AMS Manual

ADF Modeling Suite 2018

www.scm.com

Sep 18, 2018

CONTENTS

CHAPTER

ONE

GENERAL

1.1 Overview

AMS is the new driver program 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 Motivation and progress

The Amsterdam Modeling Suite has grown substantially over the last decade, and in the 2017 release includes 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 has, however, led to a certain degree of unnecessary inhomogenity within the suite: The input for the same task, e.g. a geometry optimization, can differ quite a lot between the different progams 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 has also led to a certain level of feature fragmentation, where some features are 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 introducing the AMS driver program with the 2018 release of the suite. The idea of this reorganization is to have only a single program called AMS 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, AMS also allows running *external programs as an engine* (page 45) 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. GCMC) 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 45). 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.

1.3 Input, execution and output

With the introduction of AMS in the 2018 release of the Amsterdam Modeling Suite, there are 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 51) 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 descrption of the engine inputs into separate *engine specific manuals* (page 43), 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 91) 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 36) 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
   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

Description Specification of the chemical system.

`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 geometry from an file instead of Atoms and Lattice and 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 91) 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).

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
   File string
   Section string
End
```

LoadSystem

Type Block

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 43) 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 36). 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 36) 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 61) 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 36) 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 64)

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)
SCMGO	No	Yes	Delocalized
FIRE	Fixed atoms and coordinates	Yes	Cartesian
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 95)], 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 95)]. 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 43), the overhead of the Quasi-Newton optimizer is likely the bottleneck of the calculation, and more light-weight optimizers like *FIRE* (page 19) 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 23), the *transition state search* (page 22) as well as the calculation of the *elastic tensor* (page 39).

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

Default value 0.2

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]
   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]

Description Selects the type of the initial model Hessian, or load the Hessian from the results of a previous calulation.

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 61).

SCMGO

The SCMGO optimizer is a new implementation of a Quasi-Newton style optimizer working in delocalized coordinates. In the 2018 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).

FIRE

The Fast Inertial Relaxation Engine [*5* (page 95)] 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 43). 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
      NMin integer
      alphaStart float
      dtMax float
      dtStart float
      fAlpha float
      fDec float
      fInc float
      strainMass float
   End
End
```

GeometryOptimization

FIRE

Type Block

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

NMin

Type Integer

Default value 5

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

alphaStart

Type Float

Default value 0.1

Description Steering coefficient.

dtMax

Type Float

Default value 1.25

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 Ficticious 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.

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 less robust, and has a larger overhead than *FIRE* (page 19), 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

TrustRadius

Type Float

Default value 0.2

Description Initial value of the trust radius.

3.2.3 Troubleshooting

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 43) 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 19) optimizer, as it is internally relatively simple, stable and has the nice property that the energy can never increase during an optimization.

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 61) 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 23) 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 61) 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 61) 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 36) selected in `Properties` block.

