CIOpt Program Documentation

Ben Levine and Todd J. Martinez

8/28/2007

Descriptive Name: Optimize conical intersections or minima
Binary Module: CIOpt.e
Description: This program is designed to optimize conical intersections using only energy and possibly gradient information. Nonadiabatic coupling vectors are not required. The sequential penalty method and smoothed energy functions are coded. Energy functions are also coded for optimization of minimal energy 3-state intersections and nearest intersections to a reference geometry. Implimentation of geometric constraints is forthcoming. It can also be used for simple minimization, although the advantages are then less compelling. It is designed to work with any back-end electronic structure program without modification of the source code. This is accomplished through a "template-driven" scheme for writing and reading input/output files.
Author: Todd J. Martinez and Benjamin Levine
References: Levine, Coe, Martínez, submitted to J. Phys. Chem. A, August 2007.

Files Used:

Filename	Purpose
template.write	Template for writing input deck
template.read	Template for reading energies
template.readg	Template for reading gradients
Control.dat	Input File
iter.log	Summary of iterations
long.out	Used for restarts (not known if this works)
mplog.out	Listing of all computed energies
details	Subdirectory with all electronic structure output files and xyz files of geometries (if Cartesian coordinates are used)
linmin.out	Detailed information about line minimizations. Most useful when Powell optimization scheme is used.

Input Parameters:

There are two types of input which need to be understood. First are the template files which direct the code how to create input decks and how to read the textual output of the electronic

structure code. Second is the input file which directs the code how to do the optimization, how to run the electronic structure code, etc.

Template Files

The basic idea is that CIOpt will read template files in order to learn how to read/write output and input files.

Template-Driven Writing of Input Files

Occurrences of `%%xxx` are substituted with variable `xxx`. Note that you *must* use three characters, i.e. variable 1 should be written `%%001`. Other variables which can be substituted are indicated with the prefix `%#VARNAM`. Again, the variable name *must* be six characters long. Only variables which have been coded explicitly can be substituted. Currently, the following are implemented:

`%#STATE` = current state being calculated
`%#ISTATE` = istate
`%#JSTATE` = jstate

Example:

```
This is variable 1: %%001 and we are calculating the energy of
state %#STATE .
```

is translated as

```
This is variable 1: 0.01 and we are calculating the energy of
state 1.
```

Template-Driven Reading of Output Files

Note that all directives are processed in order. Essentially, we provide directives to find search strings in the output, skip a given number of lines, and read variables from fixed positions in the lines. The directives are:

^iiistring	Find iiith occurrence of string after start of line
*iiistring	Find iiith occurrence of string anywhere in a line Note that we only count one occurrence per line
@iii	Skip iii lines
&%iiffffjjjkk%llggggmmmnn	Using format string ffff of length ii, read par(j) starting at position kk, then do same for par(m) with format string gggg of length ll starting at position nn
!string%iiffffjjjkk%llggggmmmnn	Like above, but will search for line beginning with string ^iiistring

Program Input

This is mostly namelist-driven and should be in a file named `Control.dat`. A sample input file with optimization in Cartesian coordinates is:

```
&control
 natoms=6
 nstates=2
 istate=2
 nefunc=7
 zlagrange=.true.
 ztolramp=.true.
 zforward=.true.
 nopt=3
 crunstr='/usr/localmqm/molpro2002.8/bin/molpro tmp.com'
/
  C     -0.026579475   -0.016976589   -0.080109655
  C     -0.152980476    0.050503427    1.258095561
  H      0.716185182    0.654602534    1.166951122
  H      0.034632433    0.800637262   -0.229743441
  H      0.150545672   -0.918277298   -0.619230517
  H     -0.756803336    0.802446976    1.706561410
```

A similar sample input file for optimization in internal coordinates is:

```
&control
 natoms=6
 nstates=2
 istate=2
 nefunc=7
 zlagrange=.true.
 ztolramp=.true.
 znoncart=.true.
 zforward=.true.
 ndims=12
 nopt=3
 crunstr='/usr/localmqm/molpro2002.8/bin/molpro tmp.com'
/
2.54 0.1  0.01  1.0
2.01 0.1  0.01  1.0
82   10.0 1.0   5000.0
1.57 0.1  0.01  1.0
98   10.0 1.0   5000.0
309  10.0 1.0   10000.0
2.01 0.1  0.01  1.0
124  10.0 1.0   5000.0
117  10.0 1.0   10000.0
2.01 0.1  0.01  1.0
121  10.0 1.0   5000.0
92   10.0 1.0   10000.0
```

Variables which may appear in the namelist:

Variable	Default	Description
nstates	none	Number of electronic states
istate	none	State on which to minimize, or upper state if intersection

		search
jstate	istate-1	Lower state for intersection search
kstate	istate-2	(for nefunc = 10) Lowest state for 3 state MECI search
natoms	none	Number of atoms (only used when optimizing in Cartesian coordinates)
ndims	none	Number of geometric variables to optimize (only used when optimizing in internal coordinates)
znoncart	.false.	If true, internal coordinates are being used for optimization. Otherwise, use Cartesian coordinates.
zmat	.true.	If znoncart is true, and zmat is true the program assumes that the input coordinates are zmatrix coordinates and makes some intelligent adjustments. Specifically, it changes the step sizes for the angles and, in the case the opt=1, reorders the direction so that the angles come before the bond lengths.
nopt	3	Optimization algorithm to use (See Numerical Recipes, Ch 10 for descriptions) 1 – Powell 2 – Conjugate Gradient (no longer available) 3 – BFGS
nefunc	7	Determines the function which will be optimized 1 – $E = E_I$ (minimize on state istate) 7 – $E = \frac{E_I + E_J}{2} + \lambda \frac{\Delta E^2}{(\Delta E + \alpha)}$ (default MECI optimization) 8 – $E = \frac{E_I + E_J}{2} + \frac{1}{2}\lambda \Delta E^2$ (another reasonable MECI optimization scheme) 10 - $E = \frac{E_I + E_J + E_K}{3} + \frac{1}{3}\lambda \left(\frac{\Delta E_{IJ}{}^2}{(\Delta E_{IJ} + \alpha)} + \frac{\Delta E_{IK}{}^2}{(\Delta E_{IK} + \alpha)} + \frac{\Delta E_{JK}{}^2}{(\Delta E_{JK} + \alpha)} \right)$ (3-state MECI optimization) 11 - $E = \frac{RMSD^2}{RMSD + \alpha_{RMSD}} + \lambda \frac{\Delta E^2}{(\Delta E + \alpha)}$ (Nearest Intersection search)
dlambdagap	3.5	Initial value for penalty weight λ
dlambdagap max	100.0	Maximum value for the penalty weight λ
alpha	0.02	Smoothing parameter used for nefunc=7

tol	1.0d-06	Convergence threshold on energy change.
gtol	5.0d-03	Convergence threshold on gradient
cigap	0.001	Largest energy gap which is acceptable for a conical intersection
step	0.1	Initial step size (used in Powell optimization)
stepnd	0.01	Step size for finite differences.
spen	1.0	Initial value of σ in multiplier penalty method
zrestart	.false.	If true, will attempt to restart using long.out
zangrad	.false.	If true, analytic gradients are available
zforward	.true.	If false, central differences are used. Otherwise, use forward differences
crunstr		Command to run electronic structure program
ctmpread	template.read	Name of template file for reading energies
ctmpgread	template.readg	Name of template file for reading gradients for grad of istate
ctmpg2read	template.readg2	Name of template file for reading gradients for grad of jstate
ctmpg3read	template.readg3	Name of template file for reading gradients for grad of kstate
ctmpwrite	template.write	Name of template file for writing energy input decks
ctmpwriteg	template.writeg	Name of template file for writing gradient input decks
cinpdeck	tmp.com	Input file for electronic structure program
coutfile	tmp.out	Output file of electronic structure program
zdetails	.false.	Tree killer output. detail directory is only created if .true.
zexenev	.false.	read in excitation energies in eV, rather than absolute energies in Hartree (useful with TDDFT codes)
rmsdweights	(atomic masses)	(for nefunc=11) weights for the calculation of the RMSD distance
rmsdgeo	none	(for nefunc=11) the reference geometry for nearest intersection search
alpharmsd	.1d0	(for nefunc=11) smoothing constant for RMSD
zibf	.false.	apply Incremental Barrier Function method to constrain maximum energy of istate (works with any nefunc; see Practical Methods of Optimization by Fletcher, Ch 12 for details)
ribf	.04d0	the weight for the barrier function in IBF calculations
Emaxibf	1.0d0	the maximum energy in IBF calculations

Format of geometric data after namelist (Cartesian coordinates):

`natoms` lines of

```
char1 floatx floaty floatz
```

where `char1` will be the element name in the output XYZ files, and `floatx`, `floaty`, and `floatz` are the initial values of the Cartesian coordinates for this atom.

Format of geometric data after namelist (Internal coordinates):

`ndims` lines of

`float_initial float_step float_stepnd float_hessin`

where `float_initial` is the initial value of this geometric parameter, `float_step` is the step size for this geometric parameter (analog of step used only if `nopt=1`), `float_stepnd` is the step size for finite differences of this geometric parameter (only used if `zangrad=.false.`), and `float_hessin` is the initial value of the diagonal of the inverse Hessian matrix for this geometric parameter. Good values for usual internal coordinates are:

	Bond Length	Angle	Dihedral
`float_step`	0.1	10.0	10.0
`float_stepnd`	0.01	1.0	1.0
`float_hessin`	1.0	5000.0	10000.0