thermochem Documentation

Release 0.8.2

Adel Qalieh

Sep 22, 2022

Contents

Release v0.8.2.

Thermochem is a Python library with some useful modules for thermodynamics and thermochemistry.

CHAPTER 1

User Guide

1.1 Installing thermochem

The first step to use the thermochem library in Python is to install it onto your computer properly.

1.1.1 Pip

Installing the latest stable version (or a specific version) of thermochem is easiest using the `pip` tool. Simply run this command in your terminal:

```
$ pip install thermochem
```

If you do not have pip installed, head over to the Python installation guide, which contains instructions on installing pip.

1.1.2 Source

Thermochem is developed on GitHub. To get the source code, you should install git if you haven't already, then run the following command in your terminal:

```
$ git clone https://github.com/adelq/thermochem.git
```

If you are not interested in developing and installing git is difficult, you can download the tarball:

```
$ curl -OL https://github.com/adelq/thermochem/tarball/master
```

Once you have a copy of the source, you can install it simply using the included `setup.py`:

```
$ cd thermochem
$ python setup.py install
```

1.2 Related Projects

The thermochem library is a general-purpose library for thermochemistry and thermodynamics. There are many other Python libraries that fill similar roles, and a combination of these libraries may serve your needs best.

- ase: The Atomic Simulation Environment contains a module for thermochemistry.
- pMuTT: The Python Multiscale Thermochemistry Toolbox contains a number of thermodynamics procedures.
- thermo: thermo has numerous thermodynamic property calculations of chemicals and mixes.
- Thermosteam: Thermosteam is a standalone thermodynamic engine that is built on thermo.

CHAPTER 2

API Documentation

2.1 Burcat

This module extracts the information provided in *Third Millennium Ideal Gas and Condensed Phase Thermochemical Database for Combustion with Updates from Active Thermochemical Tables* by A. Burcat and B. Ruscic. It needs the actual database BURCAT_THR.xml to run, which is already included in the thermochem library.

class thermochem.burcat.**Element**(*formula*, *Tmin_*, *_Tmax*, *mm*, *hfr*, *elements*)

This is a helper class. It is intended to be created via an Elementdb object but it can be used on its own. Take a look at Elementdb class for example usage.

Units are in standard units: K, J, kg. Conversion functions are provided in the external units module.

One extra feature not explained in Elementdb documentation is that it contains the number of each atom, useful for computing chemical reactions.

```
>>> db = Elementdb()
>>> weird = db.getelementdata("C8H6O2")
>>> print(weird.elements)
[('C', 8), ('H', 6), ('O', 2)]
```

cp

Computes the specific heat capacity in J/kg K at 298 K (Reference T)

cp_(*T*)

Computes the specific heat capacity in J/kg K for a given temperature

cpo(*T*)

Calculates the specific heat capacity in J/mol K

density(*p*, *T*)

Density in kg/m^3.

go(*T*)

Computes the Gibbs free energy from the sensible enthalpy in J/mol

h(*T*)
Computes the total enthalpy in J/kg

ho(*T*)
Computes the sensible enthalpy in J/mol

so(*T*)
Computes entropy in J/mol K

class thermochem.burcat.**Elementdb**
Class that reads the Alexander Burcat's thermochemical database for combustion.

```
>>> db = Elementdb()
>>> oxygen = db.getelementdata("O2 REF ELEMENT")
>>> print(oxygen)
<element> O2 REF ELEMENT
>>> print('molar mass', oxygen.mm)
molar mass 0.0319988
>>> print('heat capacity', round(oxygen.cp, 6))
heat capacity 918.078952
```

The reference temperature for enthalpy is 298.15 K

```
>>> print('entropy', round(oxygen.so(298), 6))
entropy 205.133746
>>> print('gibbs free energy', round(oxygen.go(298), 6))
gibbs free energy -61134.262901
```

There's a search function. It is very useful because some names are a bit tricky. Well, not this one.

```
>>> db.search("AIR")
['AIR']
>>> air = db.getelementdata("AIR")
>>> print('air molar mass', air.mm)
air molar mass 0.02896518
>>> print('heat capacity', round(air.cp, 6))
heat capacity 1004.776251
>>> print(round(air.density(101325, 298), 6))
1.184519
```

