the energy and the Cartesian gradients. Convergence criteria can be specified separately for each of these items.

`Energy`

Type Float

Default value 1e-05

Unit Hartree

Description The criterion for changes in the energy.

`Gradients`

Type Float

Default value 0.001

Unit Hartree/Angstrom

Description The criterion for changes in the gradients.

`Step`

Type Float

Default value 0.001

Unit Angstrom

Description The maximum Cartesian step allowed for a converged geometry.

`CoordinateType`

Type Multiple Choice

Default value Auto

Options [Auto, Delocalized, Cartesian]

Description Select the type of coordinates in which to perform the optimization. If 'Auto', delocalized coordinates will be used for molecular systems, while Cartesian coordinates will be used for periodic systems. Optimization in delocalized coordinates [Delocalized] can only be used for geometry optimizations or transition state searches of molecular systems with the Quasi-Newton method. The experimental SCMGO optimizer supports [Delocalized] coordinates for both molecular and periodic systems.

`FIRE`

Type Block

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

`MapAtomsToUnitCell`

Type Bool

Default value False

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.

`RejectEnergyIncrease`

Type Bool

Default value False

Description Makes the optimizer reject steps that increase the energy. This can speed up convergence, but often causes the optimizer to get stuck on small discontinuities on the potential energy surface. It is therefore disabled by default.

`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.

`InitialHessian`

Type Block

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

`File`

Type String

Description KF file containing the initial Hessian. 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]

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).

`KeepIntermediateResults`

Type Bool

Default value False

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

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, ConjugateGradients]

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), 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 False

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

`Pressure`

Type Float

Default value 0.0

Description Optimize the structure under pressure (this will only have an effect if you are optimizing the lattice vectors). Currently only working in combination with the Quasi-Newton optimizer. For phase transitions you may consider disabling or breaking the symmetry.

`PressureUnit`

Type Multiple Choice

Default value GPa

Options [a.u., Pascal, GPa, atm, bar, kbar]

Description The unit for pressure to be used for optimizations under pressure

`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.

`SCMGO`

Type Block

Description Configures details SCMGO.

`ContractPrimitives`

Type Bool

Default value True

Description Form non-redundant linear combinations of primitive coordinates sharing the same central atom

`NumericalBMatrix`

Type Bool

Default value False

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 True

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

`logSCMGO`

Type Bool

Default value False

Description Verbose output of SCMGO internal data

`testSCMGO`

Type Bool

Default value False

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

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

`Step`

Type Float

Default value 0.001

Unit Angstrom

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

`CoordinateType`

Type Multiple Choice

Default value Cartesian

Options [Cartesian, Delocalized]

Description Select the type of coordinates in which to perform the optimization. Note that the Delocalized option should be considered experimental. Besides, it is not possible to use delocalized coordinates for periodic systems.

`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

Description If 'Type' is set to 'FromFile' then in this key you should specifiy the RKF file containing the initial Hessian. 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]

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

`KeepConvergedResults`

Type Bool

Default value True

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

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

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

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 False

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

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

Description IRC step size in mass-weighted coordinates, sqrt(amu)*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.

`Section`

Type String

Default value Molecule

Description The section on the KF file from which to load the system.

`ModeRefinement`

Type Block

Description Input data for ModeRefinement tasks.

`Displacement`

Type Float

Default value 0.001

Description Step size for finite difference calculation of frequencies and IR intensities.

`ModePath`

Type String

Description Path to a .rkf file containing the modes which are to be scanned. Which modes will be refined is selected using the criteria from the [ModeSelect] block.)

`ModeSelect`

Type Block

Description Pick which modes to refine from those read from file.

`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 refined. (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 refined. (2 numbers: a upper and a lower bound.)

`Full`

Type Bool

Default value False

Description Refine all modes.

`HighFreq`

Type Integer

Description Refine the N modes with the highest frequencies.

`HighIR`

Type Integer

Description Refine 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 refined. (2 numbers: a upper and a lower bound.)

`ImFreq`

Type Bool

Default value False

Description Refine all modes with imaginary frequencies.

`LowFreq`

Type Integer

Description Refine the N modes with the lowest frequencies. (Includes imaginary modes which are recorded with negative frequencies.)

