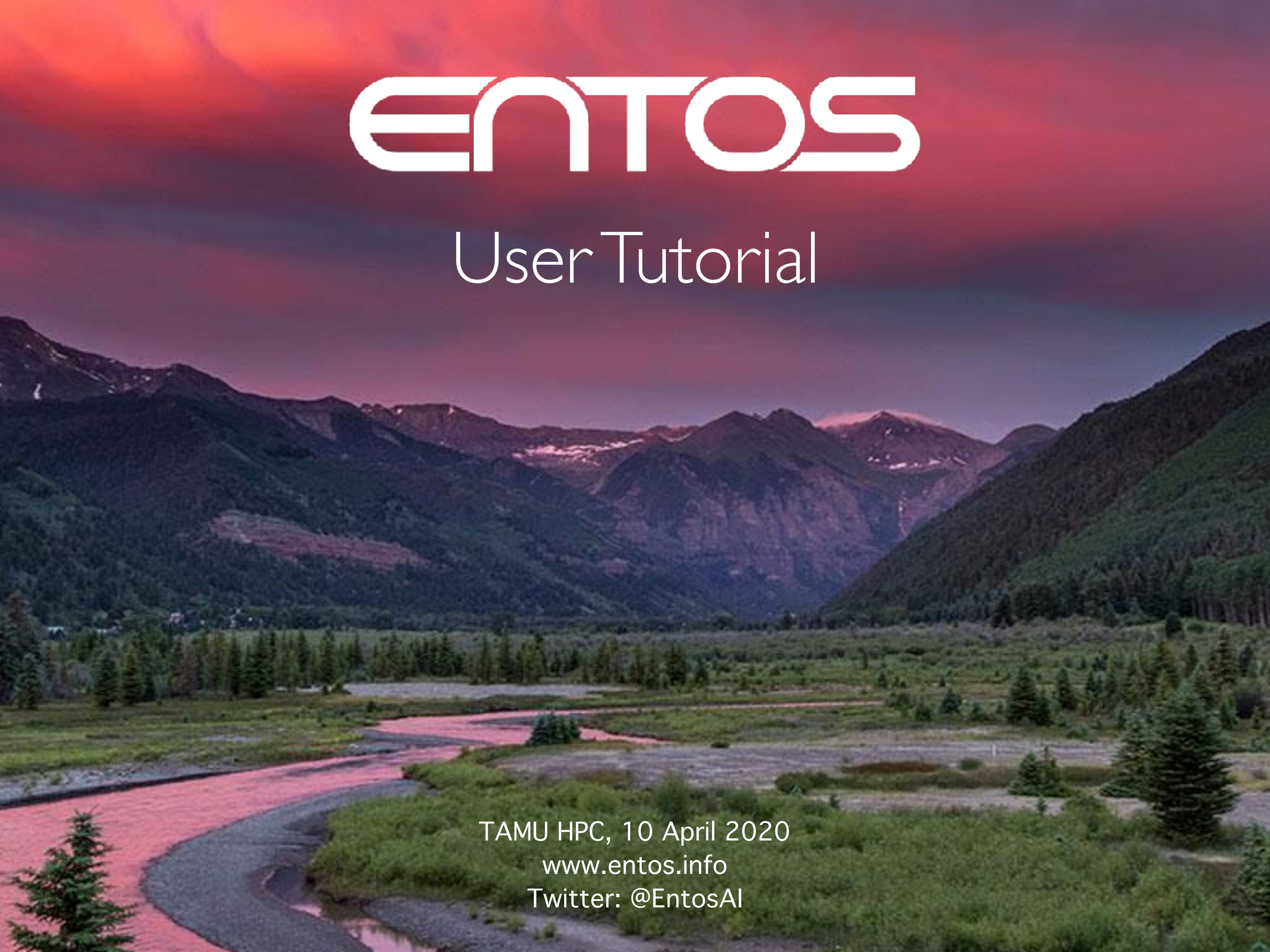


ENTOS

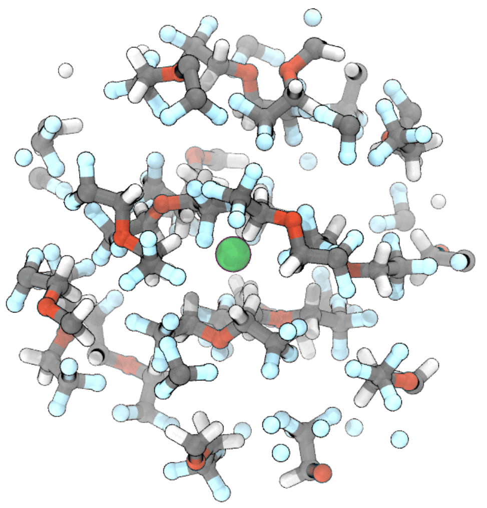
User Tutorial

TAMU HPC, 10 April 2020
www.entos.info
Twitter: @EntosAI

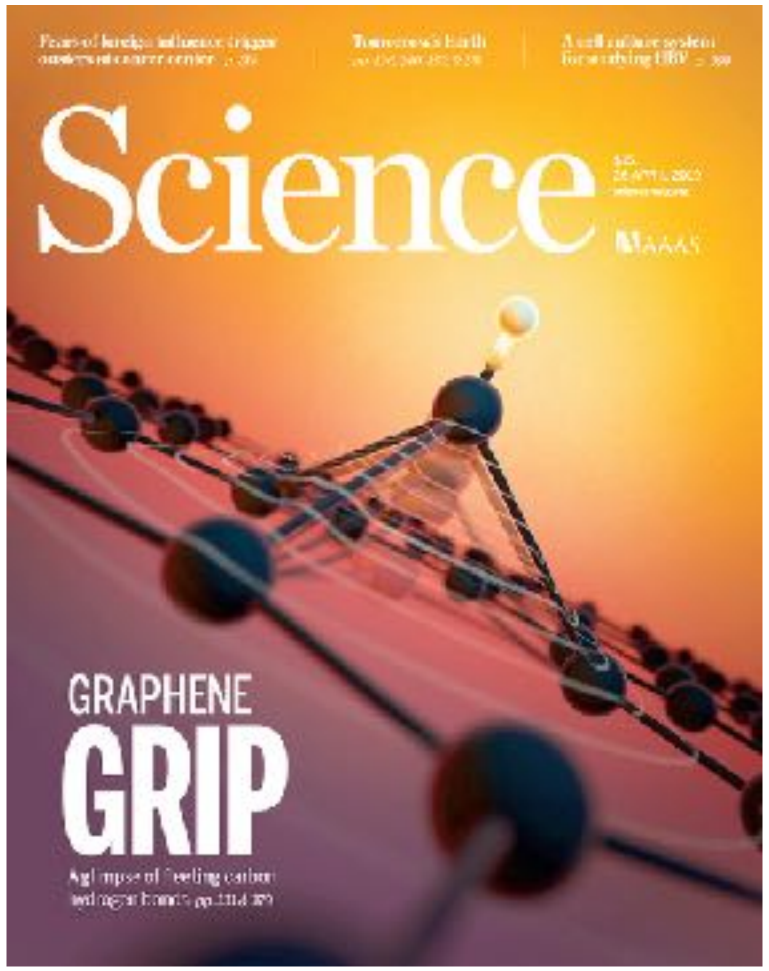
Entos Accelerates Discovery

- Entos is a fast and robust molecular simulation tool
- State-of-the-art implementations of familiar methods
- Next-generation methods
    - Physics-based machine learning
    - Quantum embedding
- Built to integrate in complex research workflows

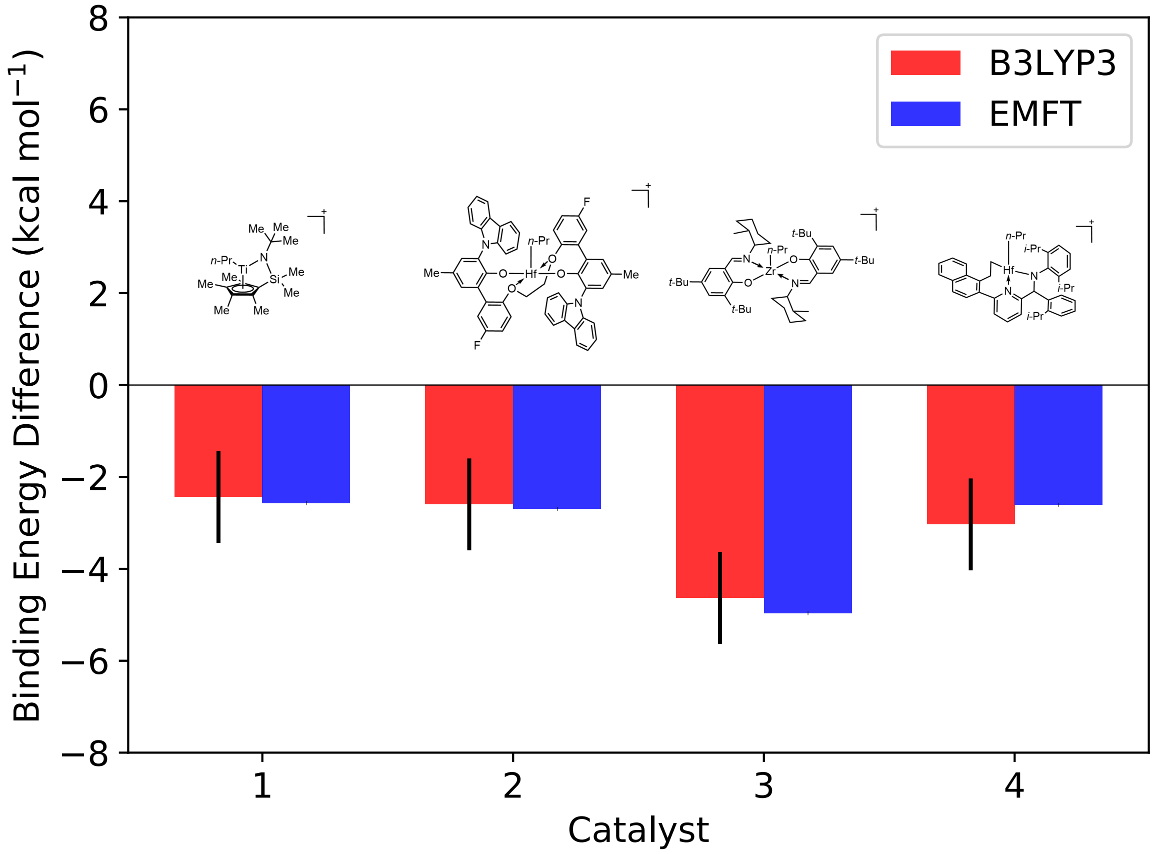
What you can do with Entos



Using MD and ab initio redox calculations to design battery electrolytes with expanded windows of electrochemical stability
Miller, in collab. with JPL and Honda., *Science*, 2018.



Using quantum embedding to discover reactivity and energy-transfer in H-on-Graphene collisions
Miller, Manby, Wodtke, et al., *Science*, 2019. *(Cover Feature)*



Using quantum embedding to accelerate polyolefin catalyst discovery
Miller, Manby, in collab. with Dow Chemical, *submitted*.

