Manual for sxextopt - external structure optimizer

Christoph Freysoldt, freysoldt@mpie.de

April 3, 2017

Contents

1 Description

`sxextopt` provides access to the structure optimizers of the SPHInX code to other codes via the atomic structure algorithm protocol (ASAP).

1.1 The atomic structure algorithm protocol (ASAP)

The main idea of the ASAP is to separate potential-energy surface calculations (e.g. from DFT codes) from algorithms that explore the potential energy surface. The DFT code and the structure algorithm (SA) code communicate via two named pipes, the control channel and the response channel, see Fig. 1. The SA code writes requests such as '*set structure*', '*run*', '*get forces*' to the control channel. The DFT code reads from the control channel and writes requested data to the response channel. The ASAP defines the possible requests as well as the formats.

The full specification of the ASAP can be found in `'structAlgoProtocol.txt'`, which should accompany this manual.

`sxextopt` acts as the structure algorithm code. To use it, you (as a code developer) will need to implement the potential energy surface (PES) side of the protocol into your code. To use it, you (as a code user) will need to run the DFT code in 'ASAP' mode (check the manual).

1.2 Named pipes

Before you start the DFT code and `sxextopt`, you must create the named pipes for the control channel and the response channels, respectively. You must further make sure that both codes use the correct file names.

Named pipes are created in UNIX systems via `mkfifo`. For instance, to create a pipe called 'control':

```
mkfifo control
```

Named pipes reside in the filesystem like normal files, and can be technically opened, closed, written to, and read from like normal files. However, they work properly only if the file is opened twice, once for reading and once for writing.

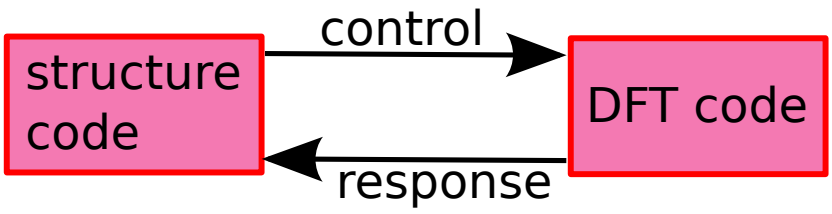

Figure 1: Schematic principle of the atomic structure algorithm protocol (ASAP): A structure algorithm code controls a DFT code via the control channel.

Whatever is written to the file, will appear exactly once for the reader (very much like normal pipes). The synchronization is taken care of by the UNIX system: as soon as the writer has written its data and *flushed* any internal buffers (usually with the `flush` function), the reading will be able to read.

Note that the UNIX system will block in a number of circumstances:

- When opening a file, until the other end is connected.
- When reading from the pipe, until something is written to it.
- When writing to a pipe, when the reading side has not yet read all previous data and the UNIX-internal buffers are filled.

2 Options for the `sxextopt` executable

Option	argument	description
`--help`		show all the available options
`--log`		create a log file
`--input`	filename	input file (default: optim.sx)
`--control`	filename	control file (default: control)
`--response`	filename	response file (default: response)

The control and response pipe name setting from the command line override any settings in the input file.

3 Input file (`optim.sx`)

In the following, only the `ricQN` minimizer is documented. Further (but worse) minimizers are implemented, please contact me on details.

The input format is best explained starting from an **example:**

```
main {
    ricQN  {
        ric { maxDist = 5; withAngles; }
        maxSteps = 30;
        dEnergy  = 1e-5;
    }
}
```

The SPHInX input format is a structured, hierarchical format with a C-like syntax. It consists of named groups and parameters. The content of a **group** (such as `ricQN`) is enclosed in curly brackets {}. Groups may contain parameters, and other groups. **Parameters** (such as `maxSteps`) are assigned values with the equal sign. Parameter assignments must be followed by a semicolon (;). Flags are special parameters, that normally do not carry a value. A flag is set by specifying its name, followed by a semicolon. Flags are unset by assigning

the value 0. Some parameters/variables may be a vector (or a list). The values are comma-separated and enclosed by square brackets [].

Comments can be added by the // and /* */ syntax. A // comment extends until the rest of the line. A /* */ comment omits everything between the /* and */ markers.

The SPHInX parser supports basic algebraic expressions, such as adding, subtracting, multiplying etc. At the top level, additional variables may be set and used in algebraic expressions.

The following parameters may be set at the top level:

parameter	description
`controlFile`	name of the control pipe
`responseFile`	name of the response pipe

At the top level, there may be two groups: the `structure` group and the `main` group. The `structure` group contains the atomic positions in the SPHInX format (see Sec. 3.1). It is needed only to specify constraints. The `main group` contains the structure minimizer(s).

3.1 The `structure` group

The structure group specifies the atomic positions in SPHInX format. It is usually *not needed* because the starting structure can be obtained from the DFT code. However, the structure group allows to specify constraints, or control the symmetry.

Example:

```
structure {
   cell = 10.2 * [[0, 1/2, 1/2],
                  [1/2, 0, 1/2],
                  [1/2, 1/2, 0]];
   species {
      element="Si";
      atom { coords = [0,0,0]; relative; }
      atom { coords = [1/4,1/4,1/4]; relative; }
   }
}
```

The following parameters may be set:

parameter	description
`movable`	(flag) allow atoms to move. Default: all atoms are movable, unless any movable flag is used for any species/atom.
`movableX`	(flag) allow atoms to move in the x direction. Default: movable, unless movableY or movableZ are used.
`movableY`	(flag) allow atoms to move in the x direction. Default: movable, unless movableX or movableZ are used.
`movableZ`	(flag) allow atoms to move in the x direction. Default: movable, unless movableX or movableY are used.
cell	(required) The unit cell (in bohr). This is a list of the three basis vectors in Cartesian coordinates.