3.4.1 Troubleshooting

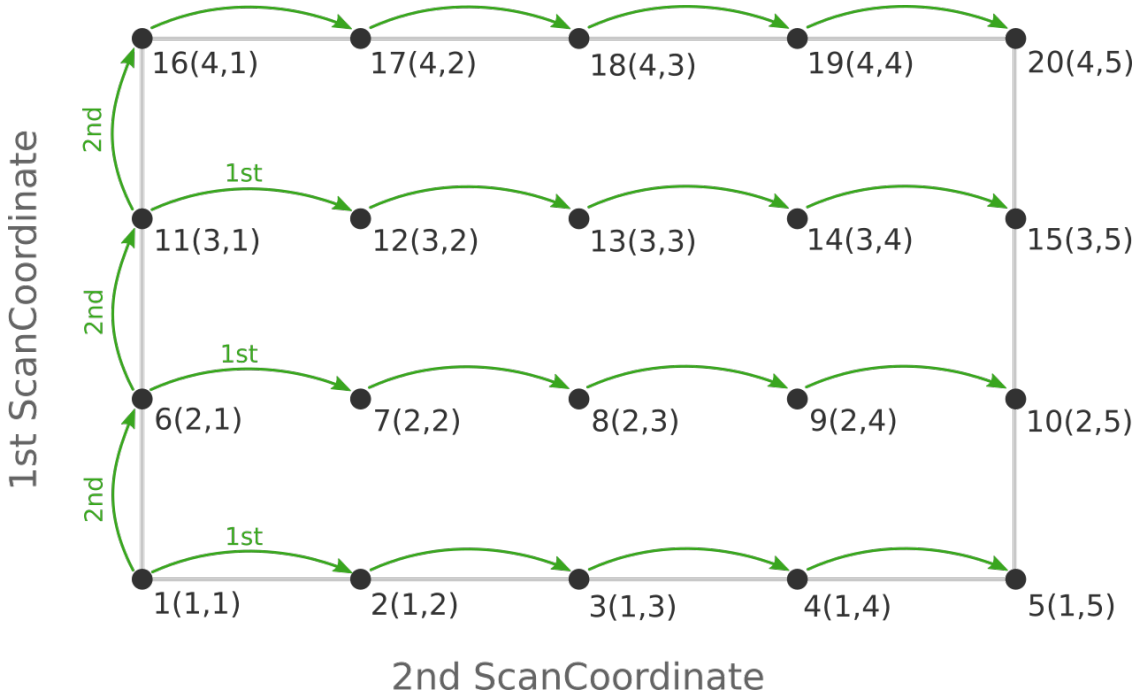
Technically all PES scan calculations are conducted as a series of geometry optimizations with constraints for the scanned coordinates, where the value of the constraint varies slowly through the scanned range. In this way every sampled point on the potential energy surface corresponds to a particular 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.

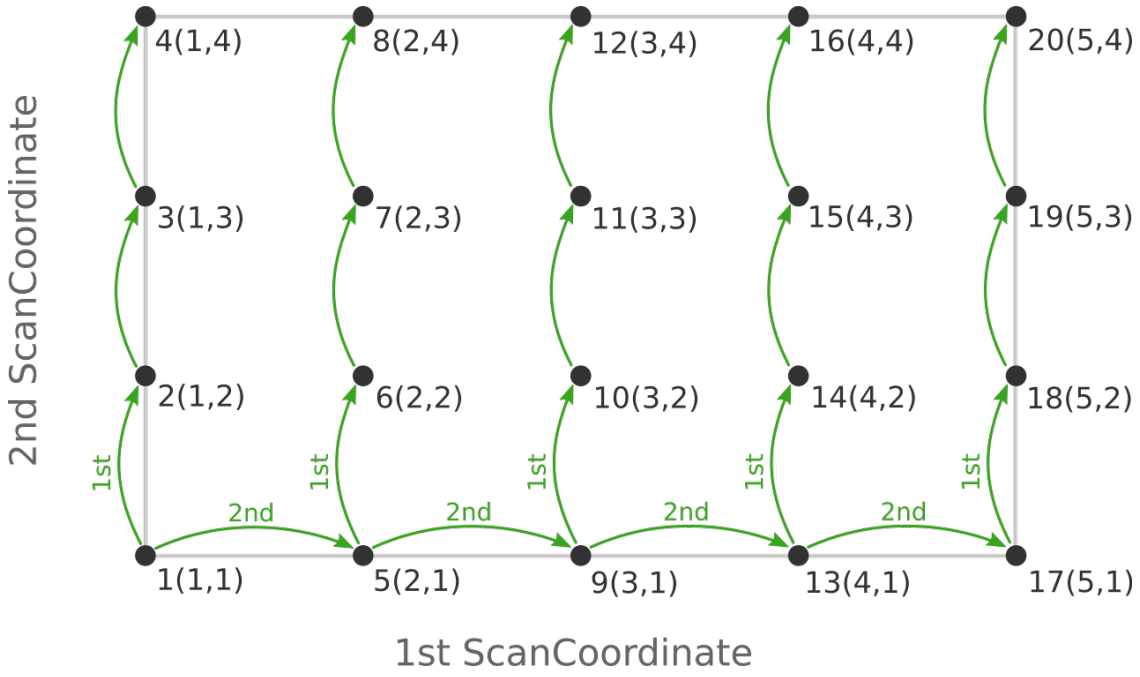
While tweaking the geometry optimizer's settings can sometimes help with convergence problems, these problems can also be easily caused by errors in the user input.

