Optking: a molecular geometry optimizer

What's new and how to use it

ROLLIN KING, BETHEL UNIVERSITY (MN)

ALEX HEIDE, UNIVERSITY OF GEORGIA

PsiCon2020

OVERVIEW

1. INTERFACING
2. NEW FEATURES

- C++ (old) optking
  - Still default Psi4 optimizer at present
  - `psi4.optimize('scf')`
- newer (python) optking
  - Duplicates all capabilities of previous
  - Actively maintained; active test suite
  - ~ all psi4 tests may run with trivial changes to input (A few complex calculations are also impacted by ongoing psi4 driver changes).
  - github psi-rking/optking (conda –c psi4/label/dev version is old)
- Default approach unchanged
  - redundant internal coordinates, empirical hessian guess, RS-RFO steps, BFGS update. IRC's in internal coordinates and P-RFO for TS's.

How to use optking

1. Psi API
2. Psithon
3. explicit looping (limited features)

New feature: ranged coordinates

- out-of-plane angles working; should help with BX3
- any coordinate can now be: '**free**', '**frozen**', or '**ranged**'
  - For example to limit the O-O distance in HOOH to 1.30-1.35Å.

```
xtra = { 'ranged_distance' : "2 3 1.30 1.35" }
```

  - To restrict the O-O-H bends to 105-110 degrees.

```
xtra = { 'ranged_bend' : "(1  2  3  105.0  110.0) (2 3 4 105.0 110.0")}
```

For now, all new keywords that are non-native to psi4 are passed through as an argument:

```
optking.optimize_psi4('hf', **xtra)
```

New Feature: Arbitrary extra forces

A force may be added to any coordinate. For example, a Hooke's law force at 0.95Å on the OH bond lengths of HOOH:

```
xtra = { 'ext_force_distance': "1 2 '-8.0*(x-0.950)'  3 4 '-8.0*(x-0.950)' "}
```

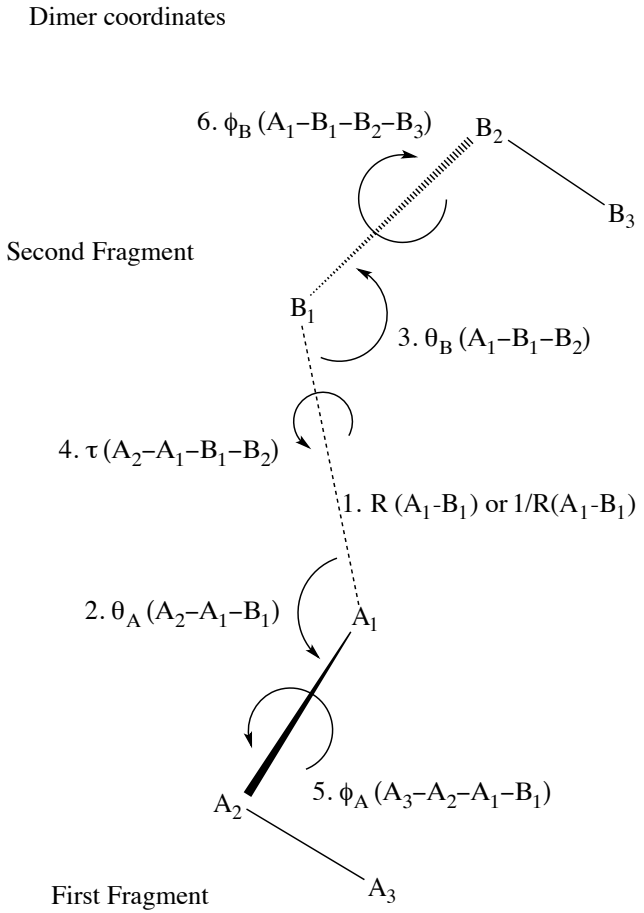
or, if you want to be more cautious, a dampened one at long-range:

```
xtra = { 'ext_force_distance': "1 2 '-8.0*(x-0.950) * exp(-20*abs(x-0.950) )'
  3 4 '-8.0*(x-0.950) * exp(-20*abs(x-0.950))'"
```

- Support for sin, cos, log, ln, log10, exp, pow, abs.
- Better response to 2017 forum request ☺

