Thermopy v0.5.1

Felipe M. Vieira <fmv1992@gmail.com>

November 8, 2015

This is the documentation for Thermopy v0.5.1 .

Contents

1 Dependencies

The dependencies for this package are **numpy** and **scipy**. Tests ran successfully with 1.9.3 and 0.16.0 respectively.

2 Nomenclature

The naming of functions, methods and compounds have no abbreviations as they differ from each literature source (like *v* or *u to denote speed*). Unnecessary long names like the prefix *standard* were dropped for the sake of brevity (as in *standard enthalpy of formation*).

The thermodynamic variables were named with its extensive names (so use *enthalpy* instead of *h*) and a modifier if needed (e.g. *heat_capacity_constant_v*, *temperature_saturation* etc).

3 Databases

Input is expected to be in SI units. Output is SI unless specified. Not all modules support input as an object from Units module.

3.1 IAPWS

Units of measurement in this module can be given either in the form of a unit (from the module Units) or built-in float.

Results are returned on a massic basis using kJ as energy unit. So enthalpy is in [kJ/kg], entropy in [kJ/(kg K)] etc.

Object: *Water(p, T)*
Methods:

temperature_saturation(p): Gives the saturation temperature at a given pressure.

pressure_saturation(T): Gives the saturation pressure at a given temperature.

gibbs_energy(p, T)

specific_volume(p, T)

internal_energy(p, T)

entropy(p, T)

enthalpy(p, T)

heat_capacity(p, T)

heat_capacity_constant_v(p, T)

speed_of_sound(p, T)

Atributes:

pc: critical pressure. 22.064 MPa.

rhoc: Critical point density. 322 kg/m3.

Tt: Triple point temperature. 273.16 K.

pt: Triple point pressure. 611.657 Pa.

hc: Enthalpy at triple point. 0.611783 J/kg.

3.2 Nasa 9 term polynomials

Input in this module must be given as a built-in float.

Results are given in SI units on a molar basis (e.g. J/mol).

Base Object: *Database()*
Explanation: This class initializes the database that contains the coefficients and other data. It should be used to search and set compounds. Compound names can be searched within the NASA 9 polynomials database with any of: name in the native .inp file, InChIKey , IUPAC name, or CAS number. Note that the only field certainly available for a compound is the native .inp name.

It is very important to notice that compounds are in the gaseous state unless otherwise specified. So looking for H_2O is equivalent to look for $H_2O_{(g)}$. If you want to look for ice look for 'H2O(cr)' or 'H2O(s)' or 'H2O(l)' for liquid water.
Methods:

list_compound(string): Lists the compounds that match the input string. Mainly used to check the availability of such a compound. String should be in any of the formats: cas number, InChIKey or usual name.

set_compound(string): Returns a Compound() object. The input string in this case should not generate more than one result in *list_compound(string)*.

Object: *Compound(Element)*
Explanation: This object should not be instantiated arbitrarily. Database() offers better ways to do that.
Methods:

heat_capacity(T)

enthalpy(T): $H_{(T)} = H_f^{298.15} + \int_{298.15}^{T} C_p dT$.

entropy(T)

gibbs_energy(T)

Atributes:

inp_name: name as given in the .inp file from the NASA website.

inchikey: InChIKey identification.

canonical_smiles: canonical smiles string.

cas_number

iupac_name

comment: comment if present in the .inp file.

reference: academic reference. See section below for details.

elements: returns a list of tuples of the type: ('ELEMENT', 'number_of_atoms').

condensed: a bool (boolean) equal to True if condensed, False otherwise.

molecular_weight: molecular weight in SI [kg/mol].

enthalpy_of_formation: enthalpy of formation at 298.15 K.

Object: *Compound_ideal_gas(Compound)*
Explanation: Subclass of Compounds which have an extended set of methods which are valid for ideal gas only.
Methods:

heat_capacity_constant_v: It follows from the relation $C_v = C_p - R$.

internal_energy: It follows from $H = U + PV$ and $PV = RT$.

Object: *Reaction(T, reactants, products, reactants_coefficients, product_coefficients)*
Explanation: Models a reaction in the context of the nasa9polynomials. *Reactants* and *Products* should be a list of Compounds in the reaction. In the same order should follow their list of coefficients. The program handles the signs of the coefficients (i.e. they can be given as positive).
Methods:

enthalpy_difference(T)

entropy_difference(T)

gibbs_energy_difference(T)

equilibrium_constant(T): $K_{eq} = exp(\frac{-\Delta G}{R*T})$.

3.3 Burcat

Input in this module must be given as float. Note that this database is inherited from Thermopy v0.4.0 and as such is not standardized or as feature rich as the NASA 9 polynomials.