The element database can create also mixtures. It returns an instance of Mixture object that can give you the same as the Element class for any mixture.

```
>>> mix = db.getmixturedata([("O2 REF ELEMENT", 20.9476),     ("N2  REF ELEMENT",␣
↪78.084),     ("CO2", 0.0319),     ("AR REF ELEMENT", 0.9365),     ])
>>> print(mix)
<Mixture>:
    O2 REF ELEMENT at 20.9476
    N2  REF ELEMENT at 78.084
    CO2 at 0.0319
    AR REF ELEMENT at 0.9365
>>> print(round(mix.cp, 6))
1004.722171
>>> print(round(mix.mm, 6))
0.028965
```

getelementdata(*formula*)
Returns an element instance given the name of the element.

getmixturedata(*components*)
Creates a mixture of components given a list of tuples containing the formula and the volume percent

search(*formula*)
List all the species containing a string. Helpful for interactive use of the database.

class thermochem.burcat.**Mixture**(*config='vol'*)
Class that models a gas mixture. Currently, only volume (molar) compositions are supported.

You can iterate through all its elements. The item returned is a tuple containing the element and the amount.

```
>>> db = Elementdb()
>>> mix = db.getmixturedata([("O2 REF ELEMENT", 20.9476),     ("N2  REF ELEMENT",␣
↪78.084),     ("CO2", 0.0319),     ("AR REF ELEMENT", 0.9365),     ])
>>> mix_list = [(e[0], round(e[1], 6)) for e in mix]
>>> for e in mix_list: print(e)
(<element> O2 REF ELEMENT, 20.9476)
(<element> N2  REF ELEMENT, 78.084)
(<element> CO2, 0.0319)
(<element> AR REF ELEMENT, 0.9365)
```

You can get elements either by index or by value.

```
>>> print(mix['CO2'])
(<element> CO2, 0.0319)
```

You can also delete components of a mixture. Needed by the MoistAir class

```
>>> mix.delete('CO2')
>>> print(mix)
<Mixture>:
    O2 REF ELEMENT at 20.9476
    N2  REF ELEMENT at 78.084
    AR REF ELEMENT at 0.9365
```

add(*component*, *prop*)
Add a component to the mixture

cp
Computes the heat capacity at room temperature, 298.15K. Results in J/kg K.

cp_(*T*)
Computes the heat capacity at a given temperature in J/kg K.

delete(*formula*)
Delete a formula from the mixture

density(*p*, *T*)
Computes the density for a given mix of gases in kg/m^3

The equivalent R for a mix is $R_m = \frac{R_u}{M_n}$, where M_n is the equivalent molar mass for the mix.

extensive(*attr*, *T*)
Computes the extensive value for a mix. Remember that an extensive value depends on the amount of matter. Enthalpy and volume are extensive values.

$$ext = \frac{1}{N_m M_m} \sum_i N_i M_i ext_i$$

go(*T*)
Estimate the Gibbs free energy using the sensible enthalpy of the mixture in J/mol.

h(*T*)
Estimate the total enthalpy of the mixture in J/kg.

ho(*T*)
Estimate the sensible enthalpy of the mixture in J/mol.

mm
Computes the equivalent molar mass for a mix

$$M_m = \frac{1}{N_m} \sum_i N_i M_i$$

so(*T*)
Estimate the entropy of the mixture in J/mol K.

2.2 CODATA

Fundamental Physical Constants

These constants are taken from CODATA Recommended Values of the Fundamental Physical Constants: 2002. They may be found at physics.nist.gov/constants. The values are stored in the dictionary physical_constants as a tuple containing the value, the units, and the relative precision, in that order. All constants are in SI units unless otherwise stated.

