Xopt - an eXternal OPTimizer v.2.0 beta

Holger Kruse(mail2holger@gmail.com)

https://github.com/hokru/xopt

April 13, 2017

1 Introduction

The purpose of this program is to read the energy and gradient of another program (quantum-chemical, semi-empirical or even force field programs) and to optimize the structure with a hopefully very efficient algorithm. The interface is kept reasonable simple while still allowing many modifications. Since the code is open-source, adjustment is possible and encouraged.

As the program is still under heavy development the manual might be incomplete/incorrect at times.

If you find bugs or have suggestions, please let me hear about them!

Contents

2 Overview of the Features

Table 1: List of supported programs

- Turbomole
  - using `ridft`, `dscf` and `ricc2` modules.
  - _huge binaries are also supported. (not needed for Turbomole 7.x)
  - combination with the external programs `dftd3` and `gcp`.
- ORCA
- mopac12
  - combination with the external program `dftd3` .
- Amber12 (`sander` module)
- Gaussian09
- arbitrary (serial/parallel) numerical gradients through a local script
- (ask for more or add yourself ;-))

Table 2: Available optimization algorithms

- conjugate gradient
- QN-relaxation
  - rational function optimization (RFO) and size-independent RFO (SI-RFO).
- Hessian updates:
  - BFGS
  - SR1
  - SR1-BFGS
  - SS-BFSG (spectral scaling)
  - experimental SR1(SS)-BFGS
- coordinate systems
  - cartesian coordinates
  - approx. normal coordinates
  - internal coordinates (unfinished)
- Initial Hessians
  - unit Hessian
  - reading of Turbomole, ORCA and Gaussian09 Hessians
- Constraints/Restraints
  - cartesian constraints (fixing atom positions in 3D space)
  - primitive restraints (quadratic restraining potential on internal coordinates)
- transition state search
- smooth penalty function based conical section optimizations

3 Commandline Options

The program itself is called as: `xopt <structure> [-options]`
and prints to standard output. `<structure>` refers to either a xyz-file (angstrom) or a Turbomole file (tmol format, bohrs).
Please run `xopt -h` to see a full list of options. Below we explain some more tricky options in detail.

ection 4.2).

`-gcp "<level>"`
Combine the `gcp` program for gCP-correction with Turbomole. Give the level for gcp (see gcp manual). The quoation marks are needed for dft/sv(p) because of the brackets, it is good practice to always use them.

`-d3 "<func + damping>"`
Run Turbomole in combination with Grimme's `dftd3` program. See `dftd3` manual for input specifics. The version of the damping needs to be given, since there is no default! Eg. -d3 ”tpss -bj”. The ”” are necessary.

`-restart`
Read Hessian from *xopt.hess.restart* and options from *xopt.restart* to restart a calculation. Geometry is not read automatically!

`-n <integer>`
Enables the parallel numerical gradient via `xopt.pgrad` and provides the number of MPI threads. See 7.1

`-ciopt`
Enables penalty function based conical section optimizations, see 4.6

4 Preparing Inputs

xopt does not handle the input preparation for the program. Thus, the following sections describe how xopt expects the input. The individual programs are called by system calls and may involve wrapper script. It is advised to look at `getgrad.f90` and adjust the system calls to your own work environment.

4.1 ORCA

Requirement: expects a script named orca30 that calls the orca binaries.
input file: orca.in
It is necessary that the following input lines are included:

```
! ENGRAD
*xyzfile <chrg> <mult> xopt.xyz
```

where `<chrg>` is the molecular charge and `<mult>` the multiplicity.

4.2 Turbomole

Prepare a standard input using define. Supported are HF/DFT calculations using `ridft` and `dscf`, and calculations using the `ricc2` module (eg. for MP2.). A 'coord' file is automatically written and updated.