Capabilities

- Energy methods
  - Hartree-Fock, DFT, MP2
  - Semi-empirical GFN-xTB
- Unique methods
  - MO-based ML (MOB-ML)
  - Embedded mean field theory
- Solvation
  - GBSA for GFN-xTB
  - COSMO for DFT and HF
- PES exploration
  - Optimization
  - Constraints
  - MD, including QM/MM
  - Simulated annealing
- Powerful workflow integration
  - Python/Jupyter integration
- Properties, excited states, spectroscopies

Outline of the tutorial

- Part 1: Entos basics
    - Running Entos, I/O, integration
    - Basic calculations: single energy, geometry optimization, TS search
    - Tutorial Practice
- Part 2: Additional features
    - Implicit solvation, excited states, and properties
    - Tutorial Practice
- Part 3: Unique capabilities
    - Molecular orbital machine learning (MOB-ML)
    - Embedded mean-field theory (EMFT)
    - More advanced optimization and conformational search
    - Tutorial Practice

Getting started

Platforms for Entos

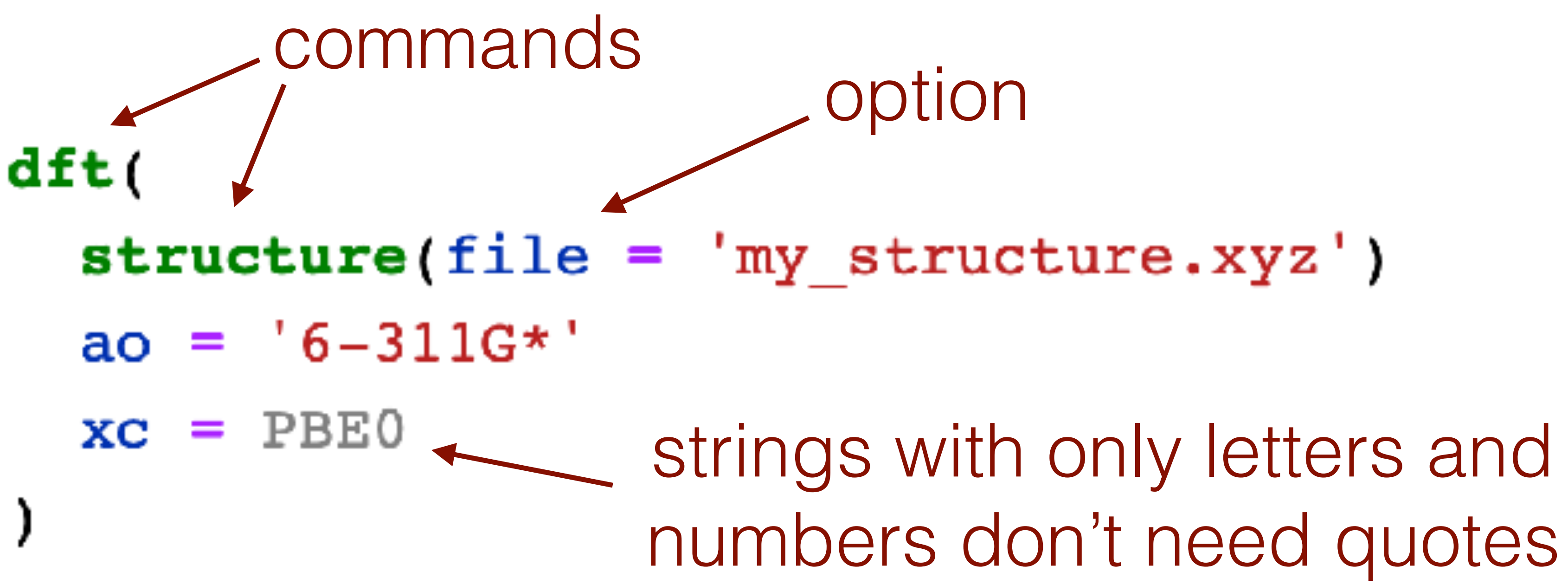
- Entos can run on:
  - Personal device or computer
  - High-performance computing facility
  - Cloud (currently AWS and DigitalOcean)
- Three modes of interaction:
  - Simple text I/O

```
$ entos my_calc.in > my_calc.out
$ entos -o my_calc.out my_calc.in
$ entos -s "xtb(structure(molecule = water))"
```

  - Integration with workflows through JSON output
  - Interactive Jupyter notebook

Text-based I/O

- Input text is hierarchical structure and case sensitive
- Basically two things – commands and options:



```
dft(
  structure(file = 'my_structure.xyz')
  ao = '6-311G*'
  xc = PBE0
)
```

- Text-based output produced…

Hierarchical input

```
dft(
  structure(molecule = water)
  ao = '6-31G' xc = B3LYP
)
gradient(
  structure(molecule = water)
  dft(
    ao = '6-31G' xc = B3LYP
  )
)
aimd(
  structure(molecule = water)
  gradient(
    dft(
      ao = '6-31G' xc = B3LYP
    )
  )
)
```

- The gradient command takes dft (or another energy command) as a subcommand
- The aim command expects a gradient subcommand to determine where the forces come from
- This hierarchical structure is consistent in Entos input, helping users to gain rapid command of more complex functionality

Text-based output

```
==================================================================================
          __________ _______ ____________ _______     _________
         / _________// _____ \\____  ____// _____ \   / ________)
        / /________   / /     \ \    | |    / /     \ \ | (_______
       |  _________|| |       | |    | |   | |       | | \________ \
        \ \_________ | |       | |    | |    \ \_____/ /__________) |
         \__________||_|       |_|    |_|     \_______//___________/

==================================================================================

 Git hash:   5b98fd20cf6aea5f297de4383e621c6e3b495217
 Git date:   Thu, 5 Mar 2020 21:34:56 +0000
 Compiler:   AppleClang 11.0.0.11000033
 Build type: RelWithDebInfo
 Version:    0.7.1

 Start time: Tue Mar 10 10:37:48 2020

 Data directory: /Users/Fred_Manby/Projects/entos/entos/data
 Number of cores: 8

--------------------------------------------------------------------------------
                                   command: dft
--------------------------------------------------------------------------------
 AO basis set:                           6-311G*
 Number of AO basis functions:         222
 Formula:                                C9H14O
 Nuclear charge:                        76.000000000
 Total charge:                           0.000000000
 Number of electrons:                   76.000000000

 Number of alpha electrons:             38
 Number of beta electrons:              38
 Number of unpaired electrons:           0
 Spin:                                   0

 Theory:                                 DFT
 Exchange-correlation functional:        PBE0
 Method for Coulomb:                     incore_df
 Method for exact exchange:              pre_transformed_df
 Number of grid points:             90688
 DF basis set:                           def2-universal-JKFIT
 Number of DF basis functions:        1004
 Schwarz screening threshold:            1.00e-10
 Coulomb fitting method:                 Cholesky
 Method for AO orthogonalization:        symmetric
 Threshold for linear dependence:        1.00e-07
```

```
------------------------
 Running restricted SCF
------------------------
Max number of iterations:             128
Conv. threshold (gradient):             1.00e-05
Conv. threshold (energy):               1.00e-06
Interpolation scheme:                   adiis+cdiis
Initial guess:                          SAD

----------------------------------------------------------------
                 energy        change          grad        time
----------------------------------------------------------------
  0      -428.407083819                    1.44e+01        0.37
  1      -425.882813614      2.52e+00      1.19e+00        0.33
  2      -425.789205472      9.36e-02      2.54e+00        0.32
  3      -426.099848643     -3.11e-01      7.19e-01        0.33
  4      -426.018211336      8.16e-02      1.81e+00        0.33
  5      -426.117800431     -9.96e-02      3.68e-01        0.33
  6      -426.121898967     -4.10e-03      2.25e-01        0.33
  7      -426.123117903     -1.22e-03      1.35e-01        0.33
  8      -426.123540320     -4.22e-04      6.21e-02        0.34
  9      -426.123654923     -1.15e-04      2.36e-02        0.33
 10      -426.123691190     -3.63e-05      1.62e-04        0.32
 11      -426.123691192     -1.49e-09      4.18e-05        0.33
 12      -426.123691192     -1.23e-10      6.73e-06        0.33

SCF converged in 12 iterations.

Molecular Dipole:                       1.115318      -0.273425      -0.032602
TOTAL ENERGY:                        -426.123691192

--------------------------------------------------------------------------------

End time:           Tue Mar 10 10:37:56 2020
Total time elapsed: 8.4 s

==================================================================================
```

Workflow integration via formatted output

- Entos can export of results as JSON

```
$ entos --mute --json-results \
        -s "water_xtb := xtb(structure(molecule = water))"
```

- Produces JSON output, with named results:

```
"water_xtb" : {
    "n_channels" : 1,
    "eigenvalues" : [-7.5787701702507571e-01, ...],
    "energy" : -5.7684979016057421e+00,
    "converged" : true,
    "n_iter" : 7,
    ...
}
```

- Can also be used for check-pointing

Using Entos in Jupyter

- Basic functionality is in a package + notebook extension:

```
import entos
%load_ext entos.extension
```

- The Entos cell magic is a great way to get started

```
%%entos
foo := xtb( structure(molecule = water) )
```

- Results can be extracted from the cell magic:

```
%%entos -f json
foo := xtb( structure(molecule = water) )

print(foo.get(“energy”))
```

- Additional Python/Jupyter in the supplementary slides at end.

Basic calculations

Specifying molecular structure

- Structure specified using the `structure` subcommand
- By name (just for testing…)

```
structure(molecule = water)
```

- Inlined xyz data (distances in Ångström)

```
structure(xyz = [[H, 0, 0, 0], [Cl, 0, 0, 1]])
```

- By xyz file

```
structure(file = 'my_structure.xyz')
```

- By formula (atom and diatom)

```
structure(formula = HCl bond_length = 1.0 angstrom)
```

Single-point energies

- Density functional theory

```
dft(
  structure(molecule = toluene)
  ao = 'cc-pVDZ'
  xc = B3LYP
)
```

- Similar input for `mp2`, `xtb`, `hf`:

```
hf(
  structure(file = 'my_structure.xyz')
  ao = 'STO-3G'
)
```

Further options

- Specifying the electronic state

```
hf(
  structure(formula = CN bond_length = 1 angstrom)
  ao = '6-31G*'
  multiplicity = triplet
  charge = -1
)
```

- Convergence parameters

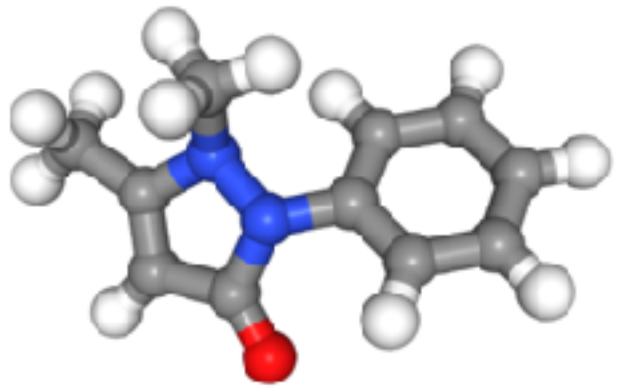
```
hf(
  ...
  orbital_grad_threshold = 1e-7
  energy_threshold = 1e-9
)
```

- Full documentation bundled as xml, or at www.entos.info/manual

Geometry optimization

- Simple, unconstrained geometry optimization

```
optimize(
  structure(file = 'phenazone.xyz')
  xtb()
)
```



- Transition-state optimization

```
optimize(
  structure(file = 'ts_guess.xyz')
  ts = true
  xyz_output = 'ts_opt.xyz'
  xtb() ! or another energy-method command such as dft
)
```

Vibrational frequencies, IR intensities, thermo

- Frequencies can be computed through the **hessian** command

```
hessian(
  structure(molecule = methanol)
  save_normal_modes = 'methanol_modes.molden'
  xtb() ! or dft, hf, etc.
)
```

```
Vibrational frequencies:
mode      frequency (cm-1)
   0                334.33
   1               1026.64
   2               1061.70
   3               1177.22
   4               1307.43
   5               1436.92
   6               1481.22
   7               1484.29
   8               2934.29
   9               2950.46
  10               3021.70
  11               3643.62
```

- Harmonic thermochemistry corrections

```
thermodynamics(
  structure(molecule = methanol)
  hessian(xtb())
  temperature = 300 kelvin
  pressure = 1 atm
)
```

units are specified for parameters that are not obviously given in a.u.