`LowFreqNoIm`

Type Integer

Description Refine the N modes with the lowest non-negative frequencies. (Imaginary modes have negative frequencies and are thus omitted here.)

`LowIR`

Type Integer

Description Refine the N modes with the smallest IR intensities.

`ModeNumber`

Type Integer List

Description Indices of the modes to refine.

`ScanModes`

Type Bool

Default value False

Description 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.

`ModeScanning`

Type Block

Description Input data for the ModeScanning task.

`Displacement`

Type Float

Default value 0.001

Description Step size for finite difference calculation of frequencies and IR intensities.

`ModePath`

Type String

Description Path to a .rkf file containing the modes which are to be scanned. Which modes will be scanned is selected using the criteria from the [ModeSelect] block.)

`ModeSelect`

Type Block

Description Pick which modes to scan from those read from file.

`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 scanned. (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 scanned. (2 numbers: a upper and a lower bound.)

`Full`

Type Bool

Default value False

Description Scan all modes.

`HighFreq`

Type Integer

Description Scan the N modes with the highest frequencies.

`HighIR`

Type Integer

Description Scan 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 scanned. (2 numbers: a upper and a lower bound.)

`ImFreq`

Type Bool

Default value False

Description Scan all modes with imaginary frequencies.

`LowFreq`

Type Integer

Description Scan the N modes with the lowest frequencies. (Includes imaginary modes which are recorded with negative frequencies.)

`LowFreqNoIm`

Type Integer

Description Scan the N modes with the lowest non-negative frequencies. (Imaginary modes have negative frequencies and are thus omitted here.)

`LowIR`

Type Integer

Description Scan the N modes with the smallest IR intensities.

`ModeNumber`

Type Integer List

Description Indices of the modes to scan.

`ModeTracking`

Type Block

Description Input data for ModeTracking task.

`Displacement`

Type Float

Default value 0.01

Description Step size (in Bohr) for finite difference calculation of frequencies and IR intensities during mode tracking iterations.

`HessianGuess`

Type Multiple Choice

Default value UFF

Options [Unit, File, UFF, Inline]

Description Sets how to obtain the guess for the Hessian used in the preconditioner.

`HessianInline`

Type Non-standard block

Description Initial guess for the (non-mass-weighted) Hessian in a 3N x 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].

`MassWeighInlineMode`

Type Bool

Default value True

Description The supplied modes must be mass-weighed. This tells the program to mass-weigh the supplied modes in case this has not yet been done. (True means the supplied modes will be mass-weighed by the program, e.g. the supplied modes are non-mass-weighed.)

`MaxIterations`

Type Integer

Description Maximum number of allowed iterations.

`ModeInline`

Type Non-standard block

Recurring True

Description Coordinates of the mode which will be tracked in a N x 3 block (same as for atoms), used when [TrackedMode] = [Inline]. Rows must be ordered in the same way as in the [System%Atoms] block.

`ModePath`

Type String

Description Path to a .rkf file containing the modes which are to be tracked. Which modes will be refined is selected using the criteria from the [ModeSelect] block.)

`ModeSelect`

Type Block

Description Pick which modes to track from modes generated from Hessian or read from file.

`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 tracked. (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 tracked. (2 numbers: a upper and a lower bound.)

`Full`

Type Bool

Default value False

Description Track all modes.

`HighFreq`

Type Integer

Description Track the N modes with the highest frequencies.

`HighIR`

Type Integer

Description Track 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 tracked. (2 numbers: a upper and a lower bound.)

`ImFreq`

Type Bool

Default value False

Description Track all modes with imaginary frequencies.

`LowFreq`

Type Integer

Description Track the N modes with the lowest frequencies. (Includes imaginary modes which are recorded with negative frequencies.)

`LowFreqNoIm`

Type Integer

Description Track the N modes with the lowest non-negative frequencies. (Imaginary modes have negative frequencies and are thus omitted here.)

`LowIR`

Type Integer

Description Track the N modes with the smallest IR intensities.

`ModeNumber`

Type Integer List

Description Indices of the modes to track.

