Python implementation of standard from IAPWS (http://www.iapws.org/release.html).

- **Home:** https://github.com/jjgomera/iapws
- **Author:** Juan José Gómez Romera <jjgomera@gmail.com>
- **License:** GPL-3
- **Documentation:** http://iapws.readthedocs.io/
CHAPTER 2

Dependences

• python 2x, 3x, compatible with both versions
• Numpy-scipy: library with mathematic and scientific tools
In Debian you can find it in official repositories in Jessie, Testing and Sid. In Ubuntu it’s in official repositories from Ubuntu Saucy (13.10). In other systems you can install using pip:

```
pip install iapws
```

or directly cloning the Github repository:

```
git clone https://github.com/jjgomera/iapws.git
```

and adding the folder to a Python path.
CHAPTER 4

Features

This module implements almost the full set of standards:

Releases:

- R1-76(2014): Revised Release on the Surface Tension of Ordinary Water Substance, \texttt{iapws\_iapws\_Tension()}
- R2-83(1992): Release on the Values of Temperature, Pressure and Density of Ordinary and Heavy Water Substances at their Respective Critical Points, \texttt{iapws\_iapws()}
- R4-84(2007): Revised Release on Viscosity and Thermal Conductivity of Heavy Water Substance, \texttt{iapws\_iapws\_D2O\_Viscosity(),iapws\_iapws\_D2O\_ThCond()}
- R5-85(1994): Release on Surface Tension of Heavy Water Substance, \texttt{iapws\_iapws\_D2O\_Tension()}
- R8-97: Release on the Static Dielectric Constant of Ordinary Water Substance for Temperatures from 238 K to 873 K and Pressures up to 1000 MPa, \texttt{iapws\_iapws\_Dielectric()}
- R9-97: Release on the Refractive Index of Ordinary Water Substance as a Function of Wavelength, Temperature and Pressure, \texttt{iapws\_iapws\_Refractive()}
- R11-07: Release on the Ionization Constant of H2O, \texttt{iapws\_iapws\_Kw()}
- R12-08: Release on the IAPWS Formulation 2008 for the Viscosity of Ordinary Water Substance, \texttt{iapws\_iapws\_Viscosity()}
- R13-08: Release on the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, \texttt{iapws\_iapws08()}
• R14-08(2011): Revised Release on the Pressure along the Melting and Sublimation Curves of Ordinary Water Substance, \texttt{iapws\_iapws\_Melting\_Pressure()}, \texttt{iapws\_iapws\_Sublimation\_Pressure()}

• R15-11: Release on the IAPWS Formulation 2011 for the Thermal Conductivity of Ordinary Water Substance, \texttt{iapws\_iapws\_ThCond()}

• R16-17: Release on the IAPWS Formulation 2017 for the Thermodynamic Properties of Heavy Water, \texttt{iapws\_iapws95.D2O()}

Supplementary Releases:

• SR1-86(1992): Revised Supplementary Release on Saturation Properties of Ordinary Water Substance, \texttt{iapws\_iapws95.MEoS\_Liquid\_Density()}, \texttt{iapws\_iapws95.MEoS\_Vapor\_Density()}, \texttt{iapws\_iapws95.MEoS\_Vapor\_Pressure()}

• SR2-01(2014): Revised Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy \(p(h,s)\) for Regions 1 and 2 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward1\_P_hs()}, \texttt{iapws\_iapws97\_Backward2\_P_hs()}

• SR3-03(2014): Revised Supplementary Release on Backward Equations for the Functions \(T(p,h), v(p,h)\), and \(T(p,s), v(p,s)\) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward3\_T_Ph()}, \texttt{iapws\_iapws97\_Backward3\_T_Ps()}, \texttt{iapws\_iapws97\_Backward3\_v_Ph()}, \texttt{iapws\_iapws97\_Backward3\_v_Ps()}

• SR4-04(2014): Revised Supplementary Release on Backward Equations \(p(h,s)\) for Region 3, Equations as a Function of \(h\) and \(s\) for the Region Boundaries, and an Equation \(T_{sat}(h,s)\) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward3\_P_hs()}

• SR5-05(2016): Revised Supplementary Release on Backward Equations for Specific Volume as a Function of Pressure and Temperature \(v(p,T)\) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward3\_v_PT()}

• SR6-08(2011): Revised Supplementary Release on Properties of Liquid Water at 0.1 MPa, \texttt{iapws\_iapws\_Liquid()}

• SR7-09: Supplementary Release on a Computationally Efficient Thermodynamic Formulation for Liquid Water for Oceanographic Use, \texttt{iapws\_iapws08.SeaWater\_waterSupp()}

Guidelines:

• G1-90: Electrolytic Conductivity (Specific Conductance) of Liquid and Dense Supercritical Water from 0°C to 800°C and Pressures up to 1000 MPa, \texttt{iapws\_iapws\_Conductivity()}

• G2-90(1994): Solubility of Sodium Sulfate in Aqueous Mixtures of Sodium Chloride and Sulfuric Acid from Water to Concentrated Solutions, from 250°C to 350°C, \texttt{iapws\_iapws08\_solNa2SO4()}

• G3-00(2012): Revised Guideline on the Critical Locus of Aqueous Solutions of Sodium Chloride, \texttt{iapws\_iapws08\_critNaCl()}

• G4-01: Guideline on the IAPWS Formulation 2001 for the Thermodynamic Properties of Ammonia-Water Mixtures, \texttt{iapws\_ammonia()}

• G5-01(2016): Guideline on the Use of Fundamental Physical Constants and Basic Constants of Water, \texttt{iapws\_iapws()}

• G6-03: Guideline on the Tabular Taylor Series Expansion (TTSE) Method for Calculation of Thermodynamic Properties of Water and Steam Applied to IAPWS-95 as an Example (Not implemented)

• G7-04: Guideline on the Henry’s Constant and Vapor-Liquid Distribution Constant for Gases in H2O and D2O at High Temperatures, \texttt{iapws\_iapws\_Henry()}, \texttt{iapws\_iapws\_Kvalue()}
• G8-10: Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, \texttt{iapws.humidAir.HumidAir()}

• G9-12: Guideline on a Low-Temperature Extension of the IAPWS-95 Formulation for Water Vapor, \texttt{iapws.iapws95.IAPWS95._phiex()}

• G10-15: Guideline on the Thermal Conductivity of Seawater, \texttt{iapws.iapws08._ThCond_SeaWater()}

• G11-15: Guideline on a Virial Equation for the Fugacity of H2O in Humid Air, \texttt{iapws.humidAir._virial()}

• G12-15: Guideline on Thermodynamic Properties of Supercooled Water, \texttt{iapws._iapws._Supercooled()}

• G13-15: Guideline on the Fast Calculation of Steam and Water Properties with the Spline-Based Table Look-Up Method (SBTL) (Not implemented)

Advisory Notes:

• AN1-03: Uncertainties in Enthalpy for the IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use (IAPWS-95) and the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam (IAPWS-IF97)

• AN2-04(2013): Role of Various IAPWS Documents Concerning the Thermodynamic Properties of Ordinary Water Substance

• AN3-07(2014): Thermodynamic Derivatives from IAPWS Formulations, \texttt{iapws._utils.deriv_G()}, \texttt{iapws._utils.deriv_H()}

• AN4-09: Roles of IAPWS and CIPM Standards for the Density of Water

• AN5-13(2016): Industrial Calculation of the Thermodynamic Properties of Seawater, \texttt{iapws.iapws08.Seawater._waterIF97()}, \texttt{iapws.iapws08._Tb()}, \texttt{iapws.iapws08._Tf()}, \texttt{iapws.iapws08._Triple()}, \texttt{iapws.iapws08._OsmoticPressure()}

• AN6-16: Relationship between Various IAPWS Documents and the International Thermodynamic Equation of Seawater - 2010 (TEOS-10)
You can navigate the full documentation of package:

## 5.1 iapws package

### 5.1.1 iapws._iapws module

Miscellaneous IAPWS standards. This module include:

- **_Ice()**: Ice Ih state equation
- **_Liquid()**: Properties of liquid water at 0.1 MPa
- **_Supercooled()**: Thermodynamic properties of supercooled water
- **_Sublimation_Pressure()**: Sublimation pressure correlation
- **_Melting_Pressure()**: Melting pressure correlation
- **_Viscosity()**: Viscosity correlation
- **_ThCond()**: Thermal conductivity correlation
- **_Tension()**: Surface tension correlation
- **_Dielectric()**: Dielectric constant correlation
- **_Refractive()**: Refractive index correlation
- **_Kw()**: Ionization constant correlation for ordinary water
- **_Conductivity()**: Electrolytic conductivity correlation
- **_D2O_Viscosity()**: Viscosity correlation for heavy water
- **_D2O_ThCond()**: Thermal conductivity correlation for heavy water
- **_D2O_Tension()**: Surface tension correlation for heavy water
• **_D2O_Sublimation_Pressure()**: Sublimation Pressure correlation for heavy water  
• **_D2O_Melting_Pressure()**: Melting Pressure correlation for heavy water  
• **_Henry()**: Henry constant for liquid-gas equilibrium  
• **_Kvalue()**: Vapor-liquid distribution constant

```python
iapws._iapws._Ice(T, P)
```

Basic state equation for Ice Ih

**Parameters**

- `T (float)`: Temperature, [K]  
- `P (float)`: Pressure, [MPa]

**Returns**

- `prop` – Dict with calculated properties of ice. The available properties are:
  - `rho`: Density, [kg/m³]  
  - `h`: Specific enthalpy, [kJ/kg]  
  - `u`: Specific internal energy, [kJ/kg]  
  - `a`: Specific Helmholtz energy, [kJ/kg]  
  - `g`: Specific Gibbs energy, [kJ/kg]  
  - `s`: Specific entropy, [kJ/kgK]  
  - `cp`: Specific isobaric heat capacity, [kJ/kgK]  
  - `alfav`: Cubic expansion coefficient, [1/K]  
  - `beta`: Pressure coefficient, [MPa/K]  
  - `xkappa`: Isothermal compressibility, [1/MPa]  
  - `ks`: Isentropic compressibility, [1/MPa]  
  - `gt`: [g/T]P  
  - `gtt`: [g²/T²]P  
  - `gp`: [g/P]T  
  - `gpp`: [g²/P²]T  
  - `gtp`: [g²/TP]

**Return type** `dict`

**Notes**

Raise `NotImplementedError` if input isn’t in limit:

- `T` 273.16  
- `P` 208.566  
- State below the melting and sublimation lines
Examples

```python
>>> st1 = _Ice(100, 100)
>>> st1["rho"], st1["h"], st1["s"]
941.678203297 -483.491635676 -2.61195122589

>>> st2 = _Ice(273.152519, 0.101325)
>>> st2["a"], st2["u"], st2["cp"]
-0.00918701567 -333.465403393 2.09671391024

>>> st3 = _Ice(273.16, 611.657e-6)
>>> st3["alfav"], st3["beta"], st3["xkappa"], st3["ks"]
0.000159863102566 1.35714764659 1.17793449348e-04 1.14161597779e-04
```

References


iapws._iapws._Liquid(T, P=0.1)

Supplementary release on properties of liquid water at 0.1 MPa

**Parameters**

- **T** (*float*) – Temperature, [K]
- **P** (*float*) – Pressure, [MPa] Although this relation is for P=0.1MPa, can be extrapolated at pressure 0.3 MPa

**Returns**

**prop** –

Dict with calculated properties of water. The available properties are:

- **h**: Specific enthalpy, [kJ/kg]
- **u**: Specific internal energy, [kJ/kg]
- **a**: Specific Helmholtz energy, [kJ/kg]
- **g**: Specific Gibbs energy, [kJ/kg]
- **s**: Specific entropy, [kJ/kgK]
- **cp**: Specific isobaric heat capacity, [kJ/kgK]
- **cv**: Specific isochoric heat capacity, [kJ/kgK]
- **w**: Speed of sound, [m/s²]
- **rho**: Density, [kg/m³]
- **v**: Specific volume, [m³/kg]
- **vt**: [v/T]P, [m³/kgK]
- **vtt**: [v²/T²]P, [m³/kgK²]
- **vp**: [v/P]T, [m³/kg/MPa]
- **vtp**: [v²/TP], [m³/kg/MPa]
- **alfav**: Cubic expansion coefficient, [1/K]
• \( \kappa \): Isothermal compressibility, [1/MPa]
• \( s \): Isentropic compressibility, [1/MPa]
• \( \mu \): Viscosity, [mPas]
• \( k \): Thermal conductivity, [W/mK]
• \( \epsilon \): Dielectric constant, [-]

**Return type** dict

**Notes**

Raise `NotImplementedError` if input isn’t in limit:

- \( 253.15 \leq T \leq 383.15 \)
- \( 0.1 \leq P \leq 0.3 \)

**Examples**

```python
>>> st1 = _Liquid(260)
>>> st1["rho"], st1["h"], st1["s"]
997.0683602710492 -55.86223174460868 -0.20998554842619535
```

**References**

IAPWS, Revised Supplementary Release on Properties of Liquid Water at 0.1 MPa, [http://www.iapws.org/relguide/LiquidWater.html](http://www.iapws.org/relguide/LiquidWater.html)

**iapws._iapws._Supercooled\((T, P)\)**

Guideline on thermodynamic properties of supercooled water

**Parameters**

- \( T \) (float) – Temperature, [K]
- \( P \) (float) – Pressure, [MPa]

**Returns**

`prop` –

Dict with calculated properties of water. The available properties are:

- \( \rho \): Density, [kg/m³]
- \( x \): Mole fraction of low-density structure, [-]
- \( s \): Specific entropy, [kJ/kgK]
- \( h \): Specific enthalpy, [kJ/kg]
- \( u \): Specific internal energy, [kJ/kg]
- \( a \): Specific Helmholtz energy, [kJ/kg]
- \( g \): Specific Gibbs energy, [kJ/kg]
- \( \alpha \): Thermal expansion coefficient, [1/K]
• \( \kappa \) : Isothermal compressibility, [1/MPa]
• \( c_p \) : Specific isobaric heat capacity, [kJ/kgK]
• \( c_v \) : Specific isochoric heat capacity, [kJ/kgK]
• \( w \) : Speed of sound, [m/s]

Return type \( \text{dict} \)

Notes

Raise \text{NotImplementedError} if input isn’t in limit:

• \( T_m \leq T \leq 300 \)
• \( 0 < P < 1000 \)

The minimum temperature in range of validity is the melting temperature, it depend of pressure

Examples

```python
>>> liq = _Supercooled(235.15, 0.101325)
>>> liq["rho"], liq["cp"], liq["w"]
968.09999 5.997563 1134.5855
```

References

IAPWS, Guideline on Thermodynamic Properties of Supercooled Water, \url{http://iapws.org/relguide/Supercooled.html}

iapws._iapws._Sublimation_Pressure\( (T) \)

Sublimation Pressure correlation

Parameters \( T (\text{float}) \) – Temperature, [K]

Returns \( P \) – Pressure at sublimation line, [MPa]

Return type \( \text{float} \)

Notes

Raise \text{NotImplementedError} if input isn’t in limit:

• \( 50 \leq T \leq 273.16 \)

Examples

```python
>>> _Sublimation_Pressure(230)
8.947352740189152e-06
```
References


iapws._iapws._Melting_Pressure(T, ice='Ih')

Melting Pressure correlation

Parameters

• T (float) – Temperature, [K]
• ice (string) – Type of ice: Ih, III, V, VI, VII. Below 273.15 is a mandatory input, the ice Ih is the default value. Above 273.15, the ice type is unnecessary.

Returns

P – Pressure at sublimation line, [MPa]

Return type float

Notes

Raise NotImplementedError if input isn’t in limit:

• 251.165 T 715

Examples

```python
>>> _Melting_Pressure(260)
8.947352740189152e-06
>>> _Melting_Pressure(254, "III")
268.6846466336108
```

References


iapws._iapws._Viscosity(rho, T, fase=None, drho=None)

Equation for the Viscosity

Parameters

• rho (float) – Density, [kg/m³]
• T (float) – Temperature, [K]
• fase (dict, optional for calculate critical enhancement) – phase properties
• drho (float, optional for calculate critical enhancement) – [ρ/P]T at reference state,

Returns

µ – Viscosity, [Pa·s]

Return type float
Examples

```python
>>> _Viscosity(998, 298.15)
0.0008897351001498108
>>> _Viscosity(600, 873.15)
7.74301952728247e-05
```

References


```python
iapws._iapws._ThCond(rho, T, fase=None, drho=None)
```

Equation for the thermal conductivity

**Parameters**

- `rho` *(float)* – Density, \([\text{kg/m}^3]\)
- `T` *(float)* – Temperature, \([\text{K}]\)
- `fase` *(dict, optional for calculate critical enhancement)* – phase properties
- `drho` *(float, optional for calculate critical enhancement)* – \([\rho/P]T\) at reference state,

**Returns**

- `k` – Thermal conductivity, \([\text{W/mK}]\)

**Return type**

*float*

Examples

```python
>>> _ThCond(998, 298.15)
0.6077128675880629
>>> _ThCond(0, 873.15)
0.07910346589648833
```

References


```python
iapws._iapws._Tension(T)
```

Equation for the surface tension

**Parameters**

- `T` *(float)* – Temperature, \([\text{K}]\)

**Returns**

- `\sigma` – Surface tension, \([\text{N/m}]\)

**Return type**

*float*

**Notes**

Raise `NotImplementedError` if input isn’t in limit:

- \(248.15 \leq T \leq 647\)
• Extrapolate to -25°C in supercooled liquid metastable state

Examples

```python
>>> _Tension(300)
0.0716859625
>>> _Tension(450)
0.0428914992
```

References


iapws._iapws._Dielectric(rho, T)

Equation for the Dielectric constant

Parameters

- `rho (float)` – Density, [kg/m³]
- `T (float)` – Temperature, [K]

Returns `epsilon` – Dielectric constant, [-]

Return type `float`

Notes

Raise `NotImplementedError` if input isn’t in limit:

- 238 T 1200

Examples

```python
>>> _Dielectric(999.242866, 298.15)
78.5907250
>>> _Dielectric(26.0569558, 873.15)
1.12620970
```

References


iapws._iapws._Refractive(rho, T, l=0.5893)

Equation for the refractive index

Parameters

- `rho (float)` – Density, [kg/m³]
- `T (float)` – Temperature, [K]
- `l (float, optional)` – Light Wavelength, [μm]
Returns \( n \) – Refractive index, [-]

Return type  float

Notes

Raise \texttt{NotImplementedError} if input isn’t in limit:

- \( 0 \leq \rho \leq 1060 \)
- \( 261.15 \leq T \leq 773.15 \)
- \( 0.2 \leq \lambda \leq 1.1 \)

Examples

```python
>>> _Refractive(997.047435, 298.15, 0.2265)
1.39277824
>>> _Refractive(30.4758534, 773.15, 0.5893)
1.00949307
```

References


iapws._iapws._Kw(rho, T)

Equation for the ionization constant of ordinary water

Parameters

- \( \rho \) (float) – Density, [kg/m\(^3\)]
- \( T \) (float) – Temperature, [K]

Returns \( pKw \) – Ionization constant in -log10(kw), [-]

Return type  float

Notes

Raise \texttt{NotImplementedError} if input isn’t in limit:

- \( 0 \leq \rho \leq 1250 \)
- \( 273.15 \leq T \leq 1073.15 \)

Examples

```python
>>> _Kw(1000, 300)
13.906565
```
References


iapws._iapws._Conductivity(rho, T)

Equation for the electrolytic conductivity of liquid and dense supercritical water

Parameters

• \( \rho \) (float) – Density, [kg/m\(^3\)]
• \( T \) (float) – Temperature, [K]

Returns \( K \) – Electrolytic conductivity, [S/m]

Return type float

Notes

Raise \texttt{NotImplementedError} if input isn’t in limit:

• \( 600 \leq \rho \leq 1200 \)
• \( 273.15 \leq T \leq 1073.15 \)

Examples

```python
>>> _Conductivity(1000, 373.15)
1.13
```

References

IAPWS, Electrolytic Conductivity (Specific Conductance) of Liquid and Dense Supercritical Water from 0°C to 800°C and Pressures up to 1000 MPa, http://www.iapws.org/relguide/conduct.pdf

iapws._iapws._D2O_Viscosity(rho, T)

Equation for the Viscosity of heavy water

Parameters

• \( \rho \) (float) – Density, [kg/m\(^3\)]
• \( T \) (float) – Temperature, [K]

Returns \( \mu \) – Viscosity, [Pa·s]

Return type float

Examples

```python
>>> _D2O_Viscosity(998, 298.15)
0.0008897351001498108
>>> _D2O_Viscosity(600, 873.15)
7.74301952278247e-05
```
References


iapws._iapws._D2O_ThCond(rho, T)
Equation for the thermal conductivity of heavy water

Parameters

• rho (float) – Density, [kg/m³]
• T (float) – Temperature, [K]

Returns k – Thermal conductivity, [W/mK]
Return type float

Examples

```python
>>> _D2O_ThCond(998, 298.15)
0.6077128675880629
>>> _D2O_ThCond(0, 873.15)
0.07910346589648833
```

References


iapws._iapws._D2O_Tension(T)
Equation for the surface tension of heavy water

Parameters T (float) – Temperature, [K]

Returns σ – Surface tension, [N/m]
Return type float

Notes

Raise NotImplementedError if input isn’t in limit:

• 269.65 T 643.847

Examples

```python
>>> _D2O_Tension(298.15)
0.07186
>>> _D2O_Tension(573.15)
0.01399
```
References


iapws._iapws._D2O_Sublimation_Pressure(T)
Sublimation Pressure correlation for heavy water

Parameters
- T (float) – Temperature, [K]

Returns
- P – Pressure at sublimation line, [MPa]

Return type: float

Notes

Raise NotImplementedError if input isn’t in limit:
- • 210 T 276.969

Examples

```python
>>> _Sublimation_Pressure(245)
3.27390934e-5
```

References


iapws._iapws._D2O_Melting_Pressure(T, ice='Ih')
Melting Pressure correlation for heavy water

Parameters
- • T (float) – Temperature, [K]
- • ice (str) – Type of ice: Ih, III, V, VI, VII. Below 276.969 is a mandatory input, the ice Ih is the default value. Above 276.969, the ice type is unnecessary.