Molecular Dynamics

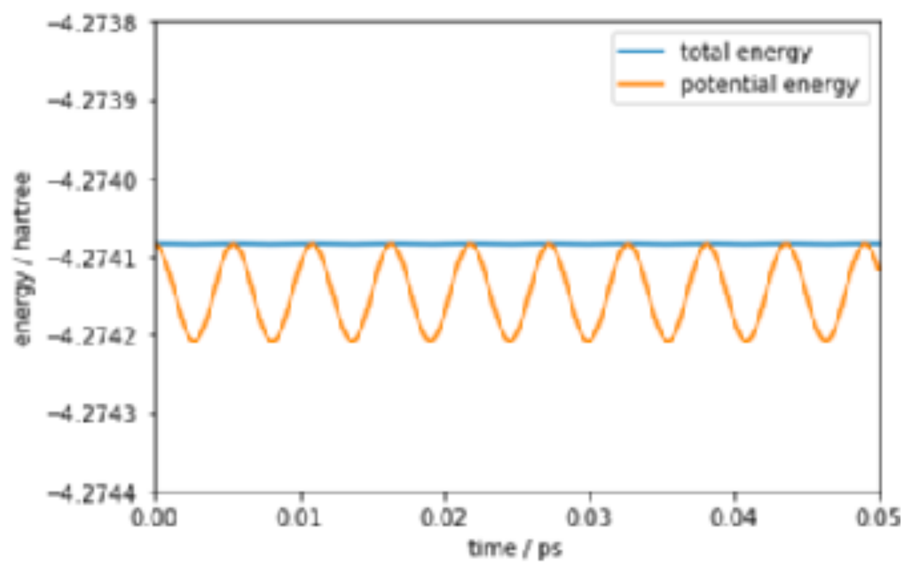
- Run a classical MD trajectory using an *ab initio* PES using the **`aimd`** command

```
aimd(
   structure(molecule = methane)
   gradient(xtb())
   n_steps = 200
   time_step = 0.25 fs
   save_to_file = true
)
```

Print out energies, coordinates, forces, velocities, etc., to file with specifiable names/locations.

- Analyze and plots results:

```
#Step         Time (ps)   Kinetic En. Potential En.  Total Energy
  0          0.0000000    0.0000000    -4.2740838    -4.2740838
  1          0.0002500    0.0000025    -4.2740863    -4.2740838
  2          0.0005000    0.0000097    -4.2740936    -4.2740839
  3          0.0007500    0.0000212    -4.2741051    -4.2740839
  4          0.0010000    0.0000359    -4.2741199    -4.2740840
  5          0.0012500    0.0000528    -4.2741368    -4.2740841
  6          0.0015000    0.0000704    -4.2741545    -4.2740842
  7          0.0017500    0.0000873    -4.2741716    -4.2740843
  8          0.0020000    0.0001023    -4.2741866    -4.2740843
```



Practical Exercises, Part #1

Using the Jupyter Notebook

- go to: jupyter.entos.info/
- log in with GitHub account details (github.com/join, if you don't have it)
- Open the Jupyter notebook tutorial (tutorial.ipynb)
- Start clicking through the examples (using shift-return)
- Explore by modifying inputs, and use save your notebook under a different filename (file→save-as, etc.) if you want it for future logins.
- Stick within Part 1 of the tutorial for now, and we'll do the rest in a moment.

Using text-based I/O via the Linux Terminal

- From your JupyterHub Home Page, select New→terminal
- Run directly with text-based input (i.e., entos -s "xtb(structure(molecule=water))")
- Create text-based input files (using nano) and run (i.e., entos my_calc.in > my_calc.out)

Solvation, excited states, and properties

Continuum solvation

- Fast GBSA implementation for GFN-xTB

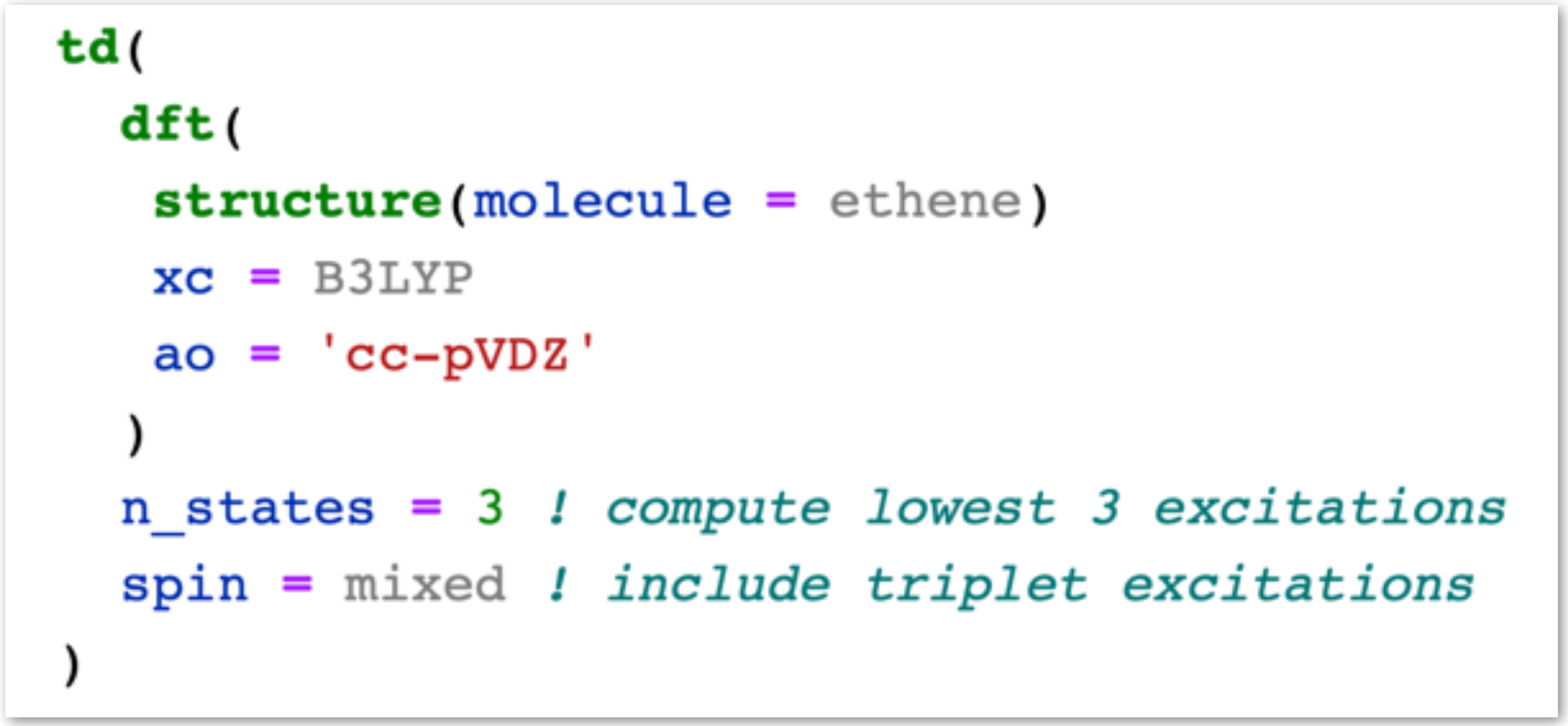
```
xtb(
  structure(file = 'my_molecule.xyz')
  solvation(solvent = benzene)
)
```

- Finer solvent control available through (documented) options
- COSMO solvation for DFT (implementation being finalized)

```
dft(
  structure(file = 'my_molecule.xyz')
  xc = PBE ao = 'cc-pVDZ'
  solvation(epsilon = 27.2)
)
```

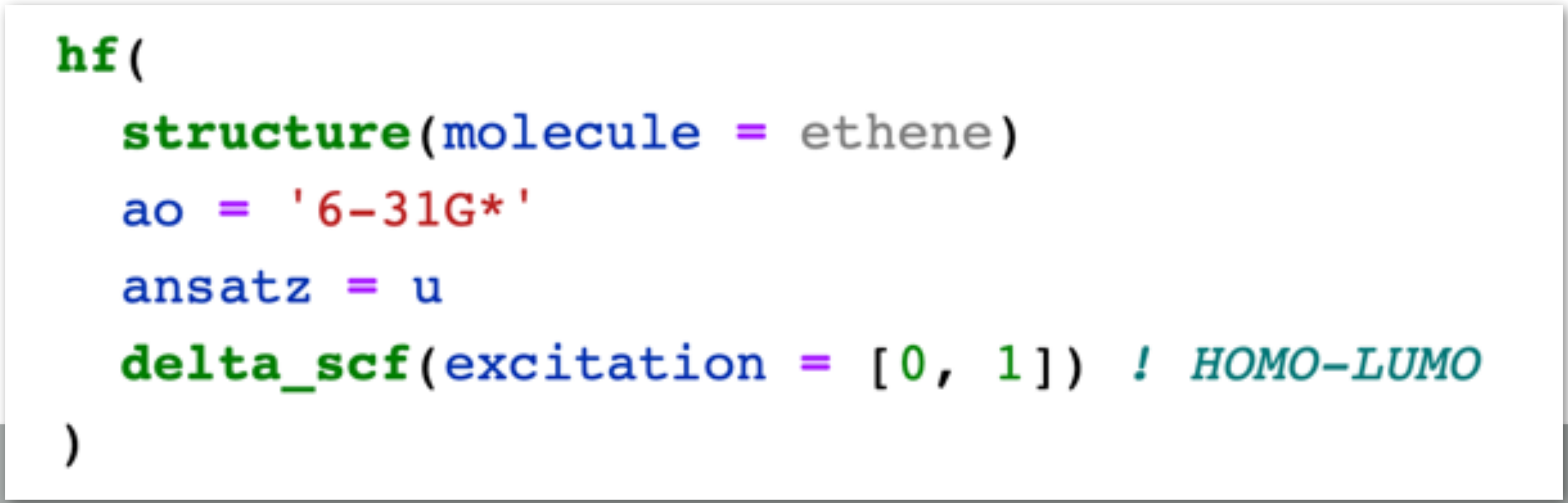
Excited states

- Linear response TDDFT



```
td(
  dft(
   structure(molecule = ethene)
   xc = B3LYP
   ao = 'cc-pVDZ'
  )
  n_states = 3 ! compute lowest 3 excitations
  spin = mixed ! include triplet excitations
)
```

- Delta-SCF:



```
hf(
  structure(molecule = ethene)
  ao = '6-31G*'
  ansatz = u
  delta_scf(excitation = [0, 1]) ! HOMO-LUMO
)
```

IR Intensities

- Already seen that IR frequencies can be computed, but we can also obtain compute a number of other properties, including IR intensities:

```
hessian(
    structure( molecule = water )
    dft(
        xc = B3LYP
        ao = 'cc-pVDZ'
    )
    intensities = true ! Calculate IR Intensities
)
```

```
Vibrational frequencies:                IR intensities:
mode      frequency (cm-1)              mode   intensity (km/mol)
   0               1639.88                 0              57.6714
   1               3774.26                 1               2.9870
   2               3885.13                 2              22.5008
```

Charge populations

- Population analysis:

```
hf(
  structure(molecule = water)
  ao = '6-31G*'
)
population() ! get populations for previous SCF
```

```
 -----------------------------------------------
  atom     element   population      charge
 -----------------------------------------------
    1         O          8.8950      -0.8950
    2         H          0.5525       0.4475
    3         H          0.5525       0.4475
 -----------------------------------------------
```

- Type of population is controlled by

```
method = {mulliken | lowdin | iao}
```