`ScanModes`

Type Bool

Default value False

Description 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.

`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.

`TrackedMode`

Type Multiple Choice

Default value File

Options [Inline, File, Hessian]

Description Set how the initial guesses for the modes are supplied.

`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.

`UpdateMethod`

Type Multiple Choice

Default value JD

Options [JD, D]

Description Chooses the method for expanding the Krylov subspace: (D) Davidson or (JD) vdVorst-Sleijpen variant of Jacobi-Davidson.

`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

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. The total momentum of the added molecule is not conserved.

`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 ADFinput, 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.

`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 False

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. ADFinput 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 TemperatureSigma.

`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 ADFinput 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 False

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.

`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

Description Specifies the time constant of the barostat.

`Type`

Type Multiple Choice

Default value None

Options [None, Berendsen, MTK]

Description Selects the type of the barostat.

`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.

`CVHD`

Type Block

Recurring True

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

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

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 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 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

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

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

rmax

Type Float

Unit Angstrom

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

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 False

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

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

`HeatExchange`

Type Block

Recurring True

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 transfered 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).

`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. 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.

FirstAtom

Type Integer

Description Index of the first atom of the region. This key is ignored if the [Box] block is present.

LastAtom

Type Integer

Description Index of the last atom of the region. 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).

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.

`FirstAtom`

Type Integer

Description Index of the first atom of the region. This key is ignored if the [Box] block is present.

`LastAtom`

Type Integer

Description Index of the last atom of the region. 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

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. ADFinput will use the thermostat temperature as default.

`Type`

Type Multiple Choice

Default value Random

Options [Zero, Random, FromFile, Input]

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 ADFinput.

`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

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

`PRD`

Type Block

Description This block is used for Parallel Replica Dynamics simulations.

`BondChange`

Type Block

Recurring True

Description Detect changes to the bonding topology and bond orders returned by the engine.

`ChangeThreshold`

Type Float

Default value 0.5

Description Trigger an event when the bond order of a bond changes from the reference state by more than this value.

`DissociationThreshold`

Type Float

Default value 0.3

Description Trigger an event when a bond dissociates (its bond order drops below this value while it was above FormationThreshold in the reference state).

`FormationThreshold`

Type Float

Default value 0.8

Description Trigger an event when a new bond forms (its bond order exceeds this value while it was below DissociationThreshold in the reference state).

`CorrelatedSteps`

Type Integer

Default value 100

Description How many steps to wait for correlated events after detecting an initial event.

`DephasingSteps`

Type Integer

Default value 100

Description Spend this many steps dephasing the individual replicas after an event.

`MolCount`

Type Block

Recurring True

Description Detect changes to the molecular composition of the system.

`nReplicas`

Type Integer

Default value 1

Description Number of replicas to run in parallel.

`Plumed`

Type Block

Description Input for PLUMED.

`Input`

Type Non-standard block

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

`Preserve`

Type Block

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

`AngularMomentum`

Type Bool

Default value True

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

`CenterOfMass`

Type Bool

Default value False

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 True

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 False

Description Print the chemical system before and after the simulation.

`Velocities`

Type Bool

Default value False

Description Print the atomic velocities before and after the simulation.

`RemoveMolecules`

Type Block

Recurring True

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.

`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 a 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 atomic units.

`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.

`SinkBox`

Type Block

Description Part of the simulation box where molecules will be removed. By default, molecules matching the formula will be removed regardless of their location. If this block is present a molecule will 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 atomic units.

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.

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.

SwapFrequency

Type Integer

Default value 100

Description Attempt an exchange every N steps.

TemperatureFactor

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.

`nReplicas`

Type Integer

Default value 1

Description Number of replicas to run in parallel.

`Restart`

Type String

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]

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

Description Number of individual thermostats forming the NHC thermostat

`Duration`

Type Integer List

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

`FirstAtom`

Type Integer

Default value 1

Description Index of the first atom to be thermostatted

`LastAtom`

Type Integer

Default value 0

Description Index of the last atom to be thermostatted. A value of zero means the last atom in the system.

`Tau`

Type Float

Unit Femtoseconds

Description The time constant of the thermostat.