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 neighbouring points.

3.5 Molecular dynamics

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

See also:

Examples (page 61)

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
   Barostat
      BulkModulus float
      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
   CalcPressure [True | False]
   Checkpoint
      Frequency integer
   End
   InitialVelocities
      File string
      Temperature float
      Type [Zero | Random | FromFile | Input]
      Values # Non-standard block. See details.
         ...
      End
   End
   NSteps integer
```

```
   Preserve
      AngularMomentum [True | False]
      CenterOfMass [True | False]
      Momentum [True | False]
   End
   Print
      System [True | False]
      Velocities [True | False]
   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
   End
End
```

3.5.1 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.

3.5.2 (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 Zero

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 key [MolecularDynamics%InitialVelocities%Values].

Values

Type Non-standard block

Description This block specifies the velocity of each atom 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.

3.5.3 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.

`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.

3.5.4 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.

`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.

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:

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

where V_0 is the volume of the unit cell (for 2D periodic system V_0 is the area of the unit cell, and for 1D periodic system V_0 is the 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.)

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.

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 86)

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 parallelly working groups.

`nCoresPerGroup`

Type Integer

Description Number of cores in each parallelly 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 83), *Example: Phonons with isotopes* (page 84) 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 54), 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 parallelly working groups. 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 parallelly 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.

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 27) 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 2018 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 45)** A flexible scripting interface that allows advanced users to use external atomistic modeling programs as engines in AMS.
- ***LennardJones*** **(page 49)** 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 91) 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:

- Spaces (blanks) between the key, the equal sign and the value are not allowed:

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

  # This is NOT OK:
  BlockName KeyName1 = value1 KeyName2= value2
  ```

- The compact notation can be used only for following types of keys: bool, integer, float and multiple_choice. It should **not** be used for sting, integer_list and float_list keys
- The compact notation cannot be used for blocks with headers
- input units cannot be defined in compact notation

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` (`a.u.`), `eV`, `kJ/mol`, `kcal/mol`, `cm^-1`
- **Length**: `Bohr` (`a.u.`), `Angstrom` (`A`), `nm`, `pm`

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 43). 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 27) 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**. In the 2018 release of AMS, double parallelization is used for the calculation of the *PES point properties* (page 36) which are derivatives, if these need to be done numerically:

- Numerical calculation of *forces / nuclear gradients* (page 37). 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 38) 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 38) 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 39). 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 41). 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 39) 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:

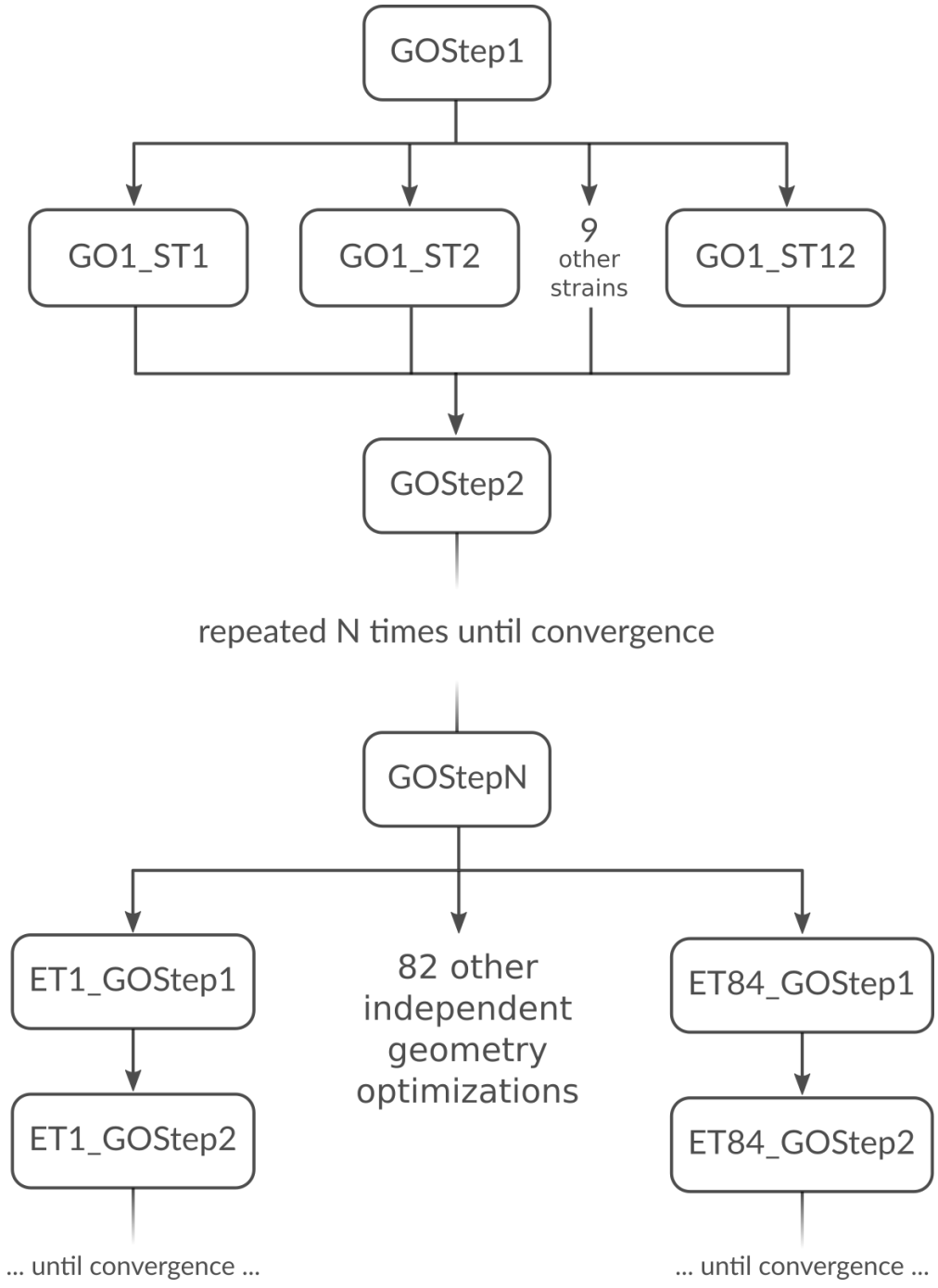
```
Task GeometryOptimization

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 38) 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:



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
   # (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
       Periodic KSpace=5
   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  0.0  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  0.0  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
      Periodic KSpace=3
   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 then 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
      Periodic KSpace=3
   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
       Periodic KSpace=3
   EndEngine

EOF

echo "Extract energy from the rkf file using adfreport:"
$ADFBIN/adfreport TS.results/dftb.rkf -r "AMSResults%Energy"
```

7.3 PES scan

7.3.1 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
```

7.3.2 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
```

7.4 Molecular dynamics

7.4.1 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
   Repulsion
       forcePolynomial true
   End
   DispersionCorrection Auto
EndEngine

eor
```

7.4.2 Example: MD for a box of water

Download H2O_nreac.run

7.5 PES point properties

7.5.1 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
      Periodic kSpace=9
   EndEngine

EOF

echo ""
echo "Begin TOC of result file"

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

echo "End TOC of result file"
```