Returns
- P – Pressure at melting line, [MPa]

Return type: float

Notes

Raise NotImplementedError if input isn’t in limit:
- • 254.415 T 315

Examples

```python
>>> _D2O_Melting_Pressure(260)
8.947352740189152e-06
>>> _D2O_Melting_Pressure(254, "III")
268.6846466336108
```
References


iapws._iapws._Henry(T, gas, liquid='H2O')
Equation for the calculation of Henry’s constant

Parameters

• \( T (\text{float}) \) – Temperature, [K]
• \( \text{gas} (\text{string}) \) – Name of gas to calculate solubility
• \( \text{liquid} (\text{string}) \) – Name of liquid solvent, can be H2O (default) or D2O

Returns \( \text{kw} \) – Henry’s constant, [MPa]

Return type \( \text{float} \)

Notes

The gas availables for H2O solvent are He, Ne, Ar, Kr, Xe, H2, N2, O2, CO, CO2, H2S, CH4, C2H6, SF6 For D2O as solvent He, Ne, Ar, Kr, Xe, D2, CH4

Raise \text{NotImplementedError} \text{if input gas or liquid are unsupported}

Examples

```python
>>> _Henry(500, "He")
1.1973
>>> _Henry(300, "D2", "D2O")
1.6594
```

References


iapws._iapws._Kvalue(T, gas, liquid='H2O')
Equation for the vapor-liquid distribution constant

Parameters

• \( T (\text{float}) \) – Temperature, [K]
• \( \text{gas} (\text{string}) \) – Name of gas to calculate solubility
• \( \text{liquid} (\text{string}) \) – Name of liquid solvent, can be H2O (default) or D2O

Returns \( \text{kd} \) – Vapor-liquid distribution constant, [-]

Return type \( \text{float} \)
Notes

The gas availables for H2O solvent are He, Ne, Ar, Kr, Xe, H2, N2, O2, CO, CO2, H2S, CH4, C2H6, SF6
For D2O as solvent He, Ne, Ar, Kr, Xe, D2, CH4
Raise NotIMplementedError if input gas or liquid are unsupported

Examples

```python
>>> _Kvalue(600, "He")
3.8019
>>> _Kvalue(300, "D2", "D2O")
14.3520
```

References

IAPWS, Guideline on the Henry’s Constant and Vapor-Liquid Distribution Constant for Gases in H2O and D2O

5.1.2 iapws._utils module

Miscellaneous internal utilities. This module include:

- `getphase()`: Get phase string of state
- `_fase`: Base class to define a phase state
- `deriv_H()`: Calculate generic partial derivative with a fundamental Helmholtz free energy equation of state
- `deriv_G()`: Calculate generic partial derivative with a fundamental Gibbs free energy equation of state

```python
iapws._utils.getphase(Tc, Pc, T, P, x, region)
```

Return fluid phase string name

Parameters

- `Tc (float)` – Critical temperature, [K]
- `Pc (float)` – Critical pressure, [MPa]
- `T (float)` – Temperature, [K]
- `P (float)` – Pressure, [MPa]
- `x (float)` – Quality, [-]
- `region (int)` – Region number, used only for IAPWS97 region definition

Returns `phase` – Phase name

Return type `str`

```python
class iapws._utils._fase
```

Class to implement a null phase

Attributes

- `Gruneisen`
- `IntP`
Ks
Kt
Prandt
Z
Z_rho
a
alfa
alfap
alfav
betap
betas
cp
cp_cv
cv
dhdP_T
dhdP_rho
dhdT_P
dhdT_rho
dhdrho_P
dhdrho_T
dpdT_rhoe
dpdrho_T
drhodP_T
drhodT_P
epsilon
f
fi
g
gamma
h
hInput
joule
k
kappa
ks
kt
iapws._utils.deriv_H(state, z, x, y, fase)
Calculate generic partial derivative $\frac{\partial z}{\partial x}|_y$ from a fundamental helmholtz free energy equation of state

**Parameters**

- **state** (*any python object*) – Only need to define P and T properties, non phase specific properties
- **z** (*str*) – Name of variables in numerator term of derivatives
- **x** (*str*) – Name of variables in denominator term of derivatives
- **y** (*str*) – Name of constant variable in partial derivative
- **fase** (*any python object*) – Define phase specific properties (v, cv, alfap, s, betap)

**Notes**

x, y and z can be the following values:

- P: Pressure
- T: Temperature
- v: Specific volume
- rho: Density
- u: Internal Energy
- h: Enthalpy
- s: Entropy
- g: Gibbs free energy
- a: Helmholtz free energy

**Returns** deriv – z/x/y
**Return type** float

**References**

IAPWS, Revised Advisory Note No. 3: Thermodynamic Derivatives from IAPWS Formulations, [http://www.iapws.org/relguide/Advise3.pdf](http://www.iapws.org/relguide/Advise3.pdf)

iapws._utils.deriv_G(state, z, x, y, fase)
Calculate generic partial derivative $\frac{\partial z}{\partial x}|_y$ from a fundamental Gibbs free energy equation of state
Parameters

- **state** (any python object) – Only need to define P and T properties, non phase specific properties
- **z** (str) – Name of variables in numerator term of derivatives
- **x** (str) – Name of variables in denominator term of derivatives
- **y** (str) – Name of constant variable in partial derivative
- **fase** (any python object) – Define phase specific properties (v, cp, alfav, s, xkappa)

Notes

x, y and z can be the following values:

- P: Pressure
- T: Temperature
- v: Specific volume
- rho: Density
- u: Internal Energy
- h: Enthalpy
- s: Entropy
- g: Gibbs free energy
- a: Helmholtz free energy

Returns deriv \( \frac{z}{x} |_{y} \)

Return type float

References


5.1.3 iapws.iapws95 module

Implemented multiparameter equation of state as a Helmholtz free energy:

- **MEoS**: Base class of multiparameter equation of state
- **IAPWS95**: 2016 revision of 1995 formulation for ordinary water
- **D2O**: 2017 formulation for heavy water.

iapws.iapws95.\_phir(\tau, \Delta, \text{coef})

Residual contribution to the adimensional free Helmholtz energy

Parameters

- **tau** (float) – Inverse reduced temperature \( Tc/T \), [-]
- **delta** (float) – Reduced density \( \rho/\rho_{oc} \), [-]
**iapws Documentation, Release 1.1.3**

- **coef**(dict) – Dictionary with equation of state parameters

**Returns**
- **fir** – Adimensional free Helmholtz energy

**Return type** float

**References**


**iapws.iapws95._phird**(tau, delta, coef)
Residual contribution to the adimensional free Helmholtz energy, delta derivative

**Parameters**
- **tau**(float) – Inverse reduced temperature Tc/T, [-]
- **delta**(float) – Reduced density rho/rhoc, [-]
- **coef**(dict) – Dictionary with equation of state parameters

**Returns**
- **fird** –

\[
\frac{\partial \phi_\delta}{\partial \delta} 
\]

**Return type** float

**References**


**iapws.iapws95._phirt**(tau, delta, coef)
Residual contribution to the adimensional free Helmholtz energy, tau derivative

**Parameters**
- **tau**(float) – Inverse reduced temperature Tc/T, [-]
- **delta**(float) – Reduced density rho/rhoc, [-]
- **coef**(dict) – Dictionary with equation of state parameters

**Returns**
- **firt** –

\[
\frac{\partial \phi_\tau}{\partial \tau} 
\]

**Return type** float
class iapws.iapws95.MEoS(**kwargs)

General implementation of multiparameter equation of state. From this derived all child class specified per individual compounds

Parameters

- `T (float)` – Temperature, [K]
- `P (float)` – Pressure, [MPa]
- `rho (float)` – Density, [kg/m³]
- `v (float)` – Specific volume, [m³/kg]
- `h (float)` – Specific enthalpy, [kJ/kg]
- `s (float)` – Specific entropy, [kJ/kgK]
- `u (float)` – Specific internal energy, [kJ/kg]
- `x (float)` – Vapor quality, [-]
- `l (float, optional)` – Wavelength of light, for refractive index, [nm]
- `rho0 (float, optional)` – Initial value of density, to improve iteration, [kg/m³]
- `T0 (float, optional)` – Initial value of temperature, to improve iteration, [K]
- `x0` (Initial value of vapor quality, necessary in bad input pair definition) – where there are two valid solution (T-h, T-s)

Notes

- It needs two incoming properties of T, P, rho, h, s, u.
- `v` as a alternate input parameter to rho
- `T-x, P-x`, preferred input pair to specified a point in two phases region

The calculated instance has the following properties:

- `P`: Pressure, [MPa]
- `T`: Temperature, [K]
- `x`: Vapor quality, [-]
- `g`: Specific Gibbs free energy, [kJ/kg]
- `a`: Specific Helmholtz free energy, [kJ/kg]
- `v`: Specific volume, [m³/kg]
- `r`: Density, [kg/m³]
- `h`: Specific enthalpy, [kJ/kg]
- `u`: Specific internal energy, [kJ/kg]
- `s`: Specific entropy, [kJ/kgK]
• cp: Specific isobaric heat capacity, [kJ/kg·K]
• cv: Specific isochoric heat capacity, [kJ/kg·K]
• cp_cv: Heat capacity ratio, [-]
• Z: Compression factor, [-]
• fi: Fugacity coefficient, [-]
• f: Fugacity, [MPa]
• gamma: Isoentropic exponent, [-]
• alfav: Isobaric cubic expansion coefficient, [1/K]
• kappa: Isothermal compressibility, [1/MPa]
• kappas: Adiabatic compressibility, [1/MPa]
• alfap: Relative pressure coefficient, [1/K]
• betap: Isothermal stress coefficient, [kg/m³]
• joule: Joule-Thomson coefficient, [K/MPa]
• betas: Isoentropic temperature-pressure coefficient, [-]
• Gruneisen: Gruneisen parameter, [-]
• virialB: Second virial coefficient, [m³/kg]
• virialC: Third virial coefficient, [m⁶/kg²]
• dpdT_rh: Derivatives, dp/dT at constant rho, [MPa/K]
• dpdrho_T: Derivatives, dp/drho at constant T, [MPa·m³/kg]
• drhodT_P: Derivatives, drho/dT at constant P, [kg/m³·K]
• drhodT_P: Derivatives, drho/dP at constant T, [kg/m³·MPa]
• dhdT_rh: Derivatives, dh/dT at constant rho, [kJ/kg·K]
• dhdT_P: Isothermal throttling coefficient, [kJ/kg·MPa]
• dhdt_P: Derivatives, dh/dT at constant P, [kJ/kg·K]
• dhdrho_T: Derivatives, dh/drho at constant T, [kJ·m³/kg²]
• dhdrho_P: Derivatives, dh/drho at constant P, [kJ·m³/kg²]
• dhP_rh: Derivatives, dh/dP at constant rho, [kJ/kg·MPa]
• kt: Isothermal Expansion Coefficient, [-]
• ks: Adiabatic Compressibility, [1/MPa]
• Ks: Adiabatic bulk modulus, [MPa]
• Kt: Isothermal bulk modulus, [MPa]
• v0: Ideal specific volume, [m³/kg]
• rho0: Ideal gas density, [kg/m³]
• u0: Ideal specific internal energy, [kJ/kg]
• h0: Ideal specific enthalpy, [kJ/kg]
• s0: Ideal specific entropy, [kJ/kg·K]
• $a_0$: Ideal specific Helmholtz free energy, [kJ/kg]
• $g_0$: Ideal specific Gibbs free energy, [kJ/kg]
• $cp_0$: Ideal specific isobaric heat capacity, [kJ/kg·K]
• $cv_0$: Ideal specific isochoric heat capacity, [kJ/kg·K]
• $w_0$: Ideal speed of sound, [m/s]
• $\gamma_0$: Ideal isoentropic exponent, [-]
• $w$: Speed of sound, [m/s]
• $\mu$: Dynamic viscosity, [Pa·s]
• $\nu$: Kinematic viscosity, [m²/s]
• $k$: Thermal conductivity, [W/m·K]
• $\alpha$: Thermal diffusivity, [m²/s]
• $\sigma$: Surface tension, [N/m]
• $\epsilon$: Dielectric constant, [-]
• $n$: Refractive index, [-]
• $\text{Prandt}$: Prandtl number, [-]
• $\text{Pr}$: Reduced Pressure, [-]
• $\text{Tr}$: Reduced Temperature, [-]
• $H_{vap}$: Vaporization heat, [kJ/kg]
• $S_{vap}$: Vaporization entropy, [kJ/kg·K]
• $Z_{\rho}$: $(Z - 1)/\rho$, [m³/kg]
• $\text{IntP}$: Internal pressure, [MPa]
• $\text{invT}$: Negative reciprocal temperature, [1/K]
• $h_{\text{Input}}$: Specific heat input, [kJ/kg]

Attributes

| CP | Gruneisen | IntP | Ks | Kt | Prandt | Z | Z_{\rho} | a | \alpha | \alpha_{p} | \alpha_{v} |
betap
betas