`Temperature`

Type Float List

Unit Kelvin

Description The target temperature of the thermostat.

`Type`

Type Multiple Choice

Default value None

Options [None, Berendsen, NHC]

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.

`SamplingFreq`

Type Integer

Default value 100

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 is greater than 0 then a temperature profile will be generates along each of the three unit cell axes. By default, no profile is generated.

`NormalModes`

Type Block

Description Configures details of a normal modes calculation.

`UseSymmetry`

Type Bool

Default value True

Description Whether or not to exploit the symmetry of the system in the normal modes calculation.

`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 Numerical differentiation is an embarrassingly parallel problem. Double parallelization 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

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

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.

`UseSymmetry`

Type Bool

Default value True

Description Whether or not to exploit the symmetry of the system for numerical differentiations.

`NumericalPhonons`

Type Block

Description Configures details of a numerical phonons calculation.

`DoubleSided`

Type Bool

Default value True

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 Computing the phonons via numerical differentiation is an embarrassingly parallel problem. Double parallelization 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. Keep in mind that the displacements for a phonon calculation are done on a super-cell system, so that every task requires more memory than the central point calculated using the primitive cell.

`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

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.

`UseSymmetry`

Type Bool

Default value True

Description Whether or not to exploit the symmetry of the system in the phonon calculation.

`PESScan`

Type Block

Description Configures the details of the potential energy surface scanning task.

`CalcPropertiesAtPESPoints`

Type Bool

Default value False

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 True

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 (x|y|z) followed by two real numbers delimiting the transit range.

`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.

`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 False

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.

`ElasticTensor`

Type Bool

Default value False

Description Whether or not to calculate the elastic tensor.

`Gradients`

Type Bool

Default value False

Description Whether or not to calculate the gradients.

`Hessian`

Type Bool

Default value False

Description Whether or not to calculate the Hessian.

`Molecules`

Type Bool

Default value False

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 False

Description Whether or not to calculate the normal modes of vibration (and of molecules the corresponding Ir intensities.)

`Other`

Type Bool

Default value True

Description Other (engine specific) properties. Details are configured in the engine block.

`Phonons`

Type Bool

Default value False

Description Whether or not to calculate the phonons for periodic systems.

`SelectedAtomsForHessian`

Type Integer List

Description Compute the Hessian matrix elements only for the atoms defined in this list (index). If not specified, the Hessian will be computed for all atoms.

`StressTensor`

Type Bool

Default value False

Description Whether or not to calculate the stress tensor.

`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.

`Symmetry`

Type Block

Description Specifying details about the details of symmetry detection and usage.

`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.

`AtomMasses`

Type Non-standard block

Description User defined atomic masses.

`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 by used by MM engines.

`Charge`

Type Float

Default value 0.0

Description The system's total charge in atomic units (only for non-periodic systems).

`FractionalCoords`

Type Bool

Default value False

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

`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.

`RandomizeCoordinates`

Type Float

Default value 0.0

Unit Angstrom

Description Apply a random noise to the atomic coordinates. This can be useful if you want to deviate from an ideal symmetric geometry.

`RandomizeStrain`

Type Float

Default value 0.0

Description Apply a random strain to the system. 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.

`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)).

Task

Type Multiple Choice

Options [SinglePoint, GeometryOptimization, TransitionStateSearch, PESScan, MolecularDynamics, ModeScanning, ModeRefinement, ModeTracking, GCMC, IRC]

Description This key is used to specify the computational task to perform.

Thermo

Type Block

Description Options for thermodynamic properties (assuming an ideal gas). The properties are computed for 'nSteps' temperatures in the range [TMin,TMax].

Pressure

Type Float

Default value 1.0

Unit atm

Description The pressure at which the thermodynamic properties are computed.

TMax

Type Float

Default value 298.15

Unit Kelvin

Description Maximum value for the temperature range.

TMin

Type Float

Default value 298.15

Unit Kelvin

Description Minimum value for the temperature range.

nSteps

Type Integer

Default value 1

Description The number of temperatures in the range [TMin,TMax].

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).

UseSymmetry

Type Bool

Default value True

Description Whether to use the system’s symmetry at the application level.

INDEX