Several helper functions are provided:

value(key) returns the value of the physical constant. unit(key) returns the units of the physical constant. precision(key) returns the relative precision of the physical constant. find(sub) prints out a list of keys containing the string sub.

thermochem.codata.**value**(*key*)
value indexed by key

thermochem.codata.**unit**(*key*)
unit indexed by key

thermochem.codata.**precision**(*key*)
relative precision indexed by key

thermochem.codata.**find**(*sub*)
list all keys containing the string sub

2.3 Combustion

class thermochem.combustion.**Combustor**(*fuels*, *phi*, *db*)
Combustor that is able to characterize the combustion of a mixture of fuels

adiabatic_flame_temp(*T*)
This is the adiabatic flame temp for the given mixtures of reactants and products. If you want the true adiabatic flame temperature remember to set the equivalence ratio to 1. Otherwise you will always get lower temperatures.

heat_of_comb(*T*)
Calculates the heat of combustion per kg of fuel. Checked ok

class thermochem.combustion.**SimpleCombustor**(*fuel*, *phi*, *db*)
This class models a simple combustor that uses fuel as a reductor and air as a single oxidizer. The combustion is complete, no CO nor radicals are formed.

It only supports fuels with C, N, H and O. If you put a more complicated fuel it will ignore the rest of atoms to balance the reaction.

adiabatic_flame_temp(*T*)
This is the adiabatic flame temp for the given mixtures of reactants and products. If you want the true adiabatic flame temperature remember to set the equivalence ratio to 1. Otherwise you will always get lower temperatures.

heat_of_comb(*T*)
Calculates the heat of combustion per kg of fuel. Checked ok

thermochem.combustion.**balance**(*fuel*, *am*, *phi*)
Function that balances the combustion equation given any simple fuel element

Fuel Formula of a Burcat's database fuel as a string. The fuel can only contain C, H, O and N. Other atoms will be ignored.

Phi Equivalence ratio for air.

thermochem.combustion.**balance_mix**(*fuels*, *phi*)
function that balances the combustion equation given a mix of fuels.

Fuels type Mixture. Simple fuels formed only by C, H, O and N and the amount of each one

Phi Equivalence ratio for air.

2.4 Constants

thermochem.constants.**C2F**(*C*)
Convert Celsius to Fahrenheit

thermochem.constants.**C2K**(*C*)
Convert Celsius to Kelvin

thermochem.constants.**F2C**(*F*)
Convert Fahrenheit to Celsius

thermochem.constants.**F2K**(*F*)
Convert Fahrenheit to Kelvin

thermochem.constants.**K2C**(*K*)
Convert Kelvin to Celsius

thermochem.constants.**K2F**(*K*)
Convert Kelvin to Fahrenheit

thermochem.constants.**lambda2nu**(*lambda_*)
Convert wavelength to optical frequency

thermochem.constants.**nu2lambda**(*nu*)
Convert optical frequency to wavelength

2.5 IAPWS

class thermochem.iapws.**Water**
Taken from

The International Association for the Properties of Water and Steam. Lucerne, Switzerland. August 2007. Revised Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam.

Functions implemented:

Saturation line h(p,T) region1, region2 and no warnings yet

T_ph (*p*, *h*)
Returns the temperature (K) given the pressure (MPa) and specific enthalpy (kJ/kg). Only region 2a implemented (p<4MPa) (Reimplement).

```
>>> w = Water()
>>> w.T_ph(3,500)[0]
391.798...
>>> w.T_ph(3,500)[1]
4.1313...e+21
```

Tsat (*p*)
Returns the saturation temperature of water at a given pressure.

Remember that pressure must be between 0.000611213MPa (triple point) and 22.064MPa (critical point)

Temperatures in K, pressures in MPa.

```
>>> w = Water()
>>> w.Tsat(100000)
372.7559186113...
>>> w.Tsat(1200000)
461.1146416213...
>>> w.Tsat(100)
Traceback (most recent call last):
  File "/usr/lib/python2.5/doctest.py", line 1228, in __run
    compileflags, 1) in test.globs
  File "<doctest __main__.Water.Tsat[3]>", line 1, in <module>
    w.Tsat(100)
  File "iapws.py", line 193, in Tsat
    raise ValueError('No saturation temperature for this pressure')
ValueError: No saturation temperature for this pressure
>>> w.Tsat(101325)
373.12430000048056
```

h (*p*, *T*)
Returns specific enthalpy (J/kg) for a given pressure (Pa) and Temperature (K).

```
>>> w = Water()
>>> round(w.h(3000000,300), 6)
115.331273
>>> w.h(3500,300)
2549.9114508400203
```

There are also error codes

Results checked against the reference.

psat (*T*)
Returns the saturation pressure of water at a given temperature.