_calculable_ Check if inputs are enough to define state

cp
cp_cv
cv
dhdP_T
dhdP_rhoe
dhdT_P
dhdT_rhoe
dhdrhoe_P
dhdrhoe_T
dpdrhoe_T
drhodP_T
drhodT_P
epsilon
f
fi
g
gamma
h
hInput
joule
k
kappa
ks
kt
mu
n
nu
rho
s
u
v
w
Methods

```python
__call__(self, **kwargs) Make instance callable to can add input parameter one to one

calculo(self) Calculate procedure

derivative(self, z, x, y, fase) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a

fill(self, fase, estado) Fill phase properties
```

```python
CP = None
_Pv = None
_rhOL = None
_rhOG = None
status = 0
msg = 'Undefined'
kwargs = {'P': 0.0, 'T': 0.0, 'T0': None, 'h': None, 'l': 0.5893, 'rho': 0.0, 'rho0': None, 's': None, 'u': None, 'v': 0.0, 'x': None, 'x0': 0.5}
calculable
    Check if inputs are enough to define state
calculo(self)
    Calculate procedure
fill (self, fase, estado)
    Fill phase properties
derivative (self, z, x, y, fase)
    Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a
_saturation (self, T)
    Saturation calculation for two phase search
_Helmholtz (self, rho, T)
    Calculated properties from helmholtz free energy and derivatives
```

Parameters

- `rho (float)` – Density, [kg/m^3]
- `T (float)` – Temperature, [K]

Returns

- `prop` – Dictionary with calculated properties:
  - fir: [-]
  - fird: fir/δτ
  - firdd: 2fir/δ^2τ
  - delta: Reducen density rho/rhoc, [-]
  - P: Pressure, [kPa]
  - h: Enthalpy, [kJ/kg]

5.1. iapws package
• s: Entropy, [kJ/kgK]
• cv: Isochoric specific heat, [kJ/kgK]
• alfav: Thermal expansion coefficient, [1/K]
• betap: Isothermal compressibility, [1/kPa]

Return type  dict

References


__prop0__(self, rho, T)
Ideal gas properties

__phi0__(self, tau, delta)
Ideal gas Helmholtz free energy and derivatives

Parameters
• tau (float) – Inverse reduced temperature Tc/T, [-]
• delta (float) – Reduced density rho/rhoc, [-]

Returns prop – fio, [-] fiot: fio/τ|δ
fiod: fio/δτ
fiott: 2fio/τ²|δ
fiodt: 2fio/τδ|τ
fiottt: 2fio/τ|δ²

Return type  dictionary with ideal adimensional helmholtz energy and deriv

References


__phir__(self, tau, delta)
Residual contribution to the free Helmholtz energy

Parameters
• tau (float) – Inverse reduced temperature Tc/T, [-]
• delta (float) – Reduced density rho/rhoc, [-]

Returns

prop –
Dictionary with residual adimensional helmholtz energy and deriv:

• fir
• firt: fir/τδ,x
• fird: fir/δτ,x
• firtt: 2fir/τ²δ,x
• firdt: 2fir/τδx
• firddd: 2fir/δ²τ,x
Return type  dict

References


_virial(self, T)
Virial coefficient

Parameters  T(float) – Temperature [K]

Returns

prop –
Dictionary with residual adimensional helmholtz energy:

• B: \( \frac{\partial f}{\partial \delta} |_{\delta \to 0} \)
• C: \( 2 \frac{\partial f}{\partial \delta^2} |_{\delta \to 0} \)

Return type  dict

_derivDimensional(self, rho, T)
Calcu the dimensional form or Helmholtz free energy derivatives

Parameters

• rho (float) – Density, [kg/m³]
• T (float) – Temperature, [K]

Returns

prop –
Dictionary with residual helmholtz energy and derivatives:

• fir, [kJ/kg]
• firt: \( \frac{\partial f}{\partial T} \rho, [kJ/kgK] \)
• fird: \( \frac{\partial f}{\partial \rho} T, [kJ/m³kg²] \)
• firtt: \( \frac{\partial^2 f}{\partial T^2} \rho, [kJ/kgK²] \)
• firdt: \( \frac{\partial^2 f}{\partial T \partial \rho}, [kJ/m³kg²K] \)
• firdd: \( \frac{\partial^2 f}{\partial \rho^2} T, [kJ/m⁶kg] \)

Return type  dict

References

IAPWS, Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, Table 7, http://www.iapws.org/relguide/SeaAir.html

_surface(self, T)
Generic equation for the surface tension

Parameters  T(float) – Temperature, [K]
Returns $\sigma$ – Surface tension, [N/m]
Return type float

Notes

Need a _surf dict in the derived class with the parameters keys: sigma: coefficient exp: exponent

classmethod _Vapor_Pressure($T$)
Auxiliary equation for the vapour pressure

Parameters $T$ (float) – Temperature, [K]
Returns $P_\text{v}$ – Vapour pressure, [Pa]
Return type float

References


classmethod _Liquid_Density($T$)
Auxiliary equation for the density of saturated liquid

Parameters $T$ (float) – Temperature, [K]
Returns $\rho_\text{s}$ – Saturated liquid density, [kg/m$^3$]
Return type float

References


classmethod _Vapor_Density($T$)
Auxiliary equation for the density of saturated vapor

Parameters $T$ (float) – Temperature, [K]
Returns $\rho_\text{v}$ – Saturated vapor density, [kg/m$^3$]
Return type float

References


classmethod _dPdT_sat($T$)
Auxiliary equation for the $dP/dT$ along saturation line

Parameters $T$ (float) – Temperature, [K]
Returns $dP/dT$ – dP/dT, [MPa/K]
Return type float
References


iapws.iapws95.mainClassDoc()
Function decorator used to automatic adiction of base class MEoS in subclass __doc__

class iapws.iapws95.IAPWS95(**kwargs)
Implementation of IAPWS Formulation 1995 for ordinary water substance, (revised release of 2016), for internal procedures, see MEoS base class

Parameters

- \(T\) (float) – Temperature, [K]
- \(P\) (float) – Pressure, [MPa]
- \(\rho\) (float) – Density, [kg/m\(^3\)]
- \(v\) (float) – Specific volume, [m\(^3\)/kg]
- \(h\) (float) – Specific enthalpy, [kJ/kg]
- \(s\) (float) – Specific entropy, [kJ/kgK]
- \(u\) (float) – Specific internal energy, [kJ/kg]
- \(x\) (float) – Vapor quality, [-]
- \(l\) (float, optional) – Wavelength of light, for refractive index, [nm]
- \(\rho_0\) (float, optional) – Initial value of density, to improve iteration, [kg/m\(^3\)]
- \(T_0\) (float, optional) – Initial value of temperature, to improve iteration, [K]
- \(x_0\) (Initial value of vapor quality, necessary in bad input pair definition) – where there are two valid solution (T-h, T-s)

Notes

- It needs two incoming properties of \(T, P, \rho, h, s, u\).
- \(v\) as a alternate input parameter to \(\rho\)
- \(T-x, P-x\), preferred input pair to specified a point in two phases region

The calculated instance has the following properties:

- \(P\): Pressure, [MPa]
- \(T\): Temperature, [K]
- \(x\): Vapor quality, [-]
- \(g\): Specific Gibbs free energy, [kJ/kg]
- \(a\): Specific Helmholtz free energy, [kJ/kg]
- \(v\): Specific volume, [m\(^3\)/kg]
- \(r\): Density, [kg/m\(^3\)]
- \(h\): Specific enthalpy, [kJ/kg]
- \(u\): Specific internal energy, [kJ/kg]
• \( s \): Specific entropy, \([kJ/kg\cdot K]\)
• \( cp \): Specific isobaric heat capacity, \([kJ/kg\cdot K]\)
• \( cv \): Specific isochoric heat capacity, \([kJ/kg\cdot K]\)
• \( cp_cv \): Heat capacity ratio, [-]
• \( Z \): Compression factor, [-]
• \( fi \): Fugacity coefficient, [-]
• \( f \): Fugacity, [MPa]
• \( \gamma \): Isoentropic exponent, [-]
• \( alfav \): Isobaric cubic expansion coefficient, \([1/K]\)
• \( \kappa \): Isothermal compressibility, \([1/MPa]\)
• \( kappas \): Adiabatic compressibility, \([1/MPa]\)
• \( alfav \): Relative pressure coefficient, \([1/K]\)
• \( betap \): Isothermal stress coefficient, \([kg/m^3]\)
• \( joule \): Joule-Thomson coefficient, \([K/MPa]\)
• \( betas \): Isoentropic temperature-pressure coefficient, [-]
• \( \gamma_e \): Gruneisen parameter, [-]
• \( virialB \): Second virial coefficient, \([m^3/kg]\)
• \( virialC \): Third virial coefficient, \([m^6/kg^2]\)
• \( dpdT_{rho} \): Derivatives, \(dp/dT\) at constant \(rho\), \([MPa/K]\)
• \( dpdrho_T \): Derivatives, \(dp/drho\) at constant \(T\), \([MPa\cdot m^3/kg]\)
• \( drhodT_P \): Derivatives, \(drho/dT\) at constant \(P\), \([kg/m^3\cdot K]\)
• \( drhodP_T \): Derivatives, \(drho/dP\) at constant \(T\), \([kg/m^3\cdot MPa]\)
• \( dhdT_{rho} \): Derivatives, \(dh/dT\) at constant \(rho\), \([kJ/kg\cdot K]\)
• \( dhP_T \): Derivatives, \(dh/dT\) at constant \(P\), \([kJ/kg\cdot MPa]\)
• \( dhdrho_T \): Derivatives, \(dh/drho\) at constant \(T\), \([kJ\cdot m^3/kg^2]\)
• \( dhdrho_P \): Derivatives, \(dh/drho\) at constant \(P\), \([kJ\cdot m^3/kg^2]\)
• \( dhP_{rho} \): Derivatives, \(dh/dP\) at constant \(rho\), \([kJ/kg\cdot MPa]\)
• \( \kappa_t \): Isothermal Expansion Coefficient, [-]
• \( ks \): Adiabatic Compressibility, \([1/MPa]\)
• \( Ks \): Adiabatic bulk modulus, \([MPa]\)
• \( Kt \): Isothermal bulk modulus, \([MPa]\)
• \( v0 \): Ideal specific volume, \([m^3/kg]\)
• \( rho0 \): Ideal gas density, \([kg/m^3]\)
• \( u0 \): Ideal specific internal energy, \([kJ/kg]\)
• \( h0 \): Ideal specific enthalpy, \([kJ/kg]\)
• s0: Ideal specific entropy, [kJ/kg·K]
• a0: Ideal specific Helmholtz free energy, [kJ/kg]
• g0: Ideal specific Gibbs free energy, [kJ/kg]
• cp0: Ideal specific isobaric heat capacity, [kJ/kg·K]
• cv0: Ideal specific isochoric heat capacity, [kJ/kg·K]
• w0: Ideal speed of sound, [m/s]
• gamma0: Ideal isoentropic exponent, [-]
• w: Speed of sound, [m/s]
• mu: Dynamic viscosity, [Pa·s]
• nu: Kinematic viscosity, [m²/s]
• k: Thermal conductivity, [W/m·K]
• alfa: Thermal diffusivity, [m²/s]
• sigma: Surface tension, [N/m]
• epsilon: Dielectric constant, [-]
• n: Refractive index, [-]
• Prandt: Prandtl number, [-]
• Pr: Reduced Pressure, [-]
• Tr: Reduced Temperature, [-]
• Hvap: Vaporization heat, [kJ/kg]
• Svap: Vaporization entropy, [kJ/kg·K]
• Z_rh0: (Z – 1)/ρ, [m³/kg]
• IntP: Internal pressure, [MPa]
• invT: Negative reciprocal temperature, [1/K]
• hInput: Specific heat input, [kJ/kg]

