This text documents the geometry optimizer by JH.

1 Change log

v1.0 First release version. It was not tested nor finalized for use with Gaussian or Turbomole.

- The optimizer can deal with periodic problems now. It cannot optimize the unit cell, but this feature can be easily added. It can work with space symmetries as in CIF format.
- The generation of internal coordinates was improved for weakly bonded systems: can recognize hydrogen bonds, connecting of fragments based on vdW radii. Overall the driver generates less coordinates which are more relevant. One can specify which and how many bonds can be created.
- More parameters can be specified in the opt file: the convergence criteria and initial trust radius. More can be easily added later.
- The optimization can be restarted.
- All errors are caught and reported in the log file. (This statement can never be 100% true.)
- Plenty of minor changes.
- Git is now used for code management.

v0.9 Basically a version as in the bachelor's thesis, with reworked directory structure and some minor changes in the read/write functions. Not documented.

2 Brief description and usage

What can it do? The package optimizes equilibrium geometries using programs VASP (iridium, mof, vanad), and Gaussian and Turbomole (iridium).

A typical run of the geometry optimizer is as follows. The user prepares the job as for the single point calculation. Then, he or she creates a file containing information for the driver (opt file), such as

```
program=vasp
geometry=POSCAR.init
input=INCAR
...
```

The program (`vasp`, `gaussian` or `turbomole`), initial geometry file (POSCAR format for VASP, xyz format for the others) and input file (`INCAR` in case of VASP, input file without geometry specification in case of Gaussian, file sent to `define` in case of Turbomole) are specified. The rest of the file are parameters of the algorithm which are optional and documented in section 2.1.

VASP. The geometry file (`POSCAR` format) can be in direct or Cartesian coordinates, and either with our without the line specifying elements under lattice vector lines. In latter case, the elements are read from `POTCAR`. As for the `INCAR`, `NSW` switch can be left out, as default is 0. To use both `WAVECAR` and `CHGCAR` for the initial orbital guess, `ISTART=2` and `ICHARG=1` must be given. Because of problems with electronic convergence, the optimizer checks if the maximum number of electronic steps (set by `NELM` switch, which is read by the optimizer, default is 60) does not equal the actual number of electronic steps. If it does, `WAVECAR` and `CHGCAR` are deleted and the VASP calculation performed again. This event is recorded in the log file.

Gaussian. The last line of the input file has to be the charge and spin specification. The geometry part is added by the optimizer. The keyword `Force` must be used to print energy gradient into the output. The orbitals can be restarted as usual.

The driver is started by calling the `OPT` bash script

```
OPT <opt file> <queue> <number of cpus> <memory per cpu>
```

Without any arguments, the script prints how it should be used.

The run of the optimizers consists of *energy* steps (calculation of energy and gradient) and *berny* steps (calculation of the next geometry). While running the optimizer writes log into `<opt file>.log`, beginning with information about the internal coordinates, followed by info about each berny step. The geometry trajectory is written into `geomhistory` (concatenated xyz files). In case of VASP, each generated `OSZICAR` file is printed into `OSZIhistory`. The `workspace.mat` file stores the program environment (octave variables) between subsequent berny steps. Files `e` and `g` hold the energy and gradient parsed from various program's output files. File `xyz` holds the geometry of current step, `abc` the lattice vectors, if we are periodic. Directory `qout` contains the output files of the queueing system.

If everything goes well, the optimized geometry is stored in `.optimized` file, either xyz or `POSCAR` format.

2.1 Algorithm parameters

Here, the additional optional parameters for the opt file are documented. They are all listed as if set with the default values.

- `maxsteps=100`. Maximum number of energy steps. If the maximum is reached, the optimization is stopped after performing the berny step. If restarted, they are counted from zero.

- `trust=0.3`. Initial trust radius in bohrs. This is the maximum rms of the quadratic step.

- `gradientmax=0.45e-3`, `gradientrms=0.3e-3`,
  `stepmax=1.8e-3`, `steprms=1.2e-3`
  Convergence criteria in hartrees and internal coordinate units (bohrs or radians).

- `restart=no`. The type of restart (default is no restart). If set to `energy`, the calculation is restarted at the energy step—useful when stopped during the energy calculation, when the energy calculation failed for some reason, or when the maximum number of steps was reached. If set to `berny`, it is restarted at the berny step—useful when the octave part ended with error.

2.2 Additional files

These files can be placed in the same dir as the opt file to tweak the behaviour of the algorithm.

- The `symm` file contains symmetry definition in CIF format.