http://forum.psicode.org/t/optimization-with-additional-harmonic-constraints/404

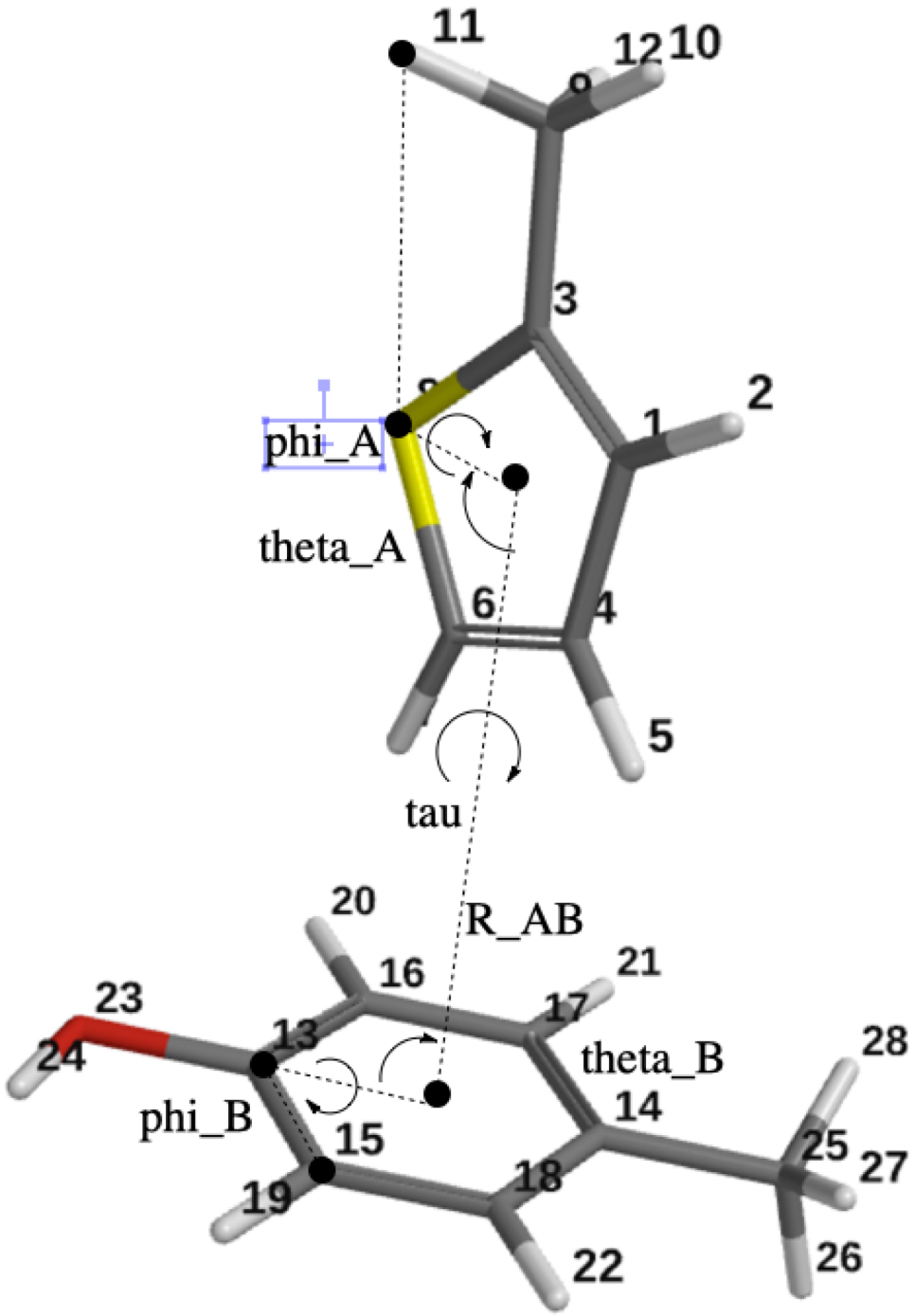
New feature: Dimer coordinates may be used to position, constrain, and optimize.



- At most 3 reference atoms on each fragment define the relative orientation.
- Each reference point is a linear combination of atomic positions within the fragment.
- Original conception with W.D. Allen (UGA) at a long-ago PsiCon in Athens.

Examples of dimer coordinate use

- Example #1. Explicit PES scan.
  - Spin one benzene around above another at a distance of 6 angstroms
  - test_dimers_benzene_pes.py , and corresponding jupyter notebook.
- Example #2. methylthiophene-tyrosine. Scan distance between monomers at fixed orientation.
  - test_dimers_mt_tyr_Rscan.py
- Example #3. methylthiophene-tyrosine. Optimize monomers and distance between them at fixed orientation.
  - test_dimers_mt_tyr_frozen_orientation.py




```
MTdimer = {
  'Natoms per frag': [12, 16],
  'A Frag'         : 1,
  'A Ref Atoms'    : [[1,3,4,6,8], [8], [11]],
  'A Label'        : 'methylthiophene',
  'B Frag'         : 2,
  'B Ref Atoms'    : [[13,14,15,16,17,18], [13], [15]],
  'B Label'        : 'tyrosine',
  'Frozen'         : ['theta_A','theta_B','tau','phi_A','phi_B']
}
```

For single-point scan of R_AB:

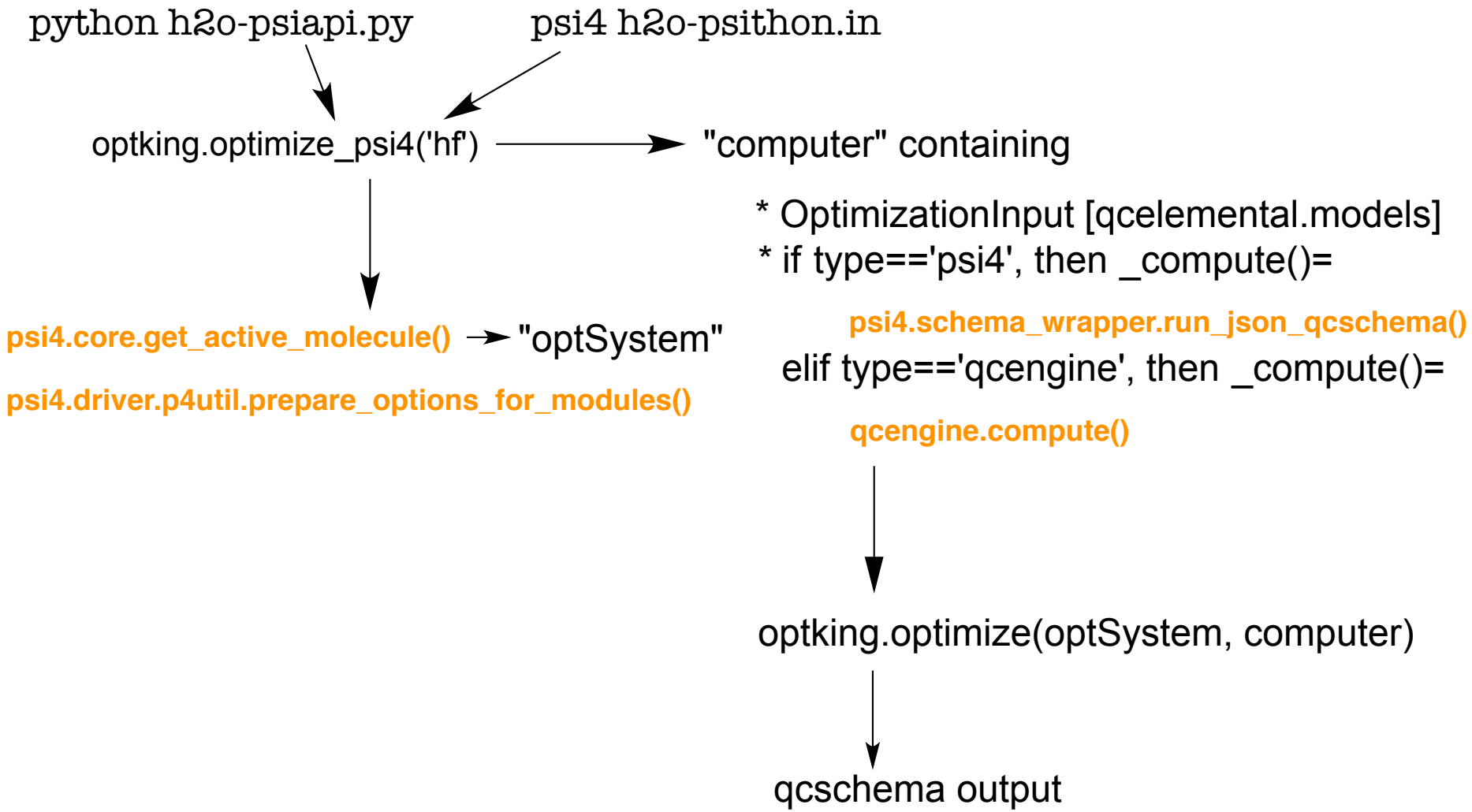
```
dimerCoord = optking.dimerfrag.DimerFrag.fromUserDict(dimer)
for R_AB in np.arange(4.0, 8.1, 1.0):
    q_target = np.array([R_AB, theta_A, theta_B, tau, phi_A, phi_B])
    Bxyz[:] = dimerCoord.orient_fragment(Axyz, Bxyz, q_target)
    xyz = psi4.core.Matrix.from_array( np.concatenate( (Axyz,Bxyz) ) )
    MT_mol.set_geometry( xyz )
    E_Rab.append( [R_AB, xyz.to_array(), psi4.energy('b3lyp-d3mbj')] )
```

For optimization or R_AB and fragments:

```
not interfragment angular coordinates.
'b3lyp-d3mbj', XtraOptParams={'interfrag_coords': str(MTdimer)}
```

Method #1, psi API

Method #2, psithon input



Method #3, explicit looping with PSIAPI (limited functionality)

python h2o-opthelper.py

psi4.set_options(psi4_options)

psi4.core.get_active_molecule()

optSystem

opt = optking.OptHelper('hf', comp_type='user')

"computer" containing

* OptimizationInput

* if type=='user', then _compute()= returns provided energy/gradient.

```
while not conv:
    # assign energy and gradient
    ....
    opt.step()
    conv = opt.test_convergence()


json_out = opt.close()
```