NMR properties

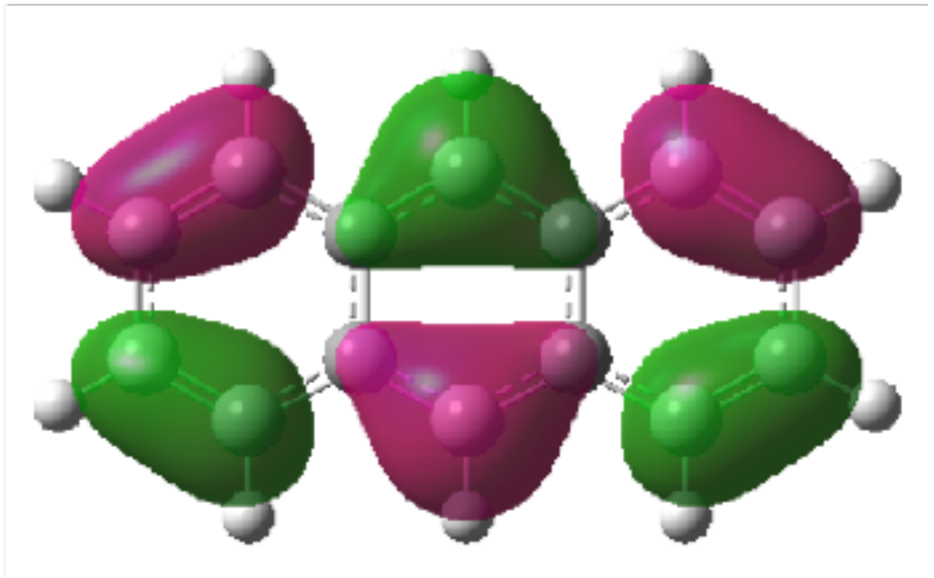
- NMR shieldings for LDA, GGA and hybrid functionals
- Works for open and closed shell

```
nmr(
  structure(molecule = methanol)
  dft(
    xc = B3LYP
    ao = 'cc-pVDZ'
  )
)
```

```
Summary of isotropic shielding constants (in ppm)
1 O     323.0073
2 H      32.5565
3 C     141.2955
4 H      27.9030
5 H      28.0305
6 H      27.9030
```

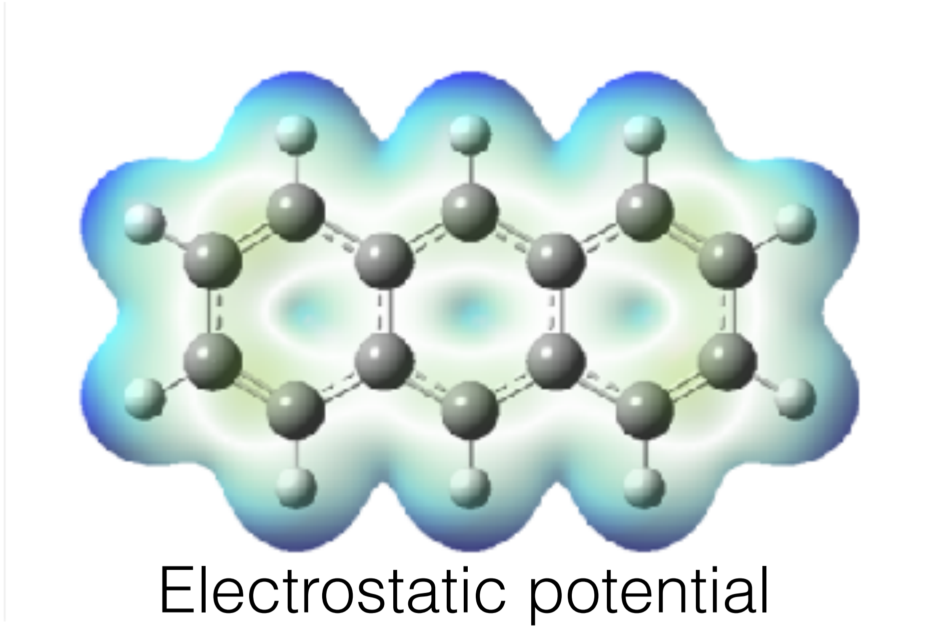
Viewing orbitals and densities

- Entos can generate cube files for densities, molecular orbitals, and electrostatic potential.
- Numerous (documented) options available.
- Cube files can be visualized with standard software.



HOMO

```
dft(
  structure( file = 'anthracene.xyz' )
  xc = B3LYP
  ao = '6-31G*'
)
cube(
  density( file = 'density.cube' )
  orbital( orbitals = homo
           file = 'orbitals.cube'
         )
  potential( file = 'esp.cube' )
)
```



Electrostatic potential
(mapped onto density)

Practical Exercises, Part #2

- Return to the Jupyter Notebook
- Explore Part 2

Summary of basic features

- Entos is a fully featured platform for simulation, property prediction, and workflow integration
- Not only the key *ab initio* methods but also…
    - Phenomenal intra-node code efficiency
    - Powerful Python integration tools
    - A notebook environment that supports on-boarding
- And additional differentiating capabilities…

Unique capabilities

Overview

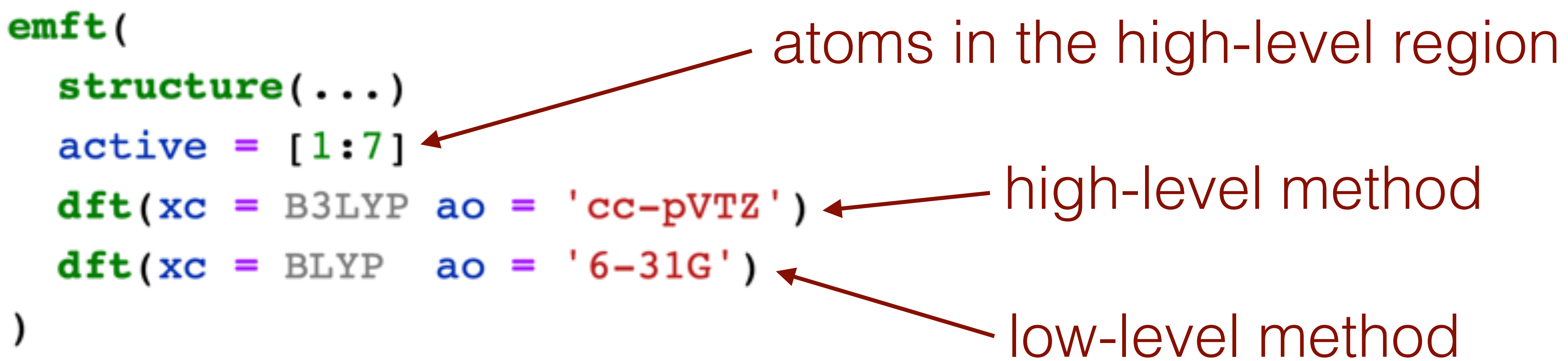
- Molecular-orbital-based Machine Learning (MOB-ML)
- Embedded mean-field theory (EMFT)
- More advanced optimization and conformational search

* MOB-ML is only available in the commercial version.

Embedded mean-field theory

Embedded mean-field theory

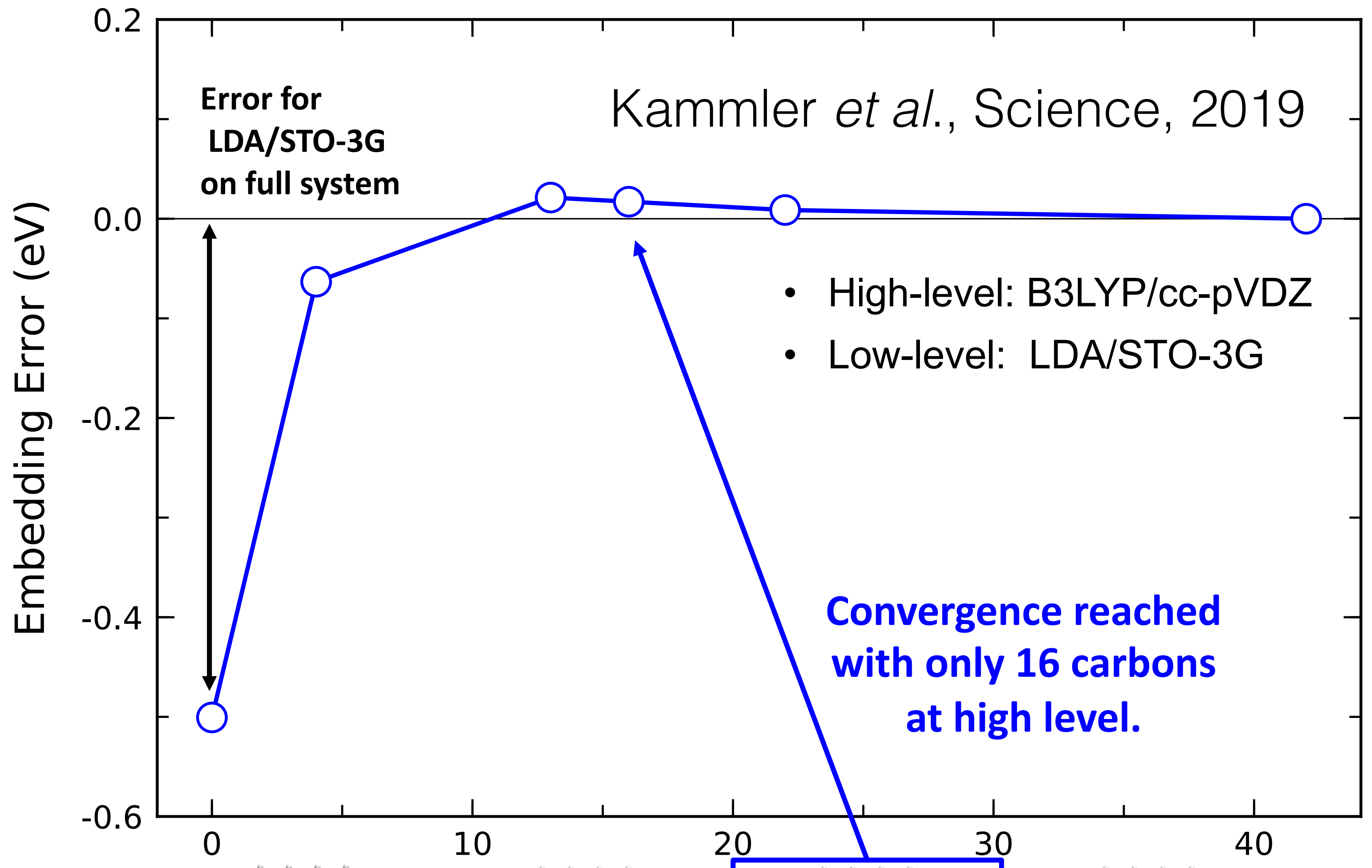
- Allows combinations of high-level and low-level SCF
- No complications with link atoms, etc
- Massively reduce cost of hybrid DFT by tuning in exact exchange where it is needed



```
emft(
  structure(...)
  active = [1:7]
  dft(xc = B3LYP ao = 'cc-pVTZ')
  dft(xc = BLYP  ao = '6-31G')
)
```

Fornace, Lee, Miyamoto, Manby, Miller, *J Chem Theory Comput.*, 11 568 (2015)

Powerful applications of EMFT

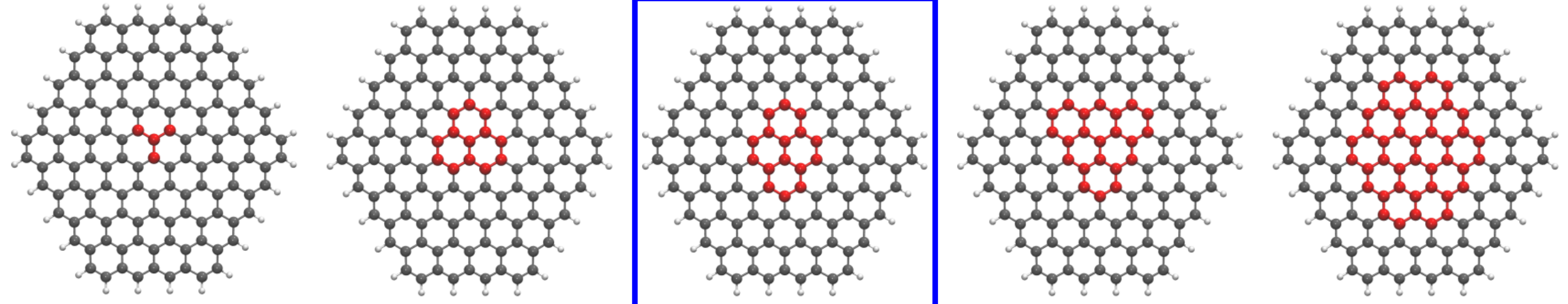


Kammler *et al.*, Science, 2019

- High-level: B3LYP/cc-pVDZ
- Low-level: LDA/STO-3G

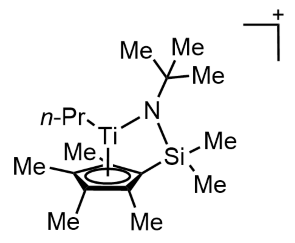
Convergence reached with only 16 carbons at high level.