The `movable` flags are applied hierarchically. Settings at the structure level can be overridden at the species or atom level. Disabling a movable flag of a surrounding group is achieved by setting to 0, e.g. `movableY = 0;`

The structure group must contain at least one `species` group. It may contain a `symmetry` group.

3.1.1 The `species` group

The `species` group defines atomic positions for one chemical species. Atoms must be sorted by their chemical species. The order in the DFT code may be different than specified here, atoms read from the DFT code are mapped by `sxextopt` to the atoms in the structure group.

The `species` group may contain the `movable, movableX, movableY, movableZ` parameters as specified above. In addition it may contain the `element` parameter to indicate the chemical symbol, enclosed by double quotes.

The `species` group must contain at least one `atom` group.

3.1.2 The `atom` group

The atom group defines atomic positions for one atom. Atoms must be sorted by their chemical species. The order in the DFT code may be different than specified here, atoms read from the DFT code are mapped by `sxextopt` to the atoms in the structure group.

The `atom` group may contain the `movable, movableX, movableY, movableZ` parameters as specified above. In addition, the following parameters may be set:

parameter	description
`coords`	(required) The atomic coordinates as a 3-vector. Unless the `relative` flag is employed, the coordinates are Cartesian (in bohr).
`relative`	(flag) The coordinates are given relative to the unit cell vectors.
`movableLine`	(optional) The movement of the atom is restricted to a line. The value gives the direction of the line as a 3-vector.

3.1.3 `symmetry` group

The `symmetry` group (within the structure group) defines the rotational symmetries of the system around the origin of the coordinate system. If not given, the symmetries are determined automatically. However, non-chemical degrees of freedom (such as spins) may break the symmetry. As all forces / displacements are symmetrized, such a situation may require to set the symmetries by hand. Alternatively, **giving an empty symmetry group switches off symmetrization**.

The `symmetry` group contains multiple `operator` groups. Each operator group contains the parameter `S`, a Cartesian rotation matrix given row-wise. The symmetries must form a group.

3.2 The `ricQN` group

The `ricQN` group requests a quasi-Newton optimization with BFGS updates [1] of an on-the-fly optimized internal-coordinate based initial guess for the Hessian.

The following parameters may be set:

parameter	description
`maxSteps`	(optional) max. number of steps, default: 50
`dX`	(optional) convergence reached only when maximum displacement (length of displacement vector of a single atom) is less than this value (in bohr). Default: 0.01
`dF`	(optional) convergence reached only when maximum force (length of force vector of a single atom) is less than this value (in Hartree/bohr). Default: 0.001
`dEnergy`	(optional) convergence reached only when change in energy is less than this value (in Hartree). Default: 10^{-4}
`nProjectors`	(optional) number of previous steps to use for BFGS updates. Default: 10
`maxStepLength`	maximum allowed displacement (length of displacement vector for a single atom) in bohr. Larger steps are reduced by the trust radius method to a value close to the maximum (within 1%). Default: 0.3
`softModeDamping`	(optional) Initial value for Hessian shift (in Hartree/bohr^2). This is overriden with the first successful fit of a positive shift parameter. Default: 1e-2.
`driftFilter`	(flag) Project out the average force and displacement. Default: yes, if no constraints are used.

The `ricQN` group may contain a `ric` group (see Sec. 3.2.1 to define the internal coordinate generation. If left out, default parameters are used, and output from the internal coordinate setup is suppressed.

3.2.1 The `ric` group

The `ric` group defines the parameters for internal coordinate generation.

The following parameters may be set:

parameter	description
`maxDist`	maximum possible distance for considering neighbors (in bohr). Default: 10
`typifyThreshold`	minimum bond length separation of distinct bond types (the f parameter in [1]). After sorting the bond lengthes, the logarithm of subsequent lengthes are compared. If they differ by less than the threshold, the two bonds are assigned the same bond type. Default: 0.05
`rmsThreshold`	minimum distance between two bond length clusters in units of their root-mean-square displacements (the R parameter of [1]). Default: 3
`planeCutLimit`	Relative size of coordination polyhedra to separate the nearest neighbors from further atoms (the P parameter of [1]). Larger values allow for more neighbors. Default: 0.95
`withAngles`	(flag) add bond angle coordinates for all bonds
`bvkAtoms`	(optional, experimental) List of atom ids (starting from 1) for which born-von-Karman transversal force constants are added. The comma-separated list must be enclosed by square brackets []. This adds a bond-directional coordinate to each bond of the atoms in the list.

4 Output

4.1 `relaxedStr.sx`

This file list the final atomic positions in SPHInX format. If the optimization does not converge, it contains the last configuration calculated.

4.2 `relaxHist.sx`

This file lists for each calculated configuration the atomic positions and forces, in SPHInX format.

4.3 `energy-structOpt.dat`

This file lists for each step the energy. The format is one step per line:

```
<it> <energy (in Hartree)>
```

The 0-th step is the initial structure.

References

[1] C. Freysoldt, *On-the-fly parameterization of internal coordinate force constants for quasi-Newton geometry optimization in atomistic calculations*, Comp. Mat. Sci. **133**, 71 (2017). http://dx.doi.org/10.1016/j.commatsci.2017.03.001