4.3 Amber

Requirement: `sander` module from Amber
Input file: *amber.in* (`sander` input) and *amber.top* (amber topology file). sander needs to write the forcedump.dat file (see AMBER manual).

4.4 Mopac

Requirement: `gcx` conversion tool (will be integrated soon). mopac binary called `mopac2012`
input file: A mopac SETUP file (named *SETUP*) containing the following keywords

```
1SCF GRAD XYZ aux(42,PRECISION=9,MOS=-99999,COMP)
```

Dont forget to add the method, charge, etc.!
I would suggest for PM6:

```
PM6 1SCF GRAD XYZ NOMM PRECISE aux(42,PRECISION=9,MOS=-99999,COMP)
```

4.5 Gaussian09

input file: g.in
Requirement: special *run-g09* script to execute Gaussian.(included in repository)

4.6 conical intersection optimization

penalty function-based (no non-diabatic coupling) CI optimizer following Levine/Martinez DOI: 10.1021/jp0761618. We can do Gaussian, Turbomole(dscf,ricc2), Orca and numerical gradients. For details see 8

5 necessary changes!

The interfaced programs are called by system calls. Chances are some of these calls need to be adapted to your own work environment. For these reasons take a look at `getgrad.f90`. The system calls should be mostly self-explanatory. Care has to be taken to use the correct names of the output files or the reading routines will not find the energy and gradient.

As an example take a look at ORCA: We call a script named `orca30` that helps with orca's parallel execution, instead of just the usual `orca` binary.

Gaussian is called through a script in the system call as well: `run-g09 g.in g.out`, which sets up a scratch directory among other things.

6 Using xopt.control

Instead of using the command line it is possible to write a control file for `xopt` named *xopt.control*. Its main purpose however lies in providing parameters for the constrained or restrained optimization. Not all command line arguments can be set in the xopt.control file (yet).

6.1 Restraints and Constraints in Optimizations

The implementation in `Xopt` is limited: Constraints are only possible for Cartesian coordinates, while restraints can be defined for internal (primitive) coordinates, e.g. bond lengths and angles. Of course, a very restrictive restraint on an internal coordinates leads to a practical constraint. Thus, using high values for the restraining potential, also internal coordinates can be 'constrained'.

The restraining potential is in the form of a simple, quadratic function with the restraining energy R being defined as

$$R(x) = k_r \cdot (p - p_{ref}) \ ,$$

with k_r as the force constant of the restraint, p the type of internal (primitive) coordinate (bond, angle or torsion) and its reference value p_{ref}.

The actual value of k_r needs to be chooses carefully and is structure and level of theory specific. The reference value x_{ref} is taken from the initial input geometry (or from the xopt.restart file if requested).

xopt checks for the file "xopt.control" at the startup. Its content will overwrite any commandline options!

```
legend:
i,j,k,l = <integer>, atomic numbers starting from 1
F = <floating point> eg. "1.0" NOT "1"
```

```
Restrained optimizations:
 $restr
 atom i F
 bond i j F
 angle i j k F
 dihed i j k l F
```

An additional gradient from the restraining potential (G_r) will be added to the total gradient: G_{total}: $G + G_r$. gnorm and all other values are calculated using G_{total} ! Thus is it better to rely on energy convergence for restrained optimizations.

One can add a target value for dihedral restraints with the keyword target:

```
  dihed i j k l F target T
```

, where T states the target value in degree.

```
Cartesian constrains:
 $freeze
 Hopt            #just optimize hydrogen positions, freeze the rest
 atom i          #freeze atom number i
 elem <string>   #freeze all elements <string> (lower case !)
 list <list>     #freeze a list of integers, eg. 'list 1-3,8,9,33-34'
```

For constrains the gradient components (and respective entries in the B-matrix) are set to zero.

Please always check the output of `Xopt`if constraints/restraints have been set as expected.