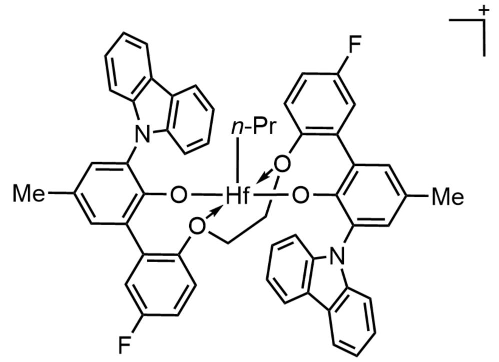
Subsystem Size

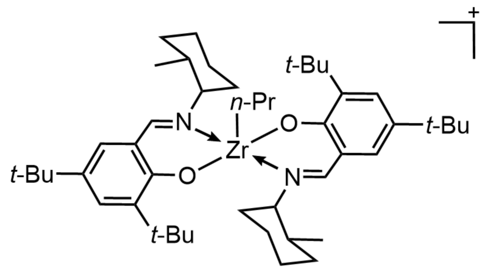


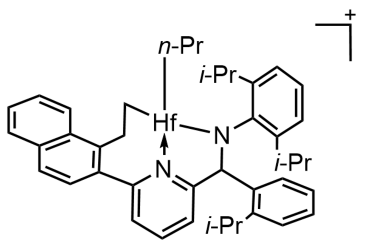
# of C at high level: 4 13 16 22 42

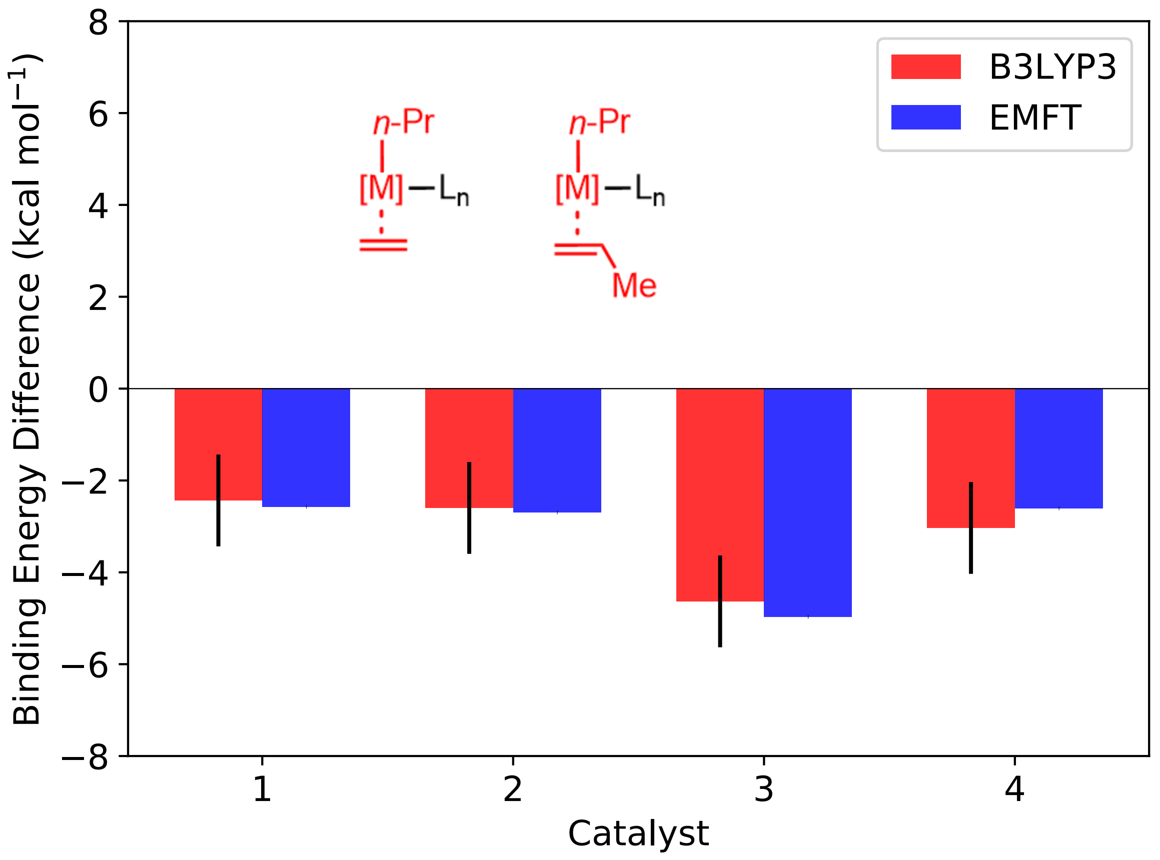
EMFT: a powerful tool for industrial catalysis


(a) Catalyst 1


(b) Catalyst 2


(c) Catalyst 3


(d) Catalyst 4



EMFT yields 20x speed-up over B3LYP without compromising accuracy.
Provides accurate binding energies, geometry optimizations, conformer stability ranking, and QM/MM solution-phase simulation.

Submitted to *JCTC*. Miller and Manby groups in collaboration with Dow Chemical.

Conformational sampling

Simulated annealing

- The ab initio MD module in Entos can be used for simulated annealing
- Dynamics is run with a specified heating and cooling protocol
- Input is structured as follows:

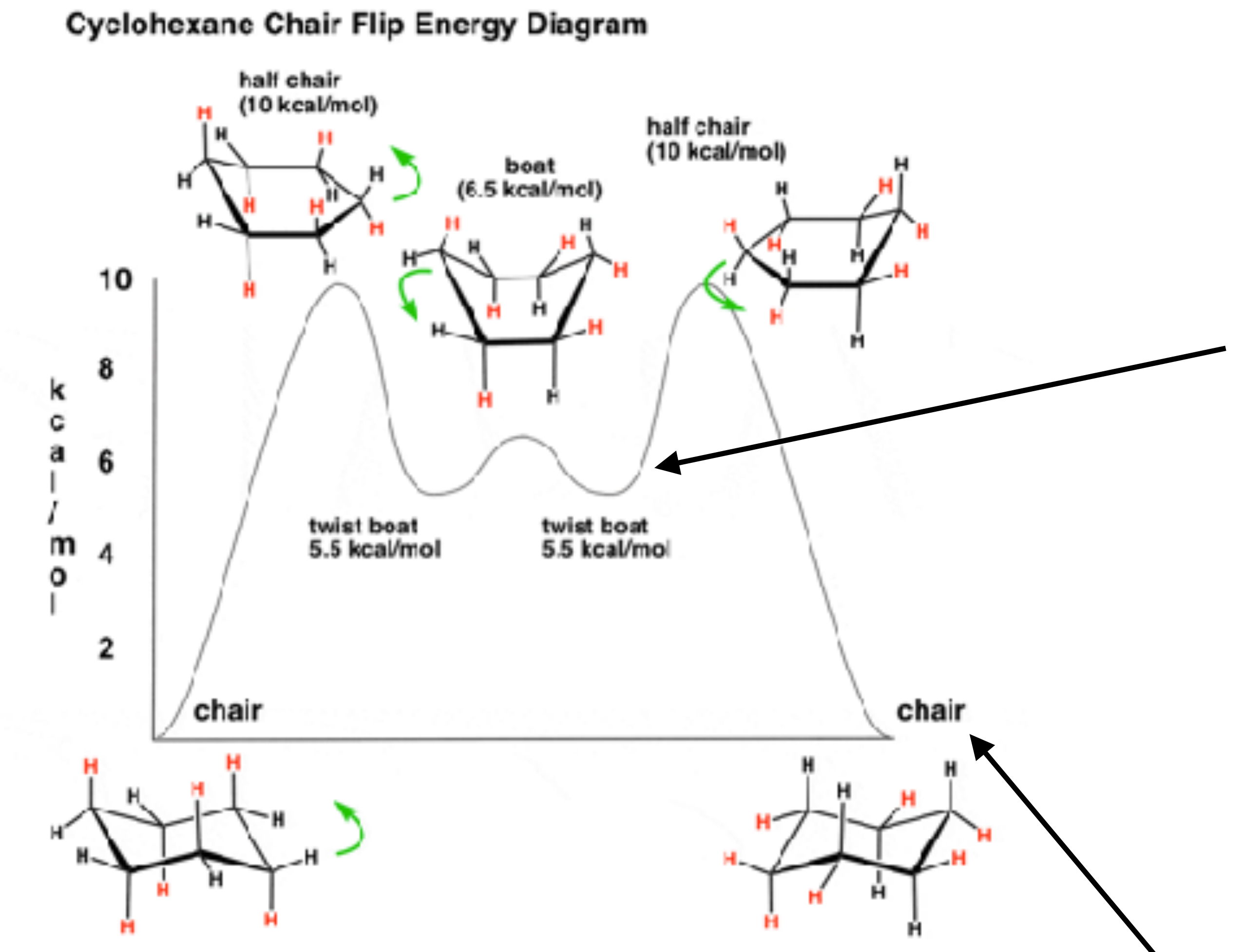
```
aimd(
   n_steps = 100000
   time_step = 1.0 fs
   structure( file = 'atoms.xyz' )
   gradient( xtb() )
   simulated_annealing(
     initial_temperature = 3000 kelvin
     final_temperature = 200 kelvin
     cooling_time = 70 ps
   )
   thermostat()
   output_steps = 50
)
```