Examples

```python
>>> water=IAPWS95(T=300, rho=996.5560)
>>> water.P, water.cv, water.w, water.s
0.0992418350 4.13018112 1501.51914 0.393062643

>>> water=IAPWS95(T=500, rho=0.435)
>>> water.P, water.cv, water.w, water.s
0.0999679423 1.50817541 548.31425 7.944882714

>>> water=IAPWS95(T=900., P=700)
>>> water.rho, water.cv, water.w, water.s
870.7690 2.66422350 2019.33608 4.17223802
```
>>> water=IAPWS95(T=300., P=0.1)
>>> water.P, water.rho, water.h, water.s, water.cp, water.w, water.virialB
0.10000 996.56 112.65 0.39306 4.1806 1501.5 -0.066682

>>> water=IAPWS95(T=500., P=0.1)
>>> water.P, water.rho, water.h, water.s, water.cp, water.w, water.virialB
0.10000 0.43514 2928.6 7.9447 1.9813 548.31 -0.0094137

>>> water=IAPWS95(T=450., x=0.5)
>>> water.T, water.P, water.rho, water.h, water.s, water.virialB
450.00 0.93220 9.5723 1761.8 4.3589 -0.013028

>>> water=IAPWS95(P=1.5, rho=1000.)
>>> water.T, water.rho, water.h, water.s, water.w, water.virialB
286.44 1000.0 57.253 0.19931 4.1855 1462.1 -0.085566

>>> water=IAPWS95(h=3000, s=8.)
>>> water.T, water.P, water.h, water.s, water.cp, water.w, water.virialB
536.24 0.11970 3000.0 8.0000 1.9984 567.04 -0.0076606

>>> water=IAPWS95(h=150, s=0.4)
>>> water.T, water.P, water.rho, water.h, water.s, water.x, water.w
301.27 35.50549 1011.48 150.00 0.40000 4.0932 1564.1

>>> water=IAPWS95(T=450., rho=300)
>>> water.T, water.P, water.rho, water.h, water.s, water.x, water.virialB
450.00 0.93220 300.00 770.82 2.1568 0.010693 -0.013028

>>> water=IAPWS95(rho=300., P=0.1)
>>> water.T, water.P, water.rho, water.h, water.s, water.x, water.virialB
372.76 0.10000 300.00 150.00 2.6315 0.026071

>>> water=IAPWS95(T=500., u=900)
>>> water.P, water.rho, water.u, water.h, water.s, water.cp, water.w
108.21 903.62 900.00 1019.8 2.4271 4.1751 1576.0

>>> water=IAPWS95(P=0.3, u=1550.)
>>> water.T, water.P, water.rho, water.h, water.s, water.u
406.67 0.30000 3.3029 1550.0 1640.8 4.3260 0.49893

>>> water=IAPWS95(rho=300, h=1000.)
>>> water.T, water.P, water.rho, water.u, water.h, water.s, water.x
494.92 2.3991 300.00 992.00 1000.0 2.6315 0.026071
```python
>>> water=IAPWS95(rho=30, s=8.)
>>> water.T, water.P, water.rho, water.u, water.h, water.s, water.cp
1562.42 21.671 30.000 4628.5 5350.9 8.0000 2.7190

>>> water=IAPWS95(rho=30, s=4.)
>>> water.T, water.P, water.rho, water.u, water.h, water.s, water.x
495.00 2.4029 30.000 1597.3 1677.4 4.0000 0.39218

>>> water=IAPWS95(rho=300, u=1000.)
>>> water.T, water.P, water.rho, water.u, water.h, water.s, water.x
496.44 2.4691 300.000 1000.0 1008.2 2.6476 0.02680

>>> water=IAPWS95(s=3., h=1000.)
>>> water.T, water.P, water.rho, water.u, water.h, water.s, water.x
345.73 0.034850 0.73526 952.60 1000.0 3.0000 0.29920

>>> water=IAPWS95(u=995., h=1000.)
>>> water.T, water.P, water.rho, water.u, water.h, water.s, water.x
501.89 2.7329 546.58 995.00 1000.0 2.6298 0.00866

>>> water=IAPWS95(u=1000., s=3.)
>>> water.T, water.P, water.rho, water.u, water.h, water.s, water.x
371.24 0.094712 1.99072 1000.00 1047.6 3.0000 0.28144
```

## References


IAPWS, Guideline on a Low-Temperature Extension of the IAPWS-95 Formulation for Water Vapor, [http://www.iapws.org/relguide/LowT.html](http://www.iapws.org/relguide/LowT.html)

IAPWS, Revised Advisory Note No. 3: Thermodynamic Derivatives from IAPWS Formulations, [http://www.iapws.org/relguide/Advise3.pdf](http://www.iapws.org/relguide/Advise3.pdf)

### Attributes

- CP
- Gruneisen
- IntP
- Ks
- Kt
- Prandt
- Z
- Z_rho
- a
- alfa
calculable Check if inputs are enough to define state

cp
cp_cv
cv
dhdP_T
dhdP_rho
dhdT_P
dhdT_rho
dhdrho_P
dhdrho_T
dpdT_rho
dpdrho_T
drhodP_T
drhodT_P
epsilon
f
fi
g
gamma
h
hInput
joule
k
kappa
ks
kt
mu
n
nu
rho
s
u
Methods

__call__(self, **kwargs) Make instance callable to can add input parameter one to one

calculo(self) Calculate procedure

derivative(self, z, x, y, fase) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a

fill(self, fase, estado) Fill phase properties

```python
name = 'water'
CASNumber = '7732-18-5'
formula = 'H2O'
synonym = 'R-718'
Tc = 647.096
rhoc = 322.0
Pc = 22.064
M = 18.015268
Tt = 273.16
Tb = 373.1243
f_acent = 0.3443
momentoDipolar = 1.855
Fi0 = {'ao_exp': [0.012436, 0.97315, 1.2795, 0.96956, 0.24873], 'ao_log': [1, 3.00632], 'ao_pow': [-8.3204464837497, 6.6832105275932], 'pow': [0, 1], 'titao': [1.28728967, 3.53734222, 7.74073708, 9.24437796, 27.5075105]}
_constants = {'A': [0.32, 0.32], 'B': [0.2, 0.2], 'C': [28, 32], 'D': [700, 800], 'R': 8.314371357587, 'a4': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]}.
Pv = {'ao': [-7.85951783, 1.84408259, -11.7866497, 22.6807411, -15.9618719, 1.80122502]}
_rhoL = {'ao': [1.99274064, 1.09965342, -0.510839303, -1.75493479, -45.5170352, -674694.45]}
rhoG = {'ao': [-2.0315024, -2.6830294, -5.38626492, -17.2991605, -44.7586581, -63.9201063]}
_phio(self, tau, delta) Low temperature extension of the IAPWS-95

_classmethod _alfa_sat(T)
Auxiliary equation for the alfa coefficient for calculate the enthalpy along the saturation line

Parameters T (float) – Temperature, [K]

Returns alfa – alfa coefficient, [kJ/kg]

Return type float

5.1. iapws package 45
classmethod _phi_sat(T)
Auxiliary equation for the phi coefficient for calculate the entropy along the saturation line

Parameters T (float) – Temperature, [K]
Returns phi – phi coefficient, [kJ/kgK]
Return type float

classmethod _Liquid_Enthalpy(T)
Auxiliary equation for the specific enthalpy for saturated liquid

Parameters T (float) – Temperature, [K]
Returns h – Saturated liquid enthalpy, [kJ/kg]
Return type float

classmethod _Vapor_Enthalpy(T)
Auxiliary equation for the specific enthalpy for saturated vapor

Parameters T (float) – Temperature, [K]
Returns h – Saturated vapor enthalpy, [kJ/kg]
Return type float

classmethod _Liquid_Entropy(T)
Auxiliary equation for the specific entropy for saturated liquid

Parameters T (float) – Temperature, [K]
Returns s – Saturated liquid entropy, [kJ/kgK]
Return type float
References


classmethod _Vapor_Entropy(T)
Auxiliary equation for the specific entropy for saturated vapor

Parameters
- \(T\) (float) – Temperature, [K]

Returns
- \(s\) – Saturated liquid entropy, [kJ/kgK]

Return type
- float

References


_visco(self, rho, T, fase)

_thermo(self, rho, T, fase)

(surface(self, T)
Generic equation for the surface tension

Parameters
- \(T\) (float) – Temperature, [K]

Returns
- \(\sigma\) – Surface tension, [N/m]

Return type
- float

Notes

Need a _surf dict in the derived class with the parameters keys: sigma: coefficient exp: exponent

class iapws.iapws95.IAPWS95_PT(P, T)
Derived class for direct P and T input

Attributes
- CP
- Gruneisen
- IntP
- Ks
- Kt
- Prandt
- Z
- Z_rho
- a
- alfa
- alfap
- alfav
betap
betas
\textbf{calculable}  Check if inputs are enough to define state
cp
cp\_cv
cv
dhdP\_T
dhdP\_rho
dhdT\_P
dhdT\_rho
dhdrho\_P
dhdrho\_T
dp\_T\_rho
dpdrho\_T
drhodP\_T
drhodT\_P
epsilon
f
fi
g
\textit{gamma}
h
hInput
joule
k
\textit{kappa}
ks
kt
mu
n
\nu
\nu
\rho
s
u
v
w
Methods

```python
__call__(self, **kwargs) Make instance callable to can add input parameter one to one

calculo(self) Calculate procedure

derivative(self, z, x, y, fase) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a

fill(self, fase, estado) Fill phase properties
```

```python
class iapws.iapws95.IAPWS95_Ph(P, h)
Derivated class for direct P and h input

Attributes

CP
Gruneisen
IntP
Ks
Kt
Prandt
Z
Z_rho
a
alfa
alfap
alfav
betap
betas

calculable Check if inputs are enough to define state
cp
cp_cv
cv
dhdP_T
dhdP_rho
dhdT_P
dhdT_rho
dhdrho_P
dhdrho_T
dpdT_rho
dpdrho_T
drhodP_T
```
Methods

__call__(self, **kwargs)  Make instance callable to can add input parameter one to one

calculo(self) Calculate procedure
derivative(self, z, x, y, fase)  Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a

fill(self, fase, estado) Fill phase properties

class iapws.iapws95.IAPWS95_Ps(P, s)
   Derivated class for direct P and s input

Attributes

   CP
   Gruneisen
   IntP
   Ks
   Kt
Prandt
Z
Z_rho
a
alfa
alfap
alfav
betap
betas
**calculable**  Check if inputs are enough to define state
cp
cp_cv
cv
dhdP_T
dhdP_rhoe

dhdT_P
dhdT_rhoe

dhdrho_P
dhdrho_T
dpdT_rhoe

dpdrho_T

drhodP_T
drhodT_P
epsilon
f
fi
g
gamma
h
hInput
joule
k
kappa
ks
kt
mu
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong></td>
<td>Make instance callable to can add input parameter one to one</td>
</tr>
<tr>
<td>calculo</td>
<td>Calculate procedure</td>
</tr>
<tr>
<td>derivative</td>
<td>Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a</td>
</tr>
<tr>
<td>fill</td>
<td>Fill phase properties</td>
</tr>
</tbody>
</table>