Results are given in SI units with a molar basis (e.g. J/mol).

Base Object: *Database()*
Explanation: The same as the previous Database() but for the Burcat's database. The methods are the same with slightly less functionality.
Methods:

list_compound(string): Lists the compounds that match the input string. Mainly used to check the availability of such a compound. String should be in the form of the usual name.

set_compound(string): Returns a Compound() object. The implementation of the database search is not as efficient as is in the NASA 9 polynomials. The fields considered are only the names contained in 'burcat_thr.xml' and the match should be exactly the same.

Object: *Compound(Element)*
Explanation: This object should not be instantiated arbitrarily. Database() offers better ways to do that.
Methods:

heat_capacity(T)

enthalpy(T): $H_{(T)} = \int_{298.15}^{T} C_p dT$
entropy(T)
gibbs_energy(T)
density_ideal_gas(p, T)
heat_capacity_massic(T)
enthalpy_massic(T)
enthalpy_engineering(T): Computes $H_{(T)} = H_f^{298.15} + \int_{298.15}^{T} C_p dT$.

Atributes:

cas: CAS name.
description: sometimes the .inp provides a description for the compound.
formula: returns the formula of the type CHON etc.
aggr_state: returns 'G', 'L' or 'S' for gas, liquid or solid.
mm: molar mass [kg/mol].
h_formation: enthalpy of formation.

Object: *Reaction(T, reactants, products, reactants_coefficients, product_coefficients)*
Explanation: Models a reaction in the context of the Burcat's databse. *Reactants* and *Products* should be a list of Compounds in the reaction. In the same order should follow their list of coefficients. The program handles the signs of the coefficients (i.e. they can be given as positive).
Methods:

equilibrium_constant(T): $K_{eq} = exp(\frac{-\Delta G}{R*T})$

Note: whenever possible use the NASA 9 database.

4 Units

Module for unit conversion.

4.1 Temperature

Units available: K, °C, °F.

4.2 Pressure

Units available: Pa, MPa, bar, psi, atm, mmwc, torr.

4.3 Enthalpy

Units available: Jkg, kJkg, kcalkg, Btulb.

4.4 Length

Units available: m, mm, inch, ft.

4.5 Massflow

Units available: kgs, kgh, lbs, lbh.

4.6 Energy

Units available: J, Btu, cal, kWh.

5 Constants

300+ Constants. The form of the constant is:
$constant = (numericvalue, units, error)$. (tuple)
Example:

```
ideal_gas_constant = (8.314472, 'J mol^-1 K^-1', 1.5e-05)
```

6 Usage and Examples

6.1 IAPWS

EXAMPLE 01: some properties

```
>>> from thermopy import iapws
>>> water1 = iapws.Water(1e5, 273.15)
>>> print(water1.gibbs_energy())
1.80213440703
>>> print(water1.pressure_saturation())
611.212677444345
>>> print(water1.pressure_saturation(373.15))
101417.97792131013
```

6.2 Nasa 9 term polynomials

EXAMPLE 01: Looking for silicon and its aggregation states: a careless user looking for properties of silicon $Si_{(s)}$ would naively look for the InChIKey 'XUIMIQQOPSSXEZ-UHFFFAOYSA-N' or just 'Si'. In fact, see what happens:

```
>>> from thermopy import nasa9polynomials as nasa9
>>> db = nasa9.Database()
>>> print(db.list_compound('XUIMIQQOPSSXEZ-UHFFFAOYSA-N'))
[('Si', 'silicon'), ('Si+', 'silicon'), ('Si-', 'silicon'), ('Si(cr)', 'silicon'), ('Si(L)', 'silicon')]
>>> solid_silicon = db.set_compound('si(cr)')
>>> liquid_silicon = db.set_compound('si(l)')
>>> gaseous_silicon = db.set_compound('si')
>>> print(solid_silicon.condensed, liquid_silicon.condensed,
... gaseous_silicon.condensed)
True True False
>>> print(solid_silicon.heat_capacity(1690), liquid_silicon.heat_capacity(1690),
... gaseous_silicon.heat_capacity(1690), sep='____')
29.1310172482____27.1998762692____21.5046220109
```

EXAMPLE 02: Hydrazine 'messing around' example.

```
>>> from thermopy import nasa9polynomials as nasa9
>>> db = nasa9.Database()
>>> db.list_compound('hydrazine')
[('N2H4', 'hydrazine')]
>>> n2h4 = db.set_compound('hydrazine')
>>> n2h4.enthalpy_of_formation
95180.0
>>> n2h4.molecular_weight
0.03204516
>>> n2h4.enthalpy(373.15) - n2h4.enthalpy(273.15)
5074.9114830065955
```