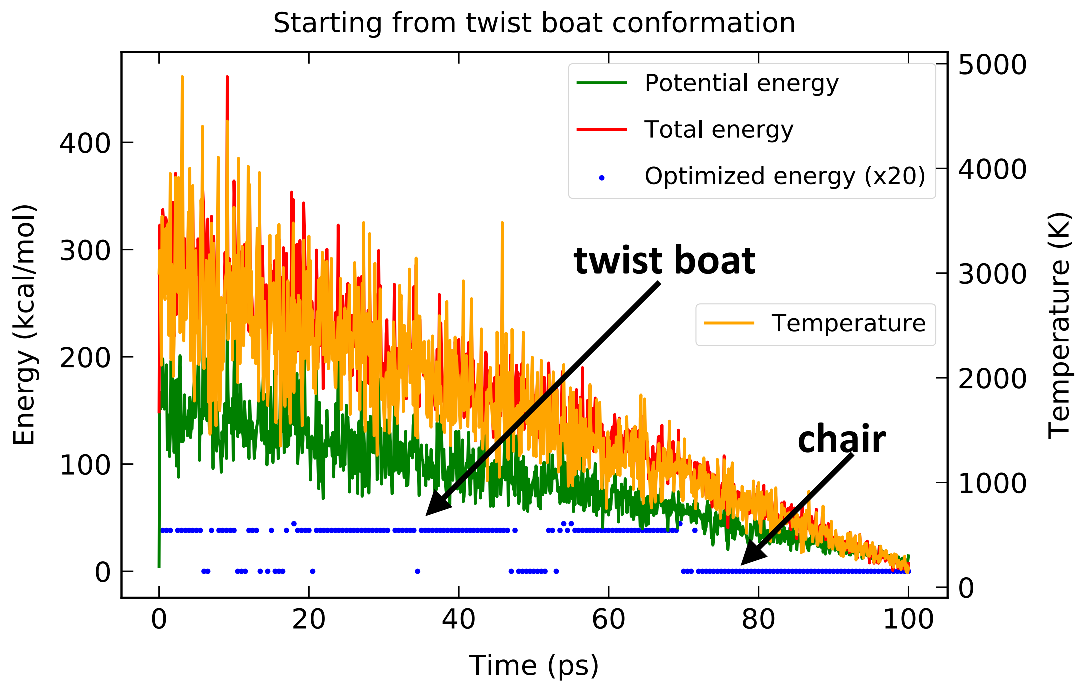
Within the basic MD input…

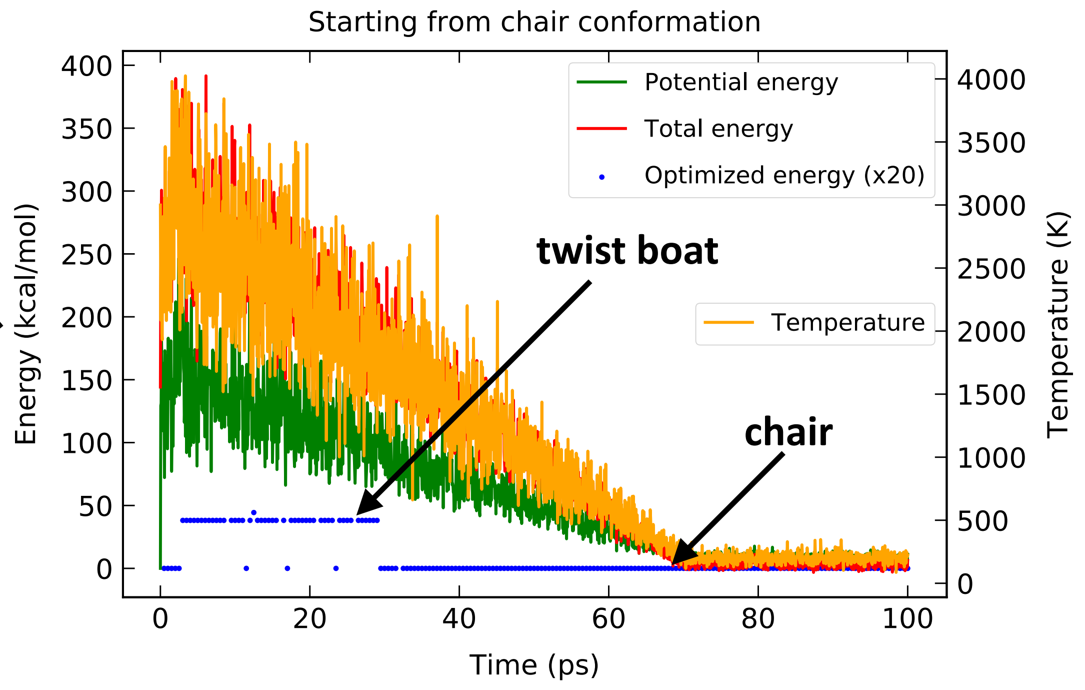
$\longleftarrow$ specify annealing…

… and temperature schedule

Example: Cyclohexane



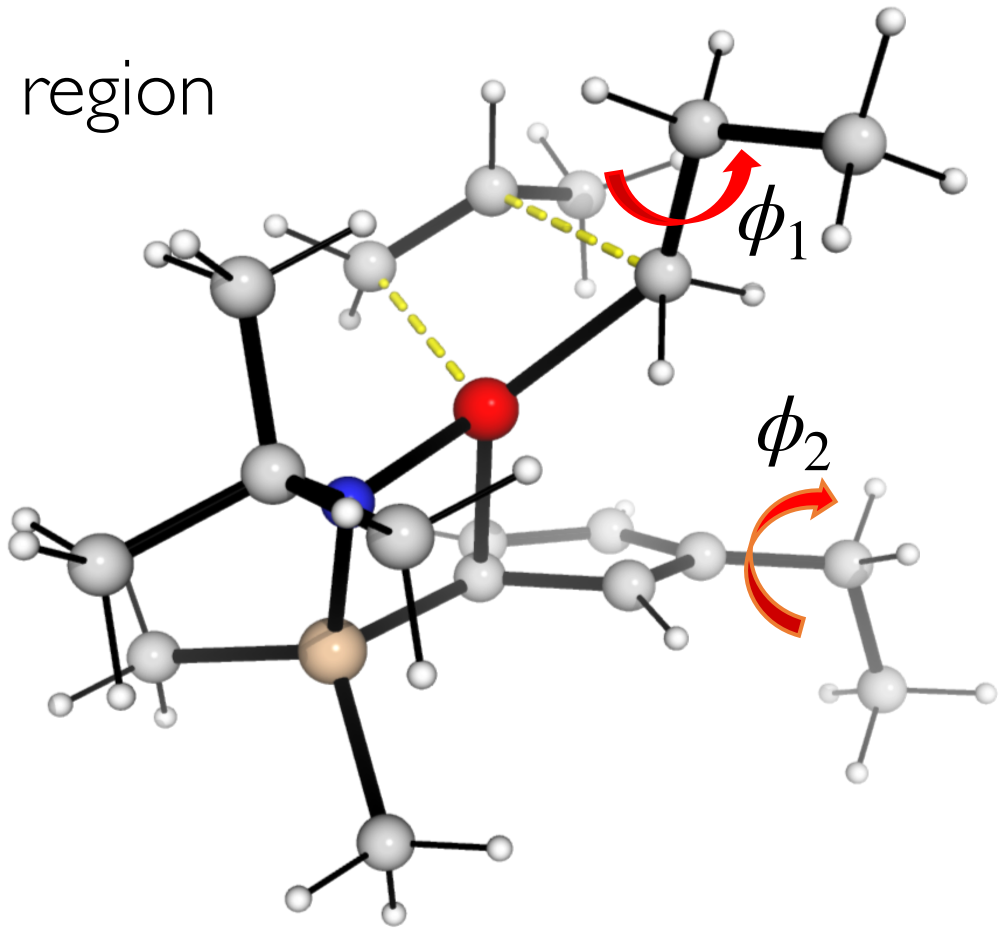




Calculation protocol:
- Simulated annealing with linear cooling schedule: 3000 K -> 200 K.
- Geometry optimize sampled configurations.
- Lower energy conformations typically found later in the simulation trajectory.

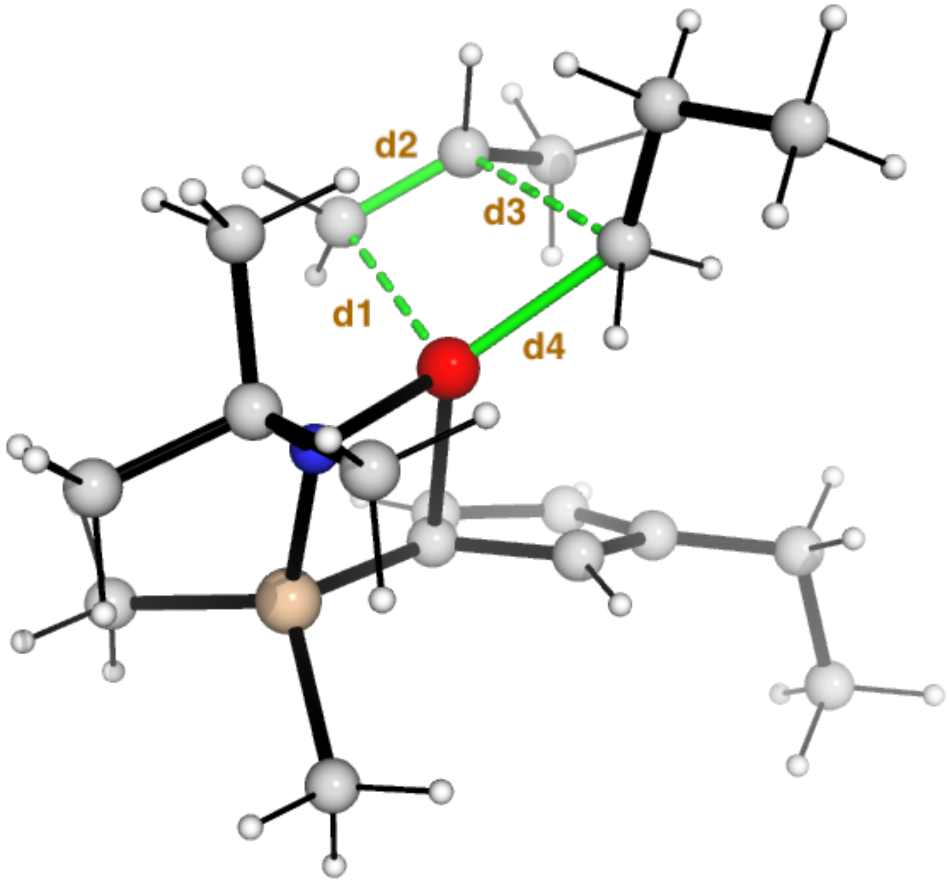
A catalysis TS example

- Ti-based C-C coupling Catalyst
- Need for conformational sampling of TS region
- Numerous conformers due to flexible modes (i.e., ϕ_1 and ϕ_2)
- Perform simulated annealing, with geometric constraints, followed by TS optimization

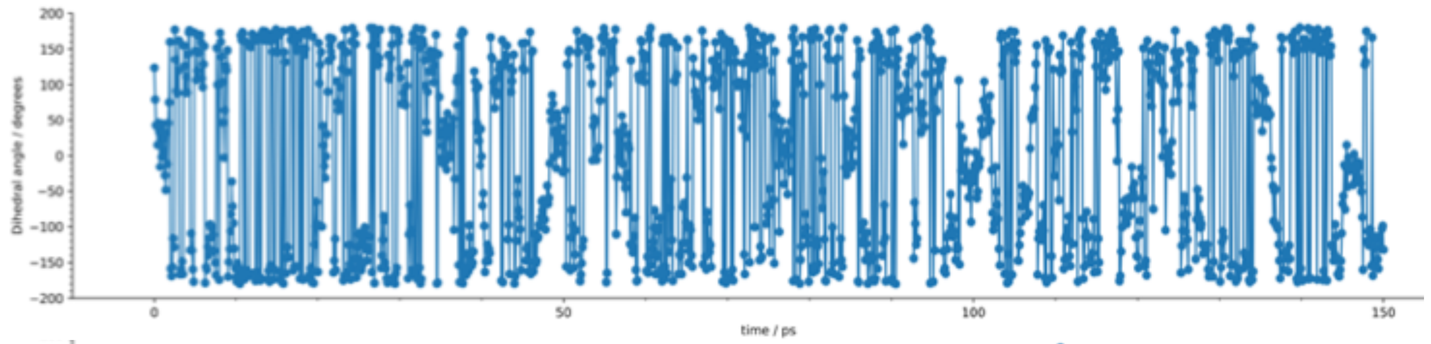


Simulated annealing

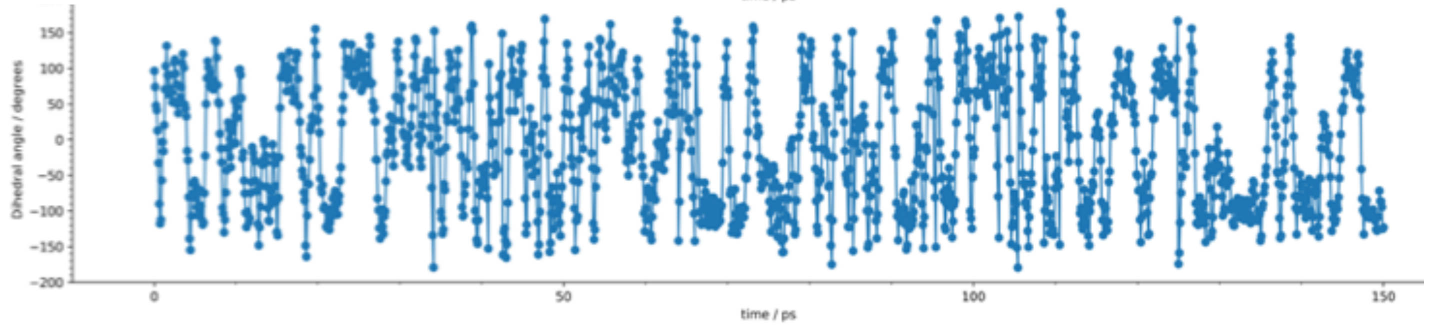
- Run annealing with green bonds restrained
- 150ps GFN1-xTB dynamics, 1fs timestep
- Heating to 800K, then back to 300K
- 150 geometries taken for further optimization