```python
class iapws.iapws95.IAPWS95_Px(P, x):
    Derivated class for direct P and v input
```

#### Attributes

- CP
- Gruneisen
- IntP
- Ks
- Kt
- Prandt
- Z
- Z_rho
- a
- alfa
- alfap
- alfav
- betap
- betas

**calculable** Check if inputs are enough to define state

- cp
- cp_cv
- cv
- dhdP_T
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(self, **kwargs)</td>
<td>Make instance callable to can add input parameter one to one</td>
</tr>
<tr>
<td>calculo(self)</td>
<td>Calculate procedure</td>
</tr>
<tr>
<td>derivative(self, z, x, y, fase)</td>
<td>Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a</td>
</tr>
</tbody>
</table>

Continued on next page
Table 6 – continued from previous page

| fill(self, fase, estado) | Fill phase properties |

class iapws.iapws95.IAPWS95_Tx(T, x)

Derivated class for direct T and x input

Attributes

CP
Gruneisen
IntP
Ks
Kt
Prandt
Z
Z_rho
a
alfa
alfap
alfav
betap
betas

Calculable Check if inputs are enough to define state

cp
cp_cv
cv
dhdP_T
dhdP_rh0
dhdT_P
dhdT_rh0
dhdrho_P
dhdrho_T
dpdT_rh0
dpdrho_T
drhodP_T
drhodT_P
epsilon
f
fi
Methods

- `__call__(self, **kwargs)` Make instance callable to can add input parameter one to one
- `calculo(self)` Calculate procedure
- `derivative(self, x, y, z)` Wrapper derivative for custom derived properties where x, y, z can be: P, T, rho, v, h, s, g, a
- `fill(self, fase, estado)` Fill phase properties

**class** `iapws.iapws95.D2O(**kwargs)`
Implementation of IAPWS Formulation for heavy water substance, for internal procedures, see MEoS base class

**Parameters**

- `T (float)` – Temperature, [K]
- `P (float)` – Pressure, [MPa]
- `rho (float)` – Density, [kg/m³]
- `v (float)` – Specific volume, [m³/kg]
- `h (float)` – Specific enthalpy, [kJ/kg]
- `s (float)` – Specific entropy, [kJ/kgK]
- `u (float)` – Specific internal energy, [kJ/kg]
- `x (float)` – Vapor quality, [-]
- `l (float, optional)` – Wavelength of light, for refractive index, [nm]
• **rho0** (*float, optional*) – Initial value of density, to improve iteration, [kg/m³]
• **T0** (*float, optional*) – Initial value of temperature, to improve iteration, [K]
• **x0** *(initial value of vapor quality, necessary in bad input pair definition)* – where there are two valid solution (T-h, T-s)

**Notes**

- It needs two incoming properties of T, P, rho, h, s, u.
- v as a alternate input parameter to rho
- T-x, P-x, preferred input pair to specified a point in two phases region

The calculated instance has the following properties:

- P: Pressure, [MPa]
- T: Temperature, [K]
- x: Vapor quality, [-]
- g: Specific Gibbs free energy, [kJ/kg]
- a: Specific Helmholtz free energy, [kJ/kg]
- v: Specific volume, [m³/kg]
- r: Density, [kg/m³]
- h: Specific enthalpy, [kJ/kg]
- u: Specific internal energy, [kJ/kg]
- s: Specific entropy, [kJ/kg·K]
- cp: Specific isobaric heat capacity, [kJ/kg·K]
- cv: Specific isochoric heat capacity, [kJ/kg·K]
- cp_cv: Heat capacity ratio, [-]
- Z: Compression factor, [-]
- fi: Fugacity coefficient, [-]
- f: Fugacity, [MPa]
- gamma: Isoentropic exponent, [-]
- alfav: Isobaric cubic expansion coefficient, [1/K]
- kappa: Isothermal compressibility, [1/MPa]
- kappas: Adiabatic compresibility, [1/MPa]
- alfap: Relative pressure coefficient, [1/K]
- betap: Isothermal stress coefficient, [kg/m³]
- joule: Joule-Thomson coefficient, [K/MPa]
- betas: Isoentropic temperature-pressure coefficient, [-]
- Gruneisen: Gruneisen parameter, [-]
- virialB: Second virial coefficient, [m³/kg]
• virialC: Third virial coefficient, \([\text{m}^6/\text{kg}^2]\)
• d\(p\)/dT\(_\text{rho}\): Derivatives, \(dp/dT\) at constant \(\rho\), [\text{MPa/K}]
• d\(\rho\)/dT\(_T\): Derivatives, \(d\rho/dT\) at constant \(T\), [\text{MPa}\cdot\text{m}^3/\text{kg}]
• d\(\rho\)/dP\(_T\): Derivatives, \(d\rho/dP\) at constant \(T\), [\text{kg/m}^3\cdot\text{MPa}]
• dh\(dT\)/rho: Derivatives, \(dh/dT\) at constant \(\rho\), [\text{kJ/kg}\cdot\text{K}]
• dh\(dP\)/T: Derivatives, \(dh/dP\) at constant \(T\), [\text{kJ/kg}]
• dh\(d\rho\)/T: Derivatives, \(dh/d\rho\) at constant \(T\), [\text{kJ}\cdot\text{m}^3/\text{kg}^2]
• dh\(d\rho\)/P: Derivatives, \(dh/d\rho\) at constant \(P\), [\text{kJ}\cdot\text{m}^3/\text{kg}^2]
• dh\(dP\)/rho: Derivatives, \(dh/dP\) at constant \(\rho\), [\text{kJ/kg}\cdot\text{MPa}]
• kt: Isothermal expansion coefficient, [-]
• ks: Adiabatic compressibility, [1/\text{MPa}]
• Ks: Adiabatic bulk modulus, [\text{MPa}]
• Kt: Isothermal bulk modulus, [\text{MPa}]
• \(v_0\): Ideal specific volume, [\text{m}^3/\text{kg}]
• \(\rho_0\): Ideal gas density, [\text{kg/m}^3]
• \(u_0\): Ideal specific internal energy, [\text{kJ/kg}]
• \(h_0\): Ideal specific enthalpy, [\text{kJ/kg}]
• \(s_0\): Ideal specific entropy, [\text{kJ/kg}\cdot\text{K}]
• \(a_0\): Ideal specific Helmholtz free energy, [\text{kJ/kg}]
• \(g_0\): Ideal specific Gibbs free energy, [\text{kJ/kg}]
• \(c_{p0}\): Ideal specific isobaric heat capacity, [\text{kJ/kg}\cdot\text{K}]
• \(c_{v0}\): Ideal specific isochoric heat capacity, [\text{kJ/kg}\cdot\text{K}]
• \(w_0\): Ideal speed of sound, [\text{m/s}]
• gamma0: Ideal isentropic exponent, [-]
• \(w\): Speed of sound, [\text{m/s}]
• \(\mu\): Dynamic viscosity, [\text{Pa}\cdot\text{s}]
• \(\nu\): Kinematic viscosity, [\text{m}^2/\text{s}]
• \(k\): Thermal conductivity, [\text{W/m}\cdot\text{K}]
• \(\alpha\): Thermal diffusivity, [\text{m}^2/\text{s}]
• \(\sigma\): Surface tension, [\text{N/m}]
• epsilon: Dielectric constant, [-]
• \(n\): Refractive index, [-]
• Prandt: Prandtl number, [-]
• Pr: Reduced Pressure, [-]
• Tr: Reduced Temperature, [-]
• Hvap: Vaporization heat, [kJ/kg]
• Svap: Vaporization entropy, [kJ/kg·K]
• Z_rhoe: $(Z - 1)/\rho$, [m³/kg]
• IntP: Internal pressure, [MPa]
• invT: Negative reciprocal temperature, [1/K]
• hInput: Specific heat input, [kJ/kg]

**Examples**

```python
>>> hwater=D2O(T=300, rho=996.5560)
>>> hwater.P, hwater.Liquid.cv, hwater.Liquid.w
0.0030675947 4.21191157 5332.04871
```

**References**


**Attributes**

- CP
- Gruneisen
- IntP
- Ks
- Kt
- Prandt
- Z
- Z_rhoe
- a
- alfa
- alfap
- alfav
- betap
- betas

**calculable** Check if inputs are enough to define state

- cp
- cp_cv
- cv
- dhdP_T
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(self, **kwargs)</td>
<td>Make instance callable to can add input parameter one to one</td>
</tr>
<tr>
<td>calculo(self)</td>
<td>Calculate procedure</td>
</tr>
<tr>
<td>derivative(self, z, x, y, fase)</td>
<td>Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a</td>
</tr>
</tbody>
</table>

Continued on next page
Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>fill(self, fase, estado)</th>
<th>Fill phase properties</th>
</tr>
</thead>
</table>

```
name = 'heavy water'
CASNumber = '7789-20-0'
formula = 'D2O'
synonym = 'deuterium oxide'
Tc = 643.847
rhoc = 355.9999698294
Pc = 21.671
M = 20.027508
Tt = 276.97
Tb = 374.563
f_acent = 0.364
momentoDipolar = 1.9
```

Fi0 = {'ao_exp': [0.010633, 0.99787, 2.1483, 0.3549], 'ao_hyp': [], 'ao_log': [1, 3],
_constants = {'R': 8.3144598, 'alfa3': [0.6014, 1.4723, 1.5305, 2.4297, 1.3086, 1.3528, 3.4456, 1.2645, 2.5547, 1.2148, ...
_Pv = {'ao': [-8.0236, 2.3957, -42.639, 99.569, -62.135], 'exp': [1.0, 1.5, 2.75, 3.0, 3.2]
_rhol = {'ao': [2.6406, 9.709, -18.058, 8.7202, -7.4487], 'eq': 1, 'exp': [0.3678, 1.9, 2.2, 2.63, 7.3]
_rhog = {'ao': [-3.7651, -38.673, 73.024, -132.51, 75.235, -70.412], 'eq': 3, 'exp': ...

Generic equation for the surface tension

Parameters T (float) – Temperature, [K]
Returns σ – Surface tension, [N/m]
Return type float

Notes

Need a _surf dict in the derived class with the parameters keys: sigma: coefficient exp: exponent

5.1.4 iapws.iapws97 module

IAPWS-IF97 standard implementation
The module implements the fundamental equation for the five regions (rectangular boxes) and the backward equation (marked in grey).

**IAPWS97**: Global module class with all the functionality integrated

**Fundamental equations:**
- `_Region1()`
- `_Region2()`
- `_Region3()`
- `_Region4()`
- `_TSat_P()`
- `_PSat_T()`
- `_Region5()`

**Backward equations:**
- `_Backward1_T_Ph()`
- `_Backward1_T_Ps()`
- `_Backward1_P_hs()`
- `_Backward2_T_Ph()`
• _Backward2_T_Ps()
• _Backward2_P_hs()
• _Backward3_T_Ph()
• _Backward3_T_Ps()
• _Backward3_P_hs()
• _Backward3_v_Ph()
• _Backward3_v_Ps()
• _Backward3_v_PT()
• _Backward4_T_hs()

Boundary equations:
• _h13_s()
• _h3a_s()
• _h1_s()
• _t_hs()
• _PSat_h()
• _h2ab_s()
• _h_3ab()
• _h2c3b_s()
• _hab_s()
• _hbc_P()

References:


IAPWS, Revised Supplementary Release on Backward Equations for the Functions T(p,h), v(p,h) and T(p,s), v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf

IAPWS, Revised Supplementary Release on Backward Equations p(h,s) for Region 3, Equations as a Function of h and s for the Region Boundaries, and an Equation Tsat(h,s) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf


iapws.iapws97._h13_s(s)
Parameters  \texttt{s} (\texttt{float}) – Specific entropy, [kJ/kgK]

Returns  \texttt{h} – Specific enthalpy, [kJ/kg]

Return type  \texttt{float}

Notes

Raise \texttt{NotImplementedError} if input isn’t in limit:

- \texttt{s}(100\text{MPa},623.15\text{K}) \texttt{ s} \texttt{s’}(623.15\text{K})

References

IAPWS, Revised Supplementary Release on Backward Equations \texttt{p(h,s)} for Region 3, Equations as a Function of \texttt{h} and \texttt{s} for the Region Boundaries, and an Equation \texttt{Tsat(h,s)} for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \url{http://www.iapws.org/relguide/Supp-phs3-2014.pdf}, Eq 7

Examples

\begin{verbatim}
>>> _h13_s(3.7)
1632.525047
>>> _h13_s(3.5)
1566.104611
\end{verbatim}

\texttt{iapws.iapws97.\_P23\_T}(\texttt{T})

Parameters  \texttt{T} (\texttt{float}) – Temperature, [K]

Returns  \texttt{P} – Pressure, [MPa]

Return type  \texttt{float}

References


Examples

\begin{verbatim}
>>> _P23_T(623.15)
16.52916425
\end{verbatim}

\texttt{iapws.iapws97.\_t\_P}(\texttt{P})

Parameters  \texttt{P} (\texttt{float}) – Pressure, [MPa]

Returns  \texttt{T} – Temperature, [K]

Return type  \texttt{float}
References


Examples

>>> _t_P(16.52916425)
623.15

iapws.iapws97._t_hs(h, s)

Parameters

- h (float) – Specific enthalpy, [kJ/kg]
- s (float) – Specific entropy, [kJ/kgK]

Returns T – Temperature, [K]

Return type float

Notes

Raise NotImplementedError if input isn’t in limit:

- 5.048096828 s 5.260578707
- 2.563592004e3 h 2.812942061e3

References

IAPWS, Revised Supplementary Release on Backward Equations p(h,s) for Region 3, Equations as a Function of h and s for the Region Boundaries, and an Equation Tsat(h,s) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 8

Examples

>>> _t_hs(2600, 5.1)
713.5259364
>>> _t_hs(2800, 5.2)
817.6202120

iapws.iapws97._PSat_T(T)

Parameters T (float) – Temperature, [K]

Returns P – Pressure, [MPa]

Return type float
Notes

Raise `NotImplementedError` if input isn’t in limit:

- 273.15 T 647.096

References


Examples