- The `strains` file indicates constrained coordinates, one line for each coord. The bonds are designated as `A B 0 0`, angles as `A B C 0` and dihedrals as `A B C D`, where the letters stand for atom indices.
- The `allowedbonds` file indicates which bonds can be created. Three types of specification can be supplied.
  - `Si O`. This line specifies that Si−O bond is allowed. Several such lines can be given. Bonds not matching any such line will not be created.
  - `Si 4`. This says that Si atoms can have no more than 4 bonds attached. If this rule is to be violated, only 4 nearest atoms will be bonded.
  - `4 6`. This means that the atom number 4 can have no more than 6 bonds. This rule overrides the previous type of rule. The violation behaviour is identical.

3 How it works

The `OPT` script sources the opt file, checks for possible trivial errors (such as missing geometry file), creates a simple queue job, and submits it into the queueing system.

The queue job is an octave script and upon execution is sent to

```
/home/hermann/bin/oct
```

which is a soft link to octave on all clusters. (This has to be done as octave is installed in different locations and versions on different clusters.) This is also the only non-relative path in the whole implementation. The script first recursively adds directories `matlab` and `geometry-optimizer` to the octave path. Then it calls the octave function `opt` which is the main driver calling in turns the quantum chemical program and the function `berny` (the geometry optimizer). It is also the ugliest part of the implementation, messing with the files and system variables.

First, `oct` sets the defaults for some parameters of the berny step and then updates them from the opt file. It calls the `head` function which creates the scratch dir, parses the admin scripts (such as `G09` or `VASP`) for paths to the programs and sets the environment (`$PATH` and program specific).

Then it moves the required files into the scratch dir in case of Gaussian and Turbomole. The geometry files (`xyz`, `abc`) are created. Finally, there is a `while` loop alternating between the berny and energy steps. It begins with the `berny` function, followed by check of convergence and maximum number of steps. Follows the energy and gradient calculation. Then comes `tostd` function into play, parsing the outputs and giving `e` and `g` files. Back to the start of the loop. When the loop is broken (convergence or maximum number of steps, any error thrown up), the scratch dir is removed[1] and the job is ended.

3.1 `berny` function

First, there is a check if the `workspace.mat` file exists. If it does not, the initial procedures are performed.

1. The initial geometry is loaded.
2. If `symm` exists, it is loaded. Otherwise identity symmetry is set.
3. If `abc` file exists, it is loaded, and the $2 \times 2 \times 2$ supercell is created and used in subsequent coords generation.
4. If `allowedbonds` file exists, it is read and parsed.
5. The internal coordinates are created.
   (a) All allowed bonds shorted than 1.3 times the sum of covalent radii are created.
   (b) All allowed hydrogen bonds via oxygen shorter than 2 angstroms are created.
   (c) If there are still unconnected fragments, all allowed bonds between unconnected fragments shorter than 1, 1.1, 1.2, etc. times the sum of vdW radii are created. As indicated, this process is iterative until all fragments are interconnected.
   (d) All angles greater than 45° are created.

[1]This should be obviously changed when used with Gaussian or Turbomole.

   (e) All dihedrals with 1–2–3, 2–3–4 angles both greater than 45° are created. If one of the angles is zero, so there are three atoms on a line, they are used as a new base for a dihedral. This process is recursively repeated.

6. In periodic case, from all coords differing only by translation, just the one closest to the original cell is retained.

7. If `strains` exists, it is loaded.

8. The matrix of the weights of coordinates and Hessian are constructed.

If `workspace.mat` actually does exist, it is loaded, the previous procedures skipped. The B, G and projector matrices are calculated. The `e` and `g` files are loaded, the gradient transformed into internals. Next, if it is not the first step, the Hessian and trust radius are updated, and the linear search is performed between last two steps. Follows the quadratic step, test of convergence and iterative transformation back to cartesian coordinates. At the end, `workspace.mat` is saved.

4 Details

This section describes several implementation details which are suspected from being able to cause problems. The list is of course far from complete.

4.1 Generalized inverse

A generalized inverse of a matrix is obtained by taking its singular value decomposition and inverting only the nonzero singular values. For invertible matrices, this is equivalent to an ordinary inverse. In practice, the zero values are actually nonzero but very small. In our case, they correspond to degrees of freedom which are not independent, i.e. bounded by redundancy of the internal coordinates or by symmetry of the system. If the symmetry is not perfect (numerical inaccuracy), corresponding singular values will be small, but bigger than expected just from machine precision. On the other hand, if the symmetry of some big adsorbent is broken only by a small adsorbate, the corresponding independent modes will have small singular values. Thus, a problem of separating the independent and redundant modes arises.

Currently, the threshold is first set to 10^{-10}. The ratio of biggest singular value under and smallest above this threshold is computed. If it is bigger than 1000, it is accepted, if not, the threshold is increased tenfold, and the procedure repeated. If the threshold rises to 0.1, the function (`ginv`) throws an error. The function also returns those two critical values and they are written into the log, which can be used to watch for potential problems.