7 numerical gradients

Arbitray numerical gradients can be computed via a script/executable named `mygrad.sh` located in the working directory. It does not need to be bash script. It will be excuted by a system call inside `Xopt`. It needs to do 3 things:

1. Have the program to be executed use xopt.xyz as input
2. run the program
3. provide the energy output as a single number inside xopt.energy.tmp

An example for Turbomole would look like:

```
#!/bin/bash
#1. make use of xopt.xyz
babel -ixyz xopt.xyz -otmol coord
#2. execute
dscf > dscf.out
#3. write energy into file
grep "|  total energy" dscf.out | awk '{print $5}' > xopt.energy.tmp
```

`Xopt`will read the xopt.energy.tmp value for a 2-point (central) finite difference computation of the gradient with a step size of 0.005 bohr.

7.1 parallel numerical gradient helper: xopt.pgrad

Parallel numerical gradients (max. 3*atoms threads) are available through an add-on programm called `xopt.pgrad` (see subdirectory *pgrad*). It can be compiled by running `make pgrad` or by running `make` in the pgrad directory itself. It has only been tested with OpenMPI.

Communication with `Xopt`is handled through 2 files: xopt.para.tmp contains the input for `xopt.pgrad`. It hold the number of atoms, the xyz coordinates (Å and the atom type (as integer). For water it reads:

```
3
-1.769142     -0.076181      0.000000 8
-2.065745      0.837492      0.000000 1
-0.809034      0.001317      0.000000 1
```

`xopt.pgrad.tmp` contains the output from `xopt.pgrad`. Two gradient matrices are written after each other. There are two gradients since in the case of conical section (CIopt) optimizations one can read the energy from two states (eg. GS and S1) from the same output and form both numerical gradients simultanously. If there are not two energy values inside xopt.energy.tmp, then the 2nd gradient in `xopt.pgrad.tmp` will be zero. The gradient is in atomic units and the file is organized as follows:

```
grad1(x,atom 1), grad1(y,atom 1),grad1(z,atom 1)
.
.
grad1(x,atom n), grad1(y,atom n),grad1(z,atom n)
grad2(x,atom 1), grad2(y,atom 1),grad2(z,atom 1)
```

```
.
.
grad2(x,atom n), grad2(y,atom n),grad2(z,atom n)
```

Each thread write its output into xopt.slave.* files, enumerated with process rank. The master process hold the rank 0. Its output will be copied to xopt.pgrad.out afterwards.

Internally `Xopt`will call `xopt.pgrad` as:

```
mpiexec -by-node -n <nproc> -output-filename xopt.slave $HOME/bin/xopt.pgrad
```

meaning the `xopt.pgrad` binary has to reside in the users $HOME/bin

7.1.1 task distribution

The parallel task assigment is kept very simple: each thread gets $\frac{3*atoms}{nproc}$ tasks.

Each tasks consists of a single gradient components (e.g. 2 displacements for x-coordinate of atom 1). This likely leaves several leftover tasks (check modulus 3*atom%nproc). These will be assigned to the process with the highest rank! For best parallel performance you should aim to minimize the number of leftover tasks.

In the case of 19 atoms, we have 57 tasks to calculate. If we use 16 MPI threads, then 15 threads will calculate 3 tasks, while the 16th thread will calculate 12 tasks (3+9 leftovers), resulting into a severe bottleneck. In this case it would be better to use only 14 MPI threads, since then 13 threads will run with 4 tasks and the 14th with 5 (4+1 leftover).

tip: quickly check with the python interpreter for leftover tasks: (57%16=9 and 57%14=1).