```python
>>> _PSat_T(500)
2.63889776
```

```
iapws.iapws97._TSat_P(P)

Parameters  
P (float) – Pressure, [MPa]

Returns  
T – Temperature, [K]

Return type  
float
```

Notes

Raise `NotImplementedError` if input isn’t in limit:

- 0.00061121 P 22.064

References


Examples

```python
>>> _TSat_P(10)
584.149488
```

```
iapws.iapws97._PSat_h(h)

Define the saturated line, P=f(h) for region 3

Parameters  
h (float) – Specific enthalpy, [kJ/kg]

Returns  
P – Pressure, [MPa]

Return type  
float
```

Notes

Raise `NotImplementedError` if input isn’t in limit:

- h'(623.15K) h h''(623.15K)
References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions \(T(p,h)\), \(v(p,h)\) and \(T(p,s)\), \(v(p,s)\) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 10

Examples

```python
>>> _PSat_h(1700)
17.24175718
>>> _PSat_h(2400)
20.18090839
```

`iapws.iapws97._PSat_h(s)`

Define the saturated line, \(P=f(s)\) for region 3

Parameters  
\(s\) (float) – Specific entropy, \([kJ/kgK]\)

Returns  
\(P\) – Pressure, \([MPa]\)

Return type  
float

Notes

Raise `NotImplementedError` if input isn’t in limit:

- \(s'(623.15K)\) \(s\) \(s''(623.15K)\)

References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions \(T(p,h)\), \(v(p,h)\) and \(T(p,s)\), \(v(p,s)\) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 11

Examples

```python
>>> _PSat_s(3.8)
16.87755057
>>> _PSat_s(5.2)
16.68968482
```

`iapws.iapws97._PSat_s(s)`

Parameters  
\(s\) (float) – Specific entropy, \([kJ/kgK]\)

Returns  
\(h\) – Specific enthalpy, \([kJ/kg]\)

Return type  
float

Notes

Raise `NotImplementedError` if input isn’t in limit:

- \(s'(273.15K)\) \(s\) \(s''(623.15K)\)
References

IAPWS, Revised Supplementary Release on Backward Equations p(h,s) for Region 3, Equations as a Function of h and s for the Region Boundaries, and an Equation Tsat(h,s) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 3

Examples

```python
>>> _h1_s(1)
308.5509647
>>> _h1_s(3)
1198.359754
```

iapws.iapws97._h3a_s(s)

**Parameters**  
`s` (*float*) – Specific entropy, [kJ/kgK]

**Returns**  
`h` – Specific enthalpy, [kJ/kg]

**Return type**  
`float`

**Notes**

Raise `NotImplementedError` if input isn’t in limit:

- `s'(623.15K)`

References

IAPWS, Revised Supplementary Release on Backward Equations p(h,s) for Region 3, Equations as a Function of h and s for the Region Boundaries, and an Equation Tsat(h,s) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 4

Examples

```python
>>> _h3a_s(3.8)
1685.025565
>>> _h3a_s(4.2)
1949.352563
```

iapws.iapws97._h2ab_s(s)

Define the saturated line boundary between Region 4 and 2a-2b, h=f(s)

**Parameters**  
`s` (*float*) – Specific entropy, [kJ/kgK]

**Returns**  
`h` – Specific enthalpy, [kJ/kg]

**Return type**  
`float`
Notes

Raise `NotImplementedError` if input isn’t in limit:

- \( s \approx 5.85 \) (273.15K)

References

IAPWS, Revised Supplementary Release on Backward Equations \( p(h,s) \) for Region 3, Equations as a Function of \( h \) and \( s \) for the Region Boundaries, and an Equation \( T_{sat}(h,s) \) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 5

Examples

```python
>>> _h2ab_s(7)
2723.729985
>>> _h2ab_s(9)
2511.861477
```

```
iapws.iapws97._h2c3b_s (s)
Define the saturated line boundary between Region 4 and 2c-3b, \( h=f(s) \)

Parameters

- `s` (float) – Specific entropy, [kJ/kgK]

Returns

- \( h \) – Specific enthalpy, [kJ/kg]

Return type

float

Notes

Raise `NotImplementedError` if input isn’t in limit:

- \( sc \approx 5.85 \)

References

IAPWS, Revised Supplementary Release on Backward Equations \( p(h,s) \) for Region 3, Equations as a Function of \( h \) and \( s \) for the Region Boundaries, and an Equation \( T_{sat}(h,s) \) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 6

Examples

```python
>>> _h2c3b_s(5.5)
2687.693850
>>> _h2c3b_s(4.5)
2144.360448
```

```
iapws.iapws97._Region1 (T, P)
Basic equation for region 1

Parameters

```
• \( T \) (float) – Temperature, [K]
• \( P \) (float) – Pressure, [MPa]

Returns

prop –

Dict with calculated properties. The available properties are:

• \( v \): Specific volume, \([m^3/kg]\)
• \( h \): Specific enthalpy, \([kJ/kg]\)
• \( s \): Specific entropy, \([kJ/kgK]\)
• \( cp \): Specific isobaric heat capacity, \([kJ/kgK]\)
• \( cv \): Specific isocoric heat capacity, \([kJ/kgK]\)
• \( w \): Speed of sound, \([m/s]\)
• \( alfav \): Cubic expansion coefficient, \([1/K]\)
• \( kt \): Isothermal compressibility, \([1/MPa]\)

Return type  dict

References


Examples

```python
>>> _Region1(300,3)['v']
0.00100215168
>>> _Region1(300,3)['h']
115.331273
>>> _Region1(300,3)['h']-3000*_Region1(300,3)['v']
112.324818
>>> _Region1(300,80)['s']
0.368563852
>>> _Region1(300,80)['cp']
4.01008987
>>> _Region1(300,80)['cv']
3.91736606
>>> _Region1(500,3)['w']
1240.71337
>>> _Region1(500,3)['alfav']
0.00164118128
>>> _Region1(500,3)['kt']
0.00112892188
```

iapws.iapws97._Backward1_T_Ph\((P,h)\)

Parameters

• \( P \) (float) – Pressure, [MPa]
• \( h \) (float) – Specific enthalpy, [kJ/kg]

5.1. iapws package 69
iapws Documentation, Release 1.1.3

Returns \( T \) – Temperature, [K]

Return type \( \text{float} \)

References


Examples

```python
>>> _Backward1_T_Ph(3,500)
391.798509
>>> _Backward1_T_Ph(80,1500)
611.041229
```

\texttt{iapws.iapws97.\_Backward1\_T\_Ps}(P,s)

Parameters

- \( P \ (\text{float}) \) – Pressure, [MPa]
- \( s \ (\text{float}) \) – Specific entropy, [kJ/kgK]

Returns \( T \) – Temperature, [K]

Return type \( \text{float} \)

References


Examples

```python
>>> _Backward1_T_Ps(3,0.5)
307.842258
>>> _Backward1_T_Ps(80,3)
565.899909
```

\texttt{iapws.iapws97.\_Backward1\_P\_hs}(h,s)

Parameters

- \( h \ (\text{float}) \) – Specific enthalpy, [kJ/kg]
- \( s \ (\text{float}) \) – Specific entropy, [kJ/kgK]

Returns \( P \) – Pressure, [MPa]

Return type \( \text{float} \)
References

IAPWS, Revised Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy \( p(h,s) \) for Regions 1 and 2 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-PHS12-2014.pdf, Eq 1

Examples

```python
>>> _Backward1_P_hs(0.001,0)
0.0009800980612
>>> _Backward1_P_hs(90,0)
91.92954727
>>> _Backward1_P_hs(1500,3.4)
58.68294423
```

```
iapws.iapws97._Region2(\(T, P\))
Basic equation for region 2

Parameters

- \( T \) (float) – Temperature, [K]
- \( P \) (float) – Pressure, [MPa]

Returns

\( prop \) –

Dict with calculated properties. The available properties are:

- \( v \): Specific volume, [m\(^3\)/kg]
- \( h \): Specific enthalpy, [kJ/kg]
- \( s \): Specific entropy, [kJ/kgK]
- \( cp \): Specific isobaric heat capacity, [kJ/kgK]
- \( cv \): Specific isochoric heat capacity, [kJ/kgK]
- \( w \): Speed of sound, [m/s]
- \( alfav \): Cubic expansion coefficient, [1/K]
- \( kt \): Isothermal compressibility, [1/MPa]

Return type  dict
```

References


Examples

```python
>>> _Region2(700,30)["v"]
0.00542946619
>>> _Region2(700,30)["h"]
2631.49474
```

(continues on next page)
iapws.iapws97.Region2_cp0(Tr, Pr)

Ideal properties for Region 2

Parameters
- **Tr** (*float*) – Reduced temperature, [-]
- **Pr** (*float*) – Reduced pressure, [-]

Returns
- **prop** – Array with ideal Gibbs energy partial derivatives:
  - **g**: Ideal Specific Gibbs energy [kJ/kg]
  - **gp**: g/P|T
  - **gpp**: 2g/P^2|T
  - **gt**: g/T|P
  - **gtt**: 2g/T^2|P
  - **gpt**: 2g/TP

Return type  array

References


iapws.iapws97._P_2bc(h)

Parameters  **h** (*float*) – Specific enthalpy, [kJ/kg]

Returns  **P** – Pressure, [MPa]

Return type  float

References

Examples

```python
>>> _P_2bc(3516.004323)
100.0
```

iapws.iapws97._hbc_P(P)

Parameters

- **P** (*float*) – Pressure, [MPa]

Returns

- **h** – Specific enthalpy, [kJ/kg]

Return type

*float*

References


Examples

```python
>>> _hbc_P(100)
3516.004323
```

iapws.iapws97._hab_s(s)

Parameters

- **s** (*float*) – Specific entropy, [kJ/kgK]

Returns

- **h** – Specific enthalpy, [kJ/kg]

Return type

*float*

References


Examples

```python
>>> _hab_s(7)
3376.437884
```

iapws.iapws97._Backward2a_T_Ph(P, h)

Parameters

- **P** (*float*) – Pressure, [MPa]
- **h** (*float*) – Specific enthalpy, [kJ/kg]

Returns

- **T** – Temperature, [K]

Return type

*float*
References


Examples

```python
>>> _Backward2a_T_Ph(0.001,3000)
534.433241
>>> _Backward2a_T_Ph(3,4000)
1010.77577
```

iapws.iapws97._Backward2b_T_Ph(P,h)

Parameters

- P (float) – Pressure, [MPa]
- h (float) – Specific enthalpy, [kJ/kg]

Returns T – Temperature, [K]

Return type float

References


Examples

```python
>>> _Backward2b_T_Ph(5,4000)
1015.31583
>>> _Backward2b_T_Ph(25,3500)
875.279054
```

iapws.iapws97._Backward2c_T_Ph(P,h)