Remember that temperature must be between 273.15K (triple point) and 647.096K (critical point)

Temperatures in K, Pressures in Pa.

```
>>> w = Water()
>>> w.psat(300)
3536.5894130130105
>>> w.psat(130)
Traceback (most recent call last):
  File "/usr/lib/python2.5/doctest.py", line 1228, in __run
    compileflags, 1) in test.globs
  File "<doctest __main__.Water.psat[2]>", line 1, in <module>
    w.psat(130)
  File "iapws.py", line 153, in psat
    'No saturation pressure for this temperature')
ValueError: No saturation pressure for this temperature
>>> w.psat(700)
Traceback (most recent call last):
  File "/usr/lib/python2.5/doctest.py", line 1228, in __run
    compileflags, 1) in test.globs
  File "<doctest __main__.Water.psat[3]>", line 1, in <module>
     w.psat(700)
  File "iapws.py", line 146, in psat
    'No saturation pressure for this temperature')
ValueError: No saturation pressure for this temperature
```

2.6 JANAF

This module gets thermodynamic data from the JANAF database. Files are downloaded from the NIST servers as needed and then cached locally.

Zack Gainsforth

Funding by NASA

class thermochem.janaf.**JanafPhase**(*rawdata_text*)

Class which is created by Janafdb for a specific phase.

It reads in the JANAF data file and produces functions which interpolate the thermodynamic constants.

Tr stands for reference temperature and is 298.15 K

```
>>> db = Janafdb()
>>> p = db.getphasedata(formula='O2Ti', name='Rutile', phase='cr')
>>> p.cp([500, 550, 1800]).astype(int).tolist()
[67, 68, 78]
>>> print(p.S([500, 550, 1800]))         # Entropy in J/mol/K
[  82.201    88.4565  176.876 ]
>>> print(p.gef([500, 550, 1800]))       # [G-H(Tr)]/T in J/mol/K
[  57.077   59.704  115.753]
>>> print(p.hef([500, 550, 1800]))       # H-H(Tr) in kJ/mol
[  12.562    15.9955  110.022 ]
>>> print(p.DeltaH([500, 550, 1800]))    # Standard enthalpy of formation in kJ/mol
[-943.670  -943.2295 -936.679 ]
>>> print(p.DeltaG([500, 550, 1800]))    # Gibbs free enegy in kJ/mol
[-852.157  -843.0465 -621.013 ]
>>> p.logKf([500, 550, 1800]).astype(int).tolist() # Equilibrium constant of␣
↪formation.
[89, 80, 18]
>>> print(p.cp(1000))                    # Heat capacity in J/mol/K
```

(continues on next page)

(continued from previous page)

```
74.852
>>> print(p.cp(50000))                  # Example of erroneous extrapolation.
Traceback (most recent call last):
    ...
ValueError: A value in x_new is above the interpolation range.
```

class thermochem.janaf.**Janafdb**

Class that reads the NIST JANAF tables for thermodynamic data.

Data is initially read from the web servers, and then cached.

Examples

```
>>> rutile = Janafdb().getphasedata(name='Rutile')
```

To load thermodynamic constants for TiO2, rutile.

getphasedata(*formula=None*, *name=None*, *phase=None*, *filename=None*, *cache=True*)

Returns an element instance given the name of the element. formula, name and phase match the respective fields in the JANAF index.