EXAMPLE 03: Consider a 1 kg aluminum object initially at 800 K being put in a 298.15 K xenon atmosphere (1000 mol). After the heat exchange occurs what is the temperature of the medium?

```
>>> from thermopy import nasa9polynomials as nasa9
>>> from scipy import optimize
>>> db = nasa9.Database()
>>> aluminum = db.set_compound('Al(cr)')
>>> xenon = db.set_compound('Xe')
>>> T0_Al = 800
>>> T0_Xe = 298.15
>>> xe_moles = 1000
>>> al_moles = 1 / aluminum.molecular_weight
>>> print('Elements:', aluminum, xenon)
Elements: aluminum: Al(cr) xenon: Xe
>>> print('Is the aluminum condensed?', aluminum.condensed)
Is the aluminum condensed? True
>>>
>>>
>>> def obj(x):
... return [al_moles * (aluminum.enthalpy(x[0]) - aluminum.enthalpy(T0_Al)) +
... xe_moles * (xenon.enthalpy(x[0]) - aluminum.enthalpy(T0_Xe))]
...
>>> initial_guess = 500
>>> solution = optimize.root(obj, initial_guess,
... options={'ftol': 1e-5, 'eps': 0.001},
... method='lm')
>>> print(solution)
     ipvt: array([1], dtype=int32)
        x: array([ 321.74543335])
     fjac: array([[-21701.4534997]])
    cov_x: array([[ 2.12335376e-09]])
      qtf: array([ 0.11274949])
      fun: array([-0.11274949])
  status: 2
 message: 'The relative error between two consecutive iterates is at most 0.000000'
 success: True
     nfev: 11
```

EXAMPLE 04: Consider the equilibrium reaction: $2HI \rightleftharpoons H_2 + I_2$. Find its heat of reaction and equilibrium constant at 300, 500, 900 K.

```
>>> from thermopy import nasa9polynomials as nasa9
>>> db = nasa9.Database()
>>> hydrogen = db.set_compound('H2')
>>> iodine = db.set_compound('I2')
>>> hydrogen_iodide = db.set_compound('HI')
>>> reaction1 = nasa9.Reaction(298.15, [hydrogen_iodide], [hydrogen, iodine],
... [2], [1, 1])
>>> for T in (300, 500, 900):
... print('heat of reaction', reaction1.enthalpy_difference(T), '\n',
```

```
... 'equilibrium constant', reaction1.equilibrium_constant(T))
...
heat of reaction -52718.9284088
 equilibrium constant 17.7261341155
heat of reaction -52808.175834
 equilibrium constant 0.00375742077156
heat of reaction -54084.1965933
 equilibrium constant 1.27580495983e-05
```

6.3 Burcat

As the NASA 7 term polynomials are less accurate than their 9 term counterpart, support and development for them is halted. Nevertheless the basic usage is:

```
>>> from thermopy import burcat
>>> db = burcat.Database()
>>> no = db.set_compound('no')
>>> print(no.cas, no.heat_capacity(550), no.description, sep='|')
10102-43-9|30.8488271483|NO GENERATED FROM ORIGINAL VALUES HFO=82.09 KJ
```

6.4 Units

Example:

```
>>> from thermopy import units
>>> temp = units.Enthalpy(10)
>>> print(temp.Btulb)
0.004299226139294927
>>> pressure = units.Pressure(10).unit('atm')
>>> print(pressure)
1013250.0
```

6.5 Constants

Since this module is not used directly by the end user and some constants names usually have more than one common name (e.g. ideal gas constant, universal gas constant, etc) no examples are provided here. For reliable use open the python file.

7 Sources for *reference* attribute

The reference contents of one of the .inp files is reproduced here for convenience.

```
!
!              SIX-CHARACTER REFERENCE-DATE CODES
!
!
!  Letters         Reference                          Numbers
!  ------------------------------------------------------------------
!
!    g      Glenn Research Center              Month/year calculated
!
!    j      NIST-JANAF Thermochemical          Month/year of table
!           Tables. Chase,1998
!
!    tpis   Thermodynamic Properties of        Year of volume
!           Individual Substances. Gurvich,
!           1978,1979,1982,1989,1991,1996
!
!    n      TRC Thermodynamic Tables, NIST     Month/year of table
!
!    bar    Barin:Thermochemical Data of       Year of volume
!           Pure Substances. Barin,1989
!
!    coda   CODATA Key Values for              Year of volume
!           Thermodynamics. Cox,1989
!
!    srd    Standard Reference Data            Year of J.Phys.Chem.Ref.
!                                              Data journal
```

References

[1] Bonnie J. McBride, Michael J. Zehe, and Sanford Gordon. NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species. September 2002.