7.1.2 scratch directories for xopt.pgrad

There are possible modes:

1.[default] xopt.pgrad will automatically make scratch directories in the working directory named `xopt-tmp-$(PID)`

2.[set by -scratch] You can set a custom scratch directory (like /scratch/myname/) for computations on clusters by adding an additional line to `xopt.pgrad.tmp`. The above water example would then read:

```
3
-1.769142     -0.076181       0.000000 8
-2.065745      0.837492       0.000000 1
-0.809034      0.001317       0.000000 1
/scratch/Beeblebrox
```

Xopt can take care of all that by itself. The first mode is the default and used when only `-numgrad -n <nproc>` is specified. The second mode is actived if additionally `-scratch /scratch/Beeblebrox` is set. Additionally, `Xopt`will do the energy computation in the same scratch path. This way, you can submit the computation directly on the NFS directory on the cluster and the actual computations will be carried out in the specified local scratch directories.

As long as `mygrad.sh` and `xopt.pgrad.tmp` are present, you can call `xopt.pgrad` directly (with mpirun...) if you want to use it in your own scripts without `Xopt`.

8 penalty function conical intersection optimization

We assume state I ¡ state J, e.g. J=I+1.

You need to make 2 directories named `stateJ.xopt` and `stateI.xopt`. Prepare the input for each state inside the directories. It should work for Turbomole, ORCA and G09 if you follow the general preparation guidelines above.

The `Xopt`output will print something like:

```
gap[eV]:   0.024   penalty:  13.1 E(low):   -546.9431436 E(high):   -546.9422645 root fli
```

where E(low) denotes the lower state (eg. groundstate) calculated as state I in stateI.xopt and E(high) as the higher state calculated as state J inside stateI.xopt. If at any stage during the optimization E(low)¿E(high) (I¿J) root flip will be set true (=T) and E(low/higher) will be interchange, e.g. E(low) will be the energy obtained in `stateJ.xopt`. This is checked for each optimization step individually and is not tracked through previous steps.

The strategy to increase the penaly σ is as follows:

$$\sigma = \sigma + (2\Delta E_{ij}/\alpha) \ , \tag{1}$$

where ΔE_{ij} is the energy gap between state I and J, α the smoothing factor (see paper). σ is increased when ΔE_{ij} is larger than 1e-3 and the penalty function change ΔF_{ij} smaller than 5e-5.

Note 1: for gaussian groundstate SA-CASSCF calculations add IOp(10/97=100) and IOp(5/97=100). It switches the CI vectors internally so one gets the proper groundstate gradient (e.g. for CASSCF(2,2,nroot=2,stateaverage)). Not sure about nroot ¿ 2.

8.1 numerical gradients

It is assumed that the gradients for the state I and J are both obtained numerically, and that it can be done through a single calculation. Thus all calculations take place inside the current working directory! You need to modify `mygrad.sh` to write the energy value for state I in line 1, and the energy value for state J in line 2 in `xopt.energy.tmp`. Otherwise see section 7 and 7.1 regarding the mygrad.sh script and parallel execution.

You can use both the serial, as well as the parallel numerical gradient.

9 External Interface: option -extern

TBD

10 user configure file .xoptrc

With the file $HOMEẋoptrc users can set their own default parameters. Commandline options given directly to `Xopt`.

11 How to Cite?

For proper technical documentation cite the git version, github repository and authors, e.g.:

1) H.Kruse, https://github.com/hokru/xopt, version v2.0b-1-g7434

`v2.0b-1-g7434` is the git build identifier. It can also be the commit number or any 'git describe' output. The `Xopt`output prints a git build at the start that can be used.

It is generated by the Makefile using:
```
git describe --abbrev=4 --dirty --always --tags
```

If you use restraints it is likely appropriate the cite the implementation paper:

2) H. Kruse, J. Sponer PCCP, 2015,17, 1399-1410

12 Closing

I sincerely hope you find the program or parts of it useful. It is published under a GNU license and allows code re-use and sharing as long as credits are given.

Much of the development has been done in my free time and I learned a lot with this project. If you find bugs or have suggestions please write to me (mail2holger@gmail.com) or open an issue on github.

I would also be happy to hear if my project is of any help. As long as I am motivated, I will continue to improve my `Xopt`.

Good luck & Don't panic!