ϕ_1

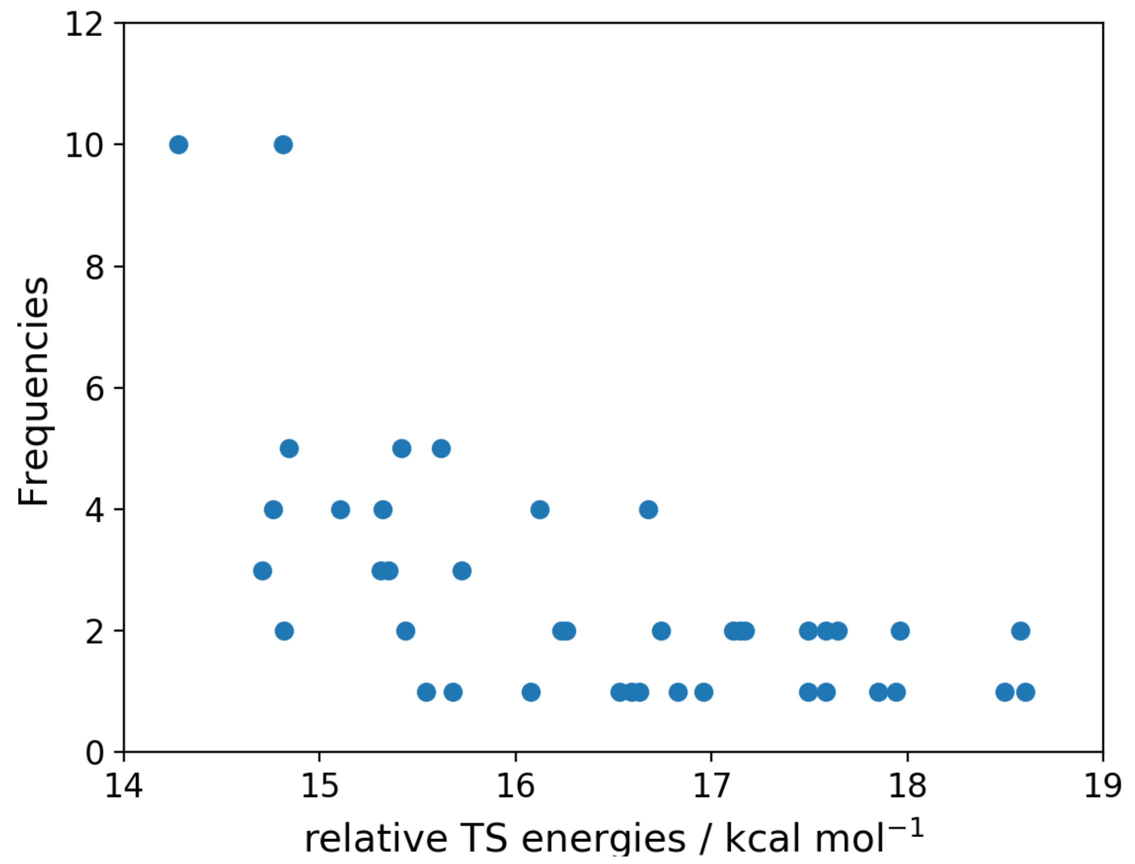


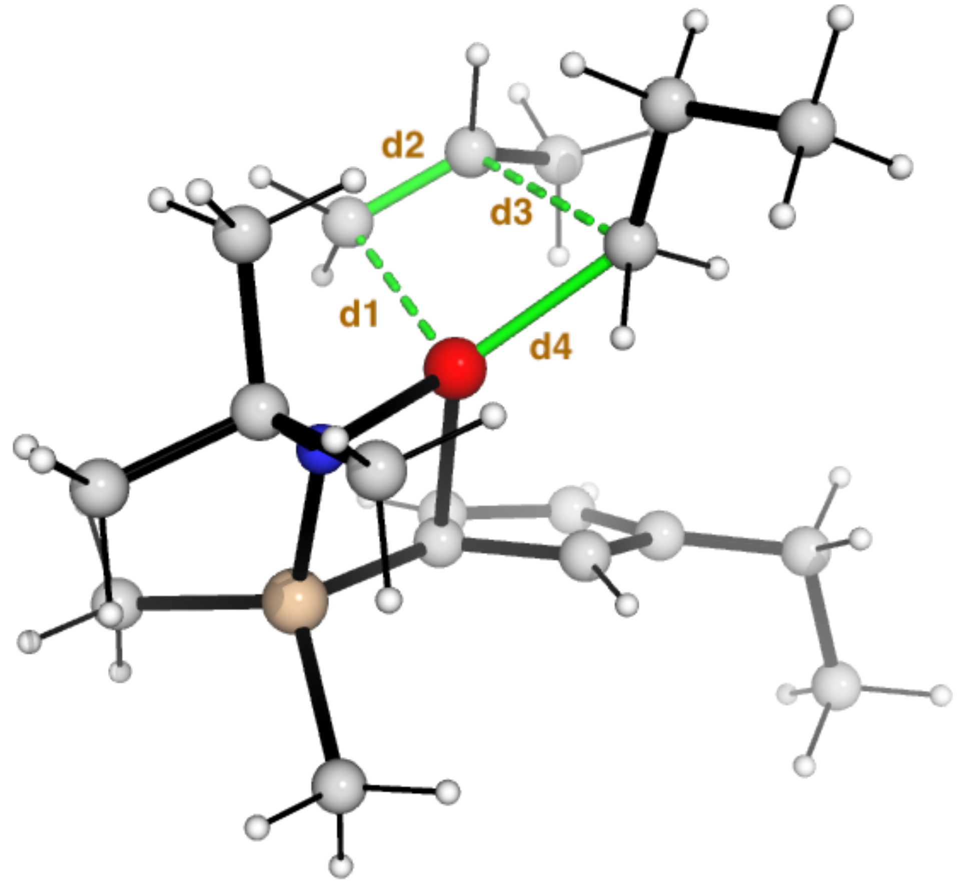
ϕ_2



Simulated annealing

- Each snapshot optimized by:
  - Constrained optimization
    (4 green bonds constrained)
  - Transition-state optimization





Torsional scan

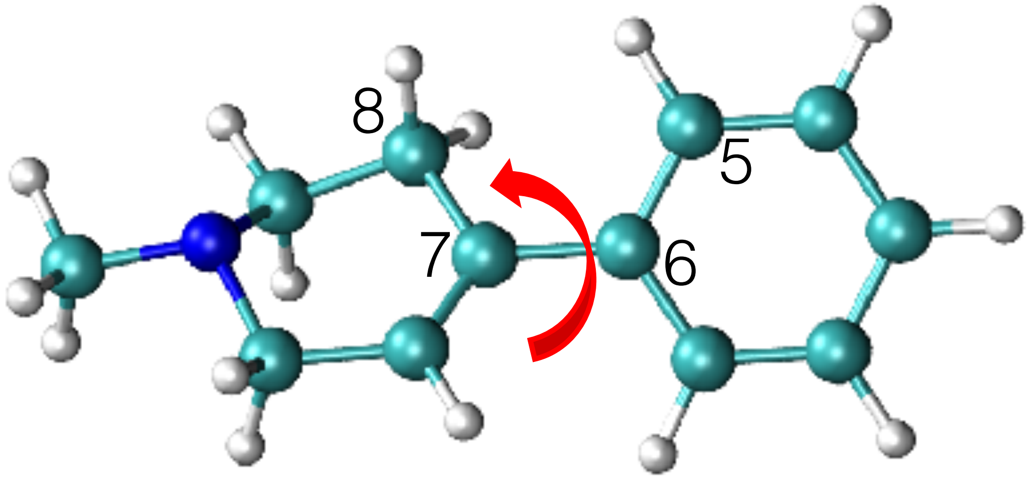
Constrained optimization

- Constraints can be added to geometry optimizations
- Bond lengths, angles and dihedrals can be constrained
- Can freeze these features, or optimize towards a target value

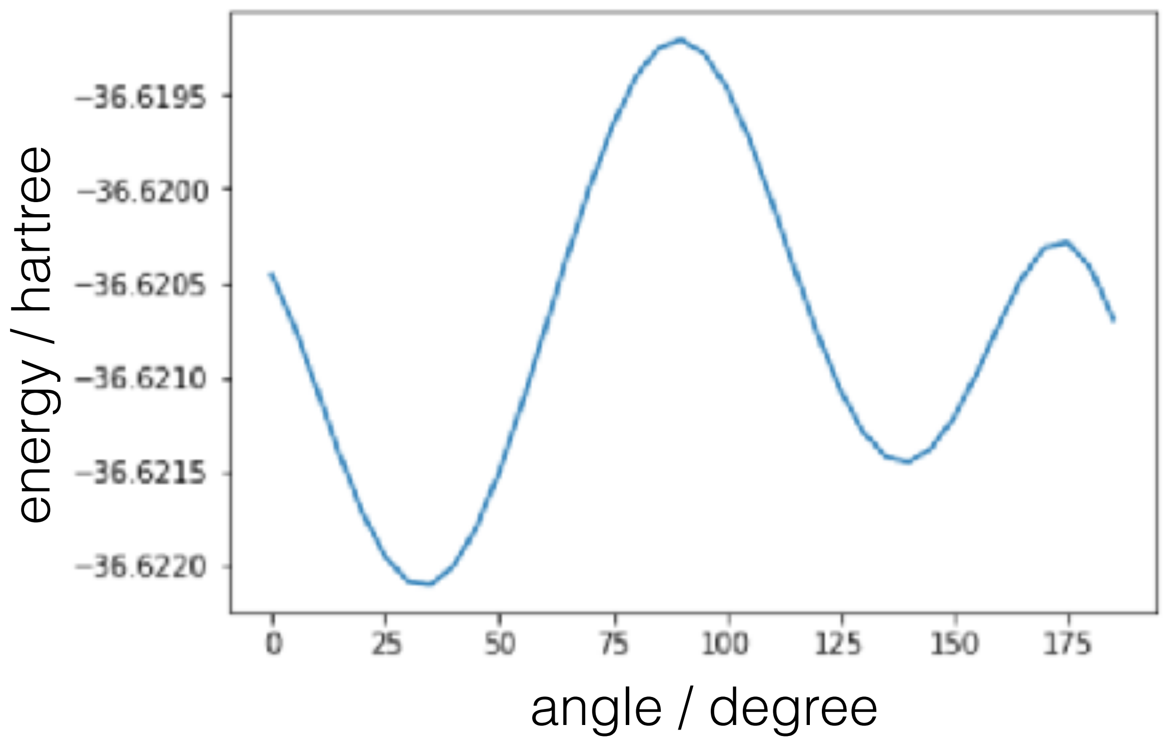
```
optimize(
  structure(molecule = methanol)
  xtb()
  bond(atoms=[1, 3] frozen = true)
  angle(atoms=[2, 1, 3] value = 108.5 degree)
  dihedral(atoms=[2, 1, 3, 4] frozen = true)
)
```