Parameters

- **formula** (*str*) – Select records that match the chemical formula
- **name** (*str*) – Select records that match the chemical/mineral name
- **phase** (*str*) – Select records that match the chemical phase. Must be one of the following valid phases: cr, l, cr,l, g, ref, cd, fl, am, vit, mon, pol, sln, aq, sat
- **filename** (*str*) – Select only records that match the filename on the website, which is very unique.
- **cache** (*bool, default True*) – Whether to cache the Janaf database. Setting this to false will download the Janaf database every time it is used.

Examples

```
>>> db = Janafdb()
>>> db.getphasedata(formula='O2Ti', phase='cr')
Traceback (most recent call last):
    ...
ValueError: There are 2 records matching this pattern:
    ...
Please select a unique record.
>>> db.getphasedata(formula='Oxyz')
Traceback (most recent call last):
    ...
ValueError: Did not find a phase with formula = Oxyz
            Please provide enough information to select a unique record.
            Also check that you didn't eliminate the record you want by␣
↪choosing too many constraints where one or more constraint is incorrect.
>>> db.getphasedata(formula='Oxyz', phase='l')
Traceback (most recent call last):
    ...
ValueError: Did not find a phase with formula = Oxyz, phase = l
```

(continues on next page)

(continued from previous page)

```
            Please provide enough information to select a unique record.
            Also check that you didn't eliminate the record you want by␣
↪choosing too many constraints where one or more constraint is incorrect.
>>> FeO = db.getphasedata(formula='FeO', phase='cr,l')
>>> print(FeO)
<thermochem.janaf.JanafPhase object at 0x...>
  Iron Oxide (FeO)  Fe1O1(cr,l)
    Cp(298.15) = 49.915 J/mol/K
    S(298.15) = 60.752 J/mol/K
    [G-H(298.15)]/298.15 = 60.752 J/mol/K
    H-H(298.15) = 0.000 J/mol/K
    Delta_fH(298.15) = -272.044 kJ/mol
    Delta_fG(298.15) = -251.429 kJ/mol
    log(Kf((298.15)) = 44.049
```

search (*searchstr*)

List all the species containing a string. Helpful for interactive use of the database.

Parameters **searchstr** (*str*) – The search string to look for

Returns Dataframe containing valid phases

Return type pandas.DataFrame

Examples

```
>>> db = Janafdb()
>>> s = db.search('Rb-')
>>> print(s)
     formula           name phase filename
1710     Rb-  Rubidium, Ion     g   Rb-007
>>> s = db.search('Ti')
>>> print(len(s))
88
```

2.7 Psicrometry

class thermochem.psicrometry.**MoistAir**(*gas*)

Class that models a moist gas. The computations in this class are a bit tricky because the enthalpy computations for steam far from atmospheric pressure using the bcura data have severe deviations. Considering water an ideal gas is a too strong assumption. This means that water data is computed using the IAPWS data for water an steam.

The trickiest part comes with the fact that the enthalpy reference for the IAPWS tables is the water's triple point when the enthalpy reference for the bcura tables is the absolute zero.

The IAPWS reference is the leading one.

MoistAir does not inherit from Mixture because of this.

phi (*p*, *T*)

Relative moisture given pressure and temperature.

wet_bulb_T (*p*)

Wet bulb temperature for a given pressure

2.8 Units

class thermochem.units.**Enthalpy**(*data*)
Class that models an enthalpy measure with conversion utilities

Supported units are

- Joule per kg (default)
- Kilojoule per kg (kJkg)
- Kilocalorie per kg (kcalkg)
- BTU per pound (Btulb)

```
>>> h = Enthalpy(1000)
>>> h.kJkg
1.0
>>> h.kcalkg
0.2390057361376...
>>> h.Btulb
0.42992261392949266
```

Btulb
Convert enthalpy to BTU per pound

kJkg
Convert enthalpy to kilojoules per kg

kcalkg
Convert enthalpy to kilocalories per kg

unit(*units='si'*)
Specify enthalpy units, default is joules per kg.