7.5.2 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
      Periodic kSpace=9
   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
      Periodic KSpace=9
   EndEngine

EOF

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

7.5.3 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
       Periodic kSpace=3
   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
      Periodic kSpace=5
   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
   ForceContinousPES [True | False]
   IgnoreGradientsRequest [True | False]
   IgnoreStressTensorRequest [True | False]
End
```

EngineDebugging

Type Block

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

ForceContinousPES

Type Bool

Default value False

Description If this option is set, the engine will always run in contiuous 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 continously 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.

CHAPTER

NINE

REQUIRED CITATIONS

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 2018, 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, P. Philipsen, T. Soini

Note that the engine used for a particular calculation might require you to include other references. Please refer to the specific *engine manuals* (page 45) for required citations.

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)

CHAPTER

ELEVEN

KEYWORDS

11.1 Links to manual entries

- *ElasticTensor* (page 40)
- *EngineDebugging* (page 92)
- *GeometryOptimization* (page 12)
- *LoadSystem* (page 8)
- *MolecularDynamics* (page 27)
- *NumericalPhonons* (page 41)
- *Print* (page 92)
- *Properties* (page 37)
- *System* (page 5)
- *TransitionStateSearch* (page 23)

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 contstrain 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 parallelly working groups.

nCoresPerGroup

Type Integer

Description Number of cores in each parallelly 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.

ForceContinousPES

Type Bool

Default value False

Description If this option is set, the engine will always run in contiuous 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 continously 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.

EngineRestart

Type String

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

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

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).

NMin

Type Integer

Default value 5

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

alphaStart

Type Float

Default value 0.1

Description Steering coefficient.

dtMax

Type Float

Default value 1.25

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 Ficticious 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]

Description Selects the type of the initial model Hessian, or load the Hessian from the results of a previous calulation.

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

Default value 0.2

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.

LoadEngine

Type String

Description The path to the file from which to load the engine configuration. Replaces the Engine block.

LoadSystem

Type Block

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.

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.

Barostat

Type Block

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

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.

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

Description Write the MD state and engine-specific data to the respective .rkf files once every N steps. The default is the total number of steps divided by 4.

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 Zero

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 key [MolecularDynamics%InitialVelocities%Values].

`Values`

Type Non-standard block

Description This block specifies the velocity of each atom 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.

`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.

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.

ScaleFrequency

Type Integer

Description Optional parameter used only by the Scale thermostat. If specified, the thermostat will be applied every N steps, using that step's ensemble temperature and the specified thermostat temperature to compute the scaling factor. If not specified, the thermostat will be applied at every step, using the mean temperature of the ensemble and the specified thermostat temperature to compute the scaling factor.

Tau

Type Float

Unit Femtoseconds

Description The time constant of the thermostat. Mandatory in case the Berendsen or NHC thermostat is used.

Temperature

Type Float List

Unit Kelvin

Description The target temperature of the thermostat. This key is mandatory for the Scale and Berendsen thermostats.

Type

Type Multiple Choice

Default value None

Options [None, Scale, 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

Description Write the the molecular geometry (and possibly other properties) to the .rkf file once every N steps. The default is the total number of steps divided by 1000.

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 0.0001

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 parallelly working groups.

nCoresPerGroup

Type Integer

Description Number of cores in each parallelly 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.

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 parallelly working groups. 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 parallelly 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 neighbouring 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).

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.

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.

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

Description Specification of the chemical system.

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 geometry from an file instead of Atoms and Lattice and 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).

Task

Type Multiple Choice

Options [SinglePoint, GeometryOptimization, TransitionStateSearch, PESScan, MolecularDynamics, Idle, SteepestDescent, ModeTracking, ScanFreq, TestEngine, TestSymmetry]

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

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

P

Q

R

S

T

U

V

X

Y