Torsional scan

```
template = """
opt := optimize(
  structure(file = 'mptp_scan.xyz')
  xtb()
  dihedral(
    atoms = [5, 6, 7, 8]
    value = {} degree
  )
  xyz_output = 'mptp_scan.xyz'
)
"""
energies = []
for x in range(0,190,5):
    output = entos(template.format(x))
    e = output.get("opt.energy")
    print(x, e)
    energies.append(e)
plt.plot(range(0,190,5), energies)
```



MPTP neurotoxin precursor



Practical Exercises, Part #3

- Return to the Jupyter Notebook
- Explore Part 3

Wrapping up…

- We hope that this overview has given you a taste of Entos, and its capabilities for enhancing scientific workflows
- Our team of engineers is actively and rapidly developing the codebase
- We value any feedback, or suggestions for new functionality
- Please continue using via:
    - Interactive cloud (Your account at jupyter.entos.info will remain until at least April 30)
    - TAMU HPC (Entos is installed and freely available)
    - Your personal device (announcements forthcoming)
- Feel free to contact us with any questions:

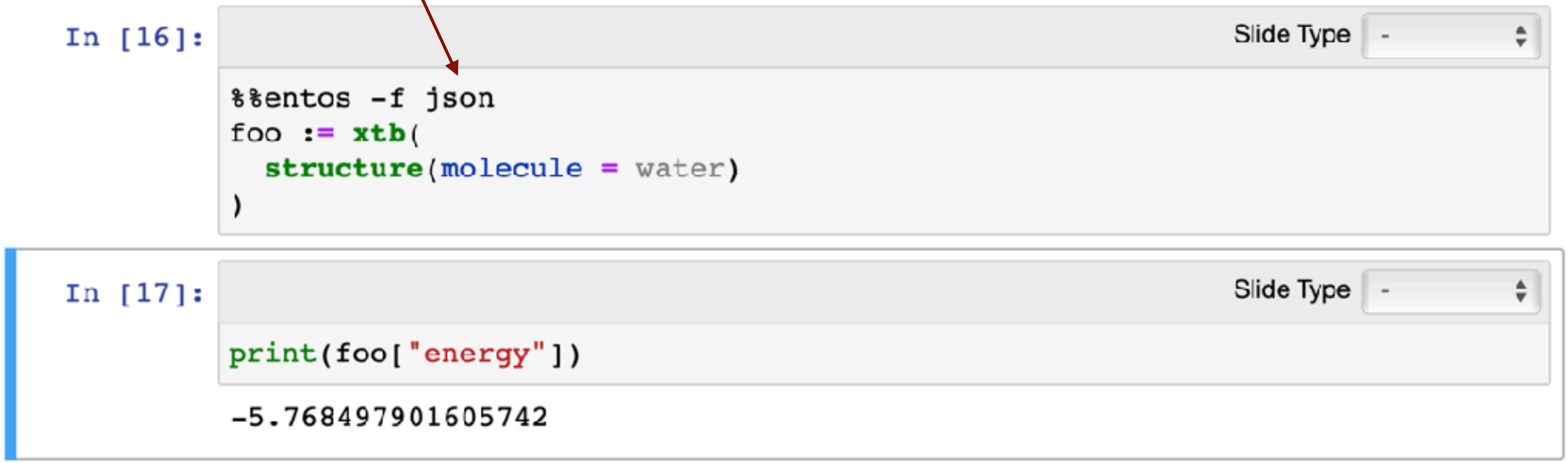
help@entos.info

Follow us via our website (entos.info) or twitter (@EntosAI) for more announcements and updates!

Supplementary: Python/Jupyter Tips

Getting results out of the magic…

… is incredibly easy



```
%%entos -f json
foo := xtb(
  structure(molecule = water)
)
```

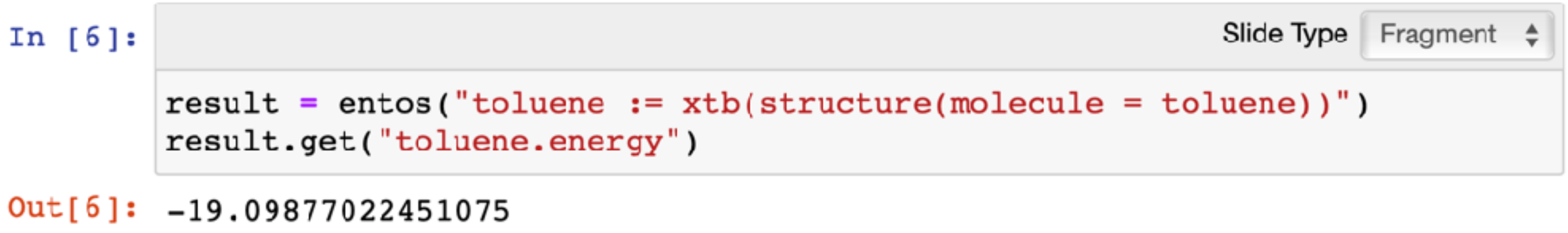
```
print(foo["energy"])
```

```
-5.768497901605742
```

The magic sets the Entos result **`foo`** to a Python dictionary with the same name. The dictionary contains the various results, including the example `"energy"` that is shown.

Calling Entos inline

- Running Entos inline produces a result that can be accessed through a path-like notation



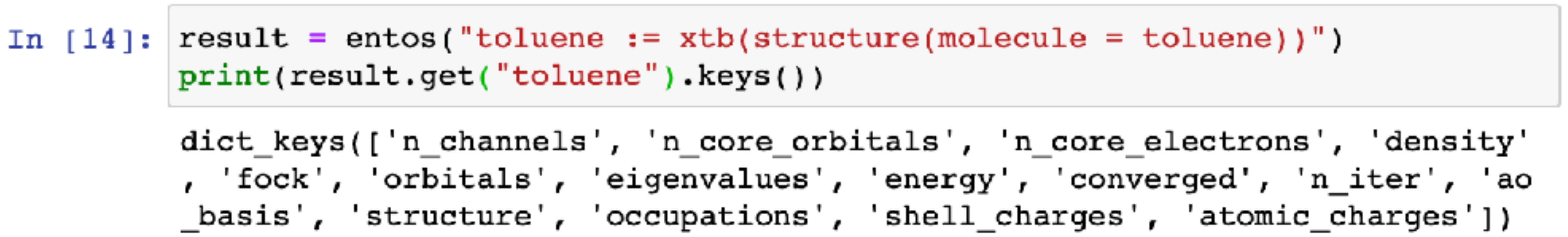
```
In [6]:
```

Slide Type Fragment

```
result = entos("toluene := xtb(structure(molecule = toluene))")
result.get("toluene.energy")
```

```
Out[6]: -19.09877022451075
```

- Result objects contain various results



```
In [14]: result = entos("toluene := xtb(structure(molecule = toluene))")
         print(result.get("toluene").keys())

         dict_keys(['n_channels', 'n_core_orbitals', 'n_core_electrons', 'density'
         , 'fock', 'orbitals', 'eigenvalues', 'energy', 'converged', 'n_iter', 'ao
         _basis', 'structure', 'occupations', 'shell_charges', 'atomic_charges'])
```

How Entos results work

- Results are assigned using the `:=` operator, and are exportable
- But can also be passed around inside Entos

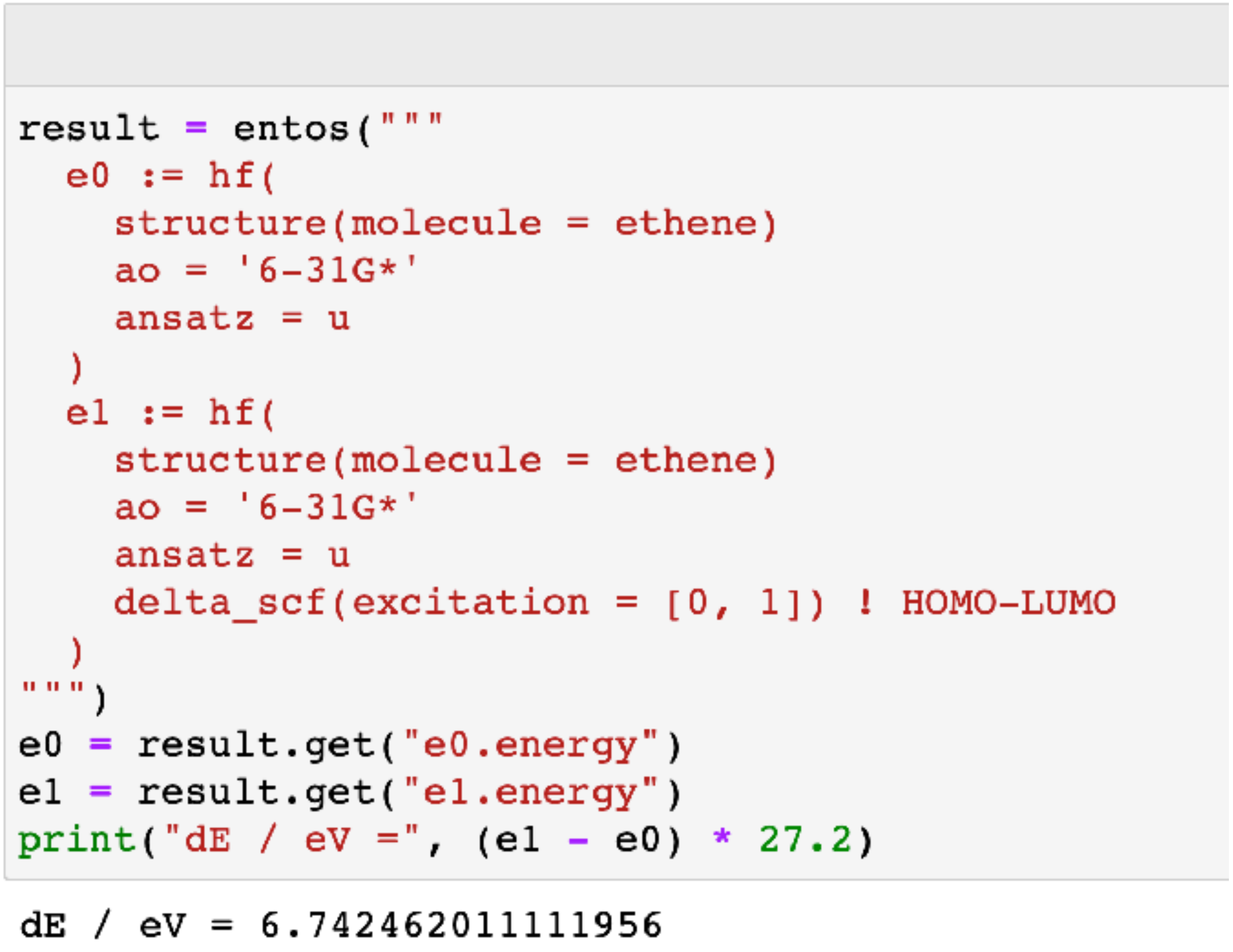
```
opt := optimize(structure(molecule = toluene) xtb())

dft(
  structure(load = opt)
  ao = 'cc-pVDZ'
  xc = B3LYP
)
```

- Other results can also be passed around (for example SCF states)

```
hf_min := hf(structure(molecule = water) ao = 'STO-3G')
dft(
  structure(molecule = water)
  load = hf_min
  ao = 'cc-pVDZ' xc = B3LYP
)
```

Example with Python integration



```
result = entos("""
  e0 := hf(
    structure(molecule = ethene)
    ao = '6-31G*'
    ansatz = u
  )
  e1 := hf(
    structure(molecule = ethene)
    ao = '6-31G*'
    ansatz = u
    delta_scf(excitation = [0, 1]) ! HOMO-LUMO
  )
""")
e0 = result.get("e0.energy")
e1 = result.get("e1.energy")
print("dE / eV =", (e1 - e0) * 27.2)
```

```
dE / eV = 6.74246201111956
```