Unit one of: 'si': joules per kg 'kJkg': kilojoules per kg 'kcalkg': kilocalories per kg 'Btulb': BTU per pound

class thermochem.units.**Length**(*data*)
Class that models a length measure with conversion utilities

Supported units are

- meter (default)
- millimeter (mm)
- inch (inch)
- foot (ft)

```
>>> l = Length(1).unit('inch')
>>> round(l.mm, 1)
25.4
>>> l.ft
0.0833333333333...
>>> round(l, 4)
0.0254
```

ft
Convert length to feet

inch
Convert length to inches

mm
Convert length to millimeters

unit(*units='m'*)
Specify length units, default is meters (m).

Unit one of 'm', 'mm', 'inch', 'ft'.

class thermochem.units.**Massflow**(*data*)
Class that models a mass flow measure with conversion utilities

Supported units are

- kg per second (default)
- kg per hour (kgh)
- pounds per second (lbs)
- pounds per hour (lbh)

kgh
Convert mass flow to kg per hour

lbh
Convert mass flow to pounds per hour

lbs
Convert mass flow to pounds per second

unit(*units='kgs'*)
Specify mass flow units, default is kg per second.

Unit one of: 'kgs': kg per second 'kgh': kg per hour 'lbs': pounds per second 'lbh': pounds per hour

class thermochem.units.**Massflowrate**(*data*)
Class that models a mass flow measure with conversion utilities

Supported units are

- $\frac{kg}{s\ m^2}$ (default)
- $\frac{lb}{s\ ft^2}$ (Btu)

class thermochem.units.**Pressure**(*data*)
Class that models a Pressure measure with conversion utilities

Suported units are

- Pascal (Pa)
- Mega Pascal (MPa)
- Bar (bar)
- Pound per square inch (psi)
- Atmosphere (atm)
- Millimeters of water column (mmwc)
- Torricelli (torr)

Normal instantiation is pressure in Pa. How much is an athmosphere?

```
>>> p = Pressure(1.0).unit('atm')
>>> p
101325.0
>>> p.torr
760.0
>>> p.mmwc
10285.839999999998
>>> p.psi
14.69594877551345
```

MPa
Convert pressure to megapascals

atm
Convert pressure to atmospheres (atm)

bar
Convert pressure to bars

mmwc
Convert pressure to millimeters of water column (mmwc)

psi
Convert pressure to pounds per square inch (psi)

torr
Convert pressure to torrs

unit(*units='Pa'*)
Specify pressure units, default is Pascal (Pa).

Unit one of 'Pa', 'MPa', 'bar', 'psi', 'atm', 'mmwc', 'torr'.

class thermochem.units.**Temperature**(*data*)
Class that models a temperature measure with conversion utilities

Supported units are

- Kelvin
- Celsius
- Fahrenheit

Normal instantiation is a temperature in Kelvin

```
>>> T = Temperature(100)
>>> T
100.0
```

But you can instantiate and specify if unit is Celsius or Fahrenheit

```
>>> T = Temperature(100).unit('F')
>>> T
310.92777777777775
```

Unit conversion is as easy as it gets.

```
>>> T.C
37.777777777777...
>>> T.F
99.999999999999...
```

You can compute with temperatures because inherits from the float built-in

```
>>> T1 = Temperature(200)
>>> T2 = Temperature(0).unit('C')
>>> round(T1+T2, 2)
473.15
```

If you don't want to use the class' attribute you can use the function *getattr* to get a value using the unit code.

```
>>> getattr(T,'C')
37.77777777777...
```

C
: Convert temperature to Celsius

F
: Convert temperature to Fahrenheit

unit (*units='K'*)
: Specify temperature units, default is Kelvin.

  Unit one of 'K', 'C', or 'F' for Kelvin, Celsius, or Fahrenheit

CHAPTER 3

Contributor Guide

3.1 Authors

Thermopy is written and maintained by Adel Qalieh and various contributors:

- Guillem Borrell i Nogueras <guillem@torroja.dmt.upm.es>
- Adel Qalieh <adelq@sas.upenn.edu>
- Zack Gainsforth <zsg@gainsforth.com>

The development of this package was originally written by Guillem Borrel i Nogueras and funded by Vulcano Sadeca S.A.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index