Parameters

- P (float) – Pressure, [MPa]
- h (float) – Specific enthalpy, [kJ/kg]

Returns T – Temperature, [K]

Return type float

References

Examples

```python
>>> _Backward2c_T_Ph(40,2700)
743.056411
>>> _Backward2c_T_Ph(60,3200)
882.756860
```

iapws.iapws97._Backward2c_T_Ph($P, h$)

**Parameters**
- $P$ (*float*) – Pressure, [MPa]
- $h$ (*float*) – Specific enthalpy, [kJ/kg]

**Returns**
- $T$ – Temperature, [K]

**Return type**
- float

iapws.iapws97._Backward2a_T_Ps($P, s$)

**Parameters**
- $P$ (*float*) – Pressure, [MPa]
- $s$ (*float*) – Specific entropy, [kJ/kgK]

**Returns**
- $T$ – Temperature, [K]

**Return type**
- float

References


```python
>>> _Backward2a_T_Ps(0.1,7.5)
399.517097
>>> _Backward2a_T_Ps(2.5,8)
1039.84917
```

iapws.iapws97._Backward2b_T_Ps($P, s$)

**Parameters**
- $P$ (*float*) – Pressure, [MPa]
- $s$ (*float*) – Specific entropy, [kJ/kgK]

**Returns**
- $T$ – Temperature, [K]

**Return type**
- float

References

Examples

```python
>>> _Backward2b_T_Ps(8, 6)
600.484040
>>> _Backward2b_T_Ps(90, 6)
1038.01126
```

iapws.iapws97._Backward2b_T_Ps(P, s)

Parameters

- `P` *(float)* – Pressure, [MPa]
- `s` *(float)* – Specific entropy, [kJ/kgK]

Returns `T` – Temperature, [K]

Return type *float*

References


Examples

```python
>>> _Backward2c_T_Ps(20, 5.75)
697.992849
>>> _Backward2c_T_Ps(80, 5.75)
949.017998
```

iapws.iapws97._Backward2c_T_Ps(P, s)

Parameters

- `P` *(float)* – Pressure, [MPa]
- `s` *(float)* – Specific entropy, [kJ/kgK]

Returns `T` – Temperature, [K]

Return type *float*

References

Examples

```python
>>> _Backward2a_P_hs(2800,6.5)
1.371012767
>>> _Backward2a_P_hs(2800,9.5)
0.001879743844
>>> _Backward2a_P_hs(4100,9.5)
0.1024788997
```

iapws.iapws97._Backward2b_P_hs(h, s)

Parameters

- **h** (*float*) – Specific enthalpy, [kJ/kg]
- **s** (*float*) – Specific entropy, [kJ/kgK]

Returns **P** – Pressure, [MPa]

Return type **float**

References

IAPWS, Revised Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy p(h, s) for Regions 1 and 2 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-PHS12-2014.pdf, Eq 4

Examples

```python
>>> _Backward2b_P_hs(2800,6)
4.793911442
>>> _Backward2b_P_hs(3600,6)
83.95519209
>>> _Backward2b_P_hs(3600,7)
7.527161441
```

iapws.iapws97._Backward2c_P_hs(h, s)

Parameters

- **h** (*float*) – Specific enthalpy, [kJ/kg]
- **s** (*float*) – Specific entropy, [kJ/kgK]

Returns **P** – Pressure, [MPa]

Return type **float**

References

Examples

>>> _Backward2c_P_hs(2800,5.1)
94.39202060
>>> _Backward2c_P_hs(2800,5.8)
8.414574124
>>> _Backward2c_P_hs(3400,5.8)
83.76903879

iapws.iapws97._Backward2c_P_hs(h, s)

Parameters

• h (float) – Specific enthalpy, [kJ/kg]
• s (float) – Specific entropy, [kJ/kgK]

Returns

P – Pressure, [MPa]

Return type: float

iapws.iapws97._Region3(rho, T)

Basic equation for region 3

Parameters

• rho (float) – Density, [kg/m³]
• T (float) – Temperature, [K]

Returns

prop –

Dict with calculated properties. The available properties are:

• v: Specific volume, [m³/kg]
• h: Specific enthalpy, [kJ/kg]
• s: Specific entropy, [kJ/kgK]
• cp: Specific isobaric heat capacity, [kJ/kgK]
• cv: Specific isocoric heat capacity, [kJ/kgK]
• w: Speed of sound, [m/s]
• alfav: Cubic expansion coefficient, [1/K]
• kt: Isothermal compressibility, [1/MPa]

Return type: dict

References


Examples
iapws.iapws97._h_3ab(P)
Define the boundary between Region 3a-3b, h=f(P)

Parameters  
P (float) – Pressure, [MPa]

Returns  
h – Specific enthalpy, [kJ/kg]

Return type  float

Examples

```python
>>> _h_3ab(25)
2095.936454
```

iapws.iapws97._tab_P(P)
Define the boundary between Region 3a-3b, T=f(P)

Parameters  
P (float) – Pressure, [MPa]

Returns  
T – Temperature, [K]

Return type  float

References


Examples

```python
>>> _tab_P(40)
693.0341408
```

iapws.iapws97._top_P(P)
Define the boundary between Region 3o-3p, T=f(P)
Parameters  \( P (\text{float}) \) – Pressure, [MPa]

Returns  \( T \) – Temperature, [K]

Return type  \( \text{float} \)

References

IAPWS, Revised Supplementary Release on Backward Equations for Specific Volume as a Function of Pressure and Temperature \( v(p,T) \) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-VPT3-2016.pdf, Eq. 2

Examples

```python
>>> _top_P(22.8)
650.0106943
```

\texttt{iapws.iapws97._twx_P}(\( P \))  
Define the boundary between Region 3w-3x, \( T=f(P) \)

Parameters  \( P (\text{float}) \) – Pressure, [MPa]

Returns  \( T \) – Temperature, [K]

Return type  \( \text{float} \)

References

IAPWS, Revised Supplementary Release on Backward Equations for Specific Volume as a Function of Pressure and Temperature \( v(p,T) \) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-VPT3-2016.pdf, Eq. 2

Examples

```python
>>> _twx_P(22.3)
648.2049480
```

\texttt{iapws.iapws97._tef_P}(\( P \))  
Define the boundary between Region 3e-3f, \( T=f(P) \)

Parameters  \( P (\text{float}) \) – Pressure, [MPa]

Returns  \( T \) – Temperature, [K]

Return type  \( \text{float} \)

References

Examples

```python
>>> _tef_P(40)
713.9593992
```

iapws.iapws97._txx_P(P, xy)

Define the boundary between 3x-3y, T=f(P)

Parameters
- P (float) – Pressure, [MPa]
- xy (string) – Subregions options: cd, gh, ij, jk, mn, qu, rx, uv

Returns T – Temperature, [K]

Return type float

References


Examples

```python
>>> _txx_P(25, "cd")
649.3659208
>>> _txx_P(23, "gh")
649.8873759
>>> _txx_P(23, "ij")
651.5778091
>>> _txx_P(23, "jk")
655.8338344
>>> _txx_P(22.8, "mn")
649.6054133
>>> _txx_P(22, "qu")
645.6355027
>>> _txx_P(22, "rx")
648.2622754
>>> _txx_P(22.3, "uv")
647.7996121
```

iapws.iapws97._Backward3a_v_Ph(P, h)

Parameters
- P (float) – Pressure, [MPa]
- h (float) – Specific enthalpy, [kJ/kg]

References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions T(p,h), v(p,h) and T(p,s), v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 4
Returns \( v \) – Specific volume, [m\(^3\)/kg]

Return type float

Examples

```python
>>> _Backward3a_v_Ph(20,1700)
0.001749903962
>>> _Backward3a_v_Ph(100,2100)
0.001676229776
```

iapws.iapws97._Backward3b_v_Ph\( (P,h) \)

Parameters

- \( P (float) \) – Pressure, [MPa]
- \( h (float) \) – Specific enthalpy, [kJ/kg]

Returns \( v \) – Specific volume, [m\(^3\)/kg]

Return type float

References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions \( T(p,h) \), \( v(p,h) \) and \( T(p,s) \), \( v(p,s) \) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%29-2014.pdf, Eq 5

Examples

```python
>>> _Backward3b_v_Ph(20,2500)
0.006670547043
>>> _Backward3b_v_Ph(100,2700)
0.002404234998
```

iapws.iapws97._Backward3_v_Ph\( (P,h) \)

Parameters

- \( P (float) \) – Pressure, [MPa]
- \( h (float) \) – Specific enthalpy, [kJ/kg]

Returns \( v \) – Specific volume, [m\(^3\)/kg]

Return type float

iapws.iapws97._Backward3a_T_Ph\( (P,h) \)

Parameters

- \( P (float) \) – Pressure, [MPa]
- \( h (float) \) – Specific enthalpy, [kJ/kg]

Returns \( T \) – Temperature, [K]

Return type float
References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions T(p,h), v(p,h) and T(p,s), v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 2

Examples

```python
>>> _Backward3a_T_Ph(20,1700)
629.3083892
>>> _Backward3a_T_Ph(100,2100)
733.6163014
```

```python
iapws.iapws97._Backward3b_T_Ph(P, h)

Parameters

- P (float) – Pressure, [MPa]
- h (float) – Specific enthalpy, [kJ/kg]

Returns T – Temperature, [K]

Return type float

References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions T(p,h), v(p,h) and T(p,s), v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 3

Examples

```python
>>> _Backward3b_T_Ph(20,2500)
641.8418053
>>> _Backward3b_T_Ph(100,2700)
842.0460876
```

iapws.iapws97._Backward3_T_Ph(P, h)

Parameters

- P (float) – Pressure, [MPa]
- h (float) – Specific enthalpy, [kJ/kg]

Returns T – Temperature, [K]

Return type float

iapws.iapws97._Backward3a_v_Ps(P, s)

Parameters

- P (float) – Pressure, [MPa]
- s (float) – Specific entropy, [kJ/kgK]

Returns v – Specific volume, [m³/kg]

5.1. iapws package
References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions $T(p,h)$, $v(p,h)$ and $T(p,s)$, $v(p,s)$ for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 8

Examples

```python
>>> _Backward3a_v_Ps(20, 3.8)
0.001733791463
>>> _Backward3a_v_Ps(100, 4)
0.001555893131
```

```python
iapws.iapws97._Backward3b_v_Ps(P, s)

Parameters

- $P$ (float) – Pressure, [MPa]
- $s$ (float) – Specific entropy, [kJ/kgK]

Returns

$v$ – Specific volume, [m$^3$/kg]

Return type float

References

IAPWS, Revised Supplementary Release on Backward Equations for the Functions $T(p,h)$, $v(p,h)$ and $T(p,s)$, $v(p,s)$ for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 9

Examples

```python
>>> _Backward3b_v_Ps(20, 5)
0.006262101987
>>> _Backward3b_v_Ps(100, 5)
0.002449610757
```

```python
iapws.iapws97._Backward3_v_Ps(P, s)

Parameters

- $P$ (float) – Pressure, [MPa]
- $s$ (float) – Specific entropy, [kJ/kgK]

Returns

$v$ – Specific volume, [m$^3$/kg]

Return type float

iapws.iapws97._Backward3a_T_Ps(P, s)

Parameters

- $P$ (float) – Pressure, [MPa]
```
• **s (float)** – Specific entropy, [kJ/kgK]

**Returns** T – Temperature, [K]

**Return type** float

**References**

IAPWS, Revised Supplementary Release on Backward Equations for the Functions T(p,h), v(p,h) and T(p,s), v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 6

**Examples**

```python
>>> _Backward3a_T_Ps(20,3.8)
628.2959869
>>> _Backward3a_T_Ps(100,4)
705.6880237
```

iapws.iapws97._Backward3b_T_Ps (P, s)

**Parameters**

• **P (float)** – Pressure, [MPa]

• **s (float)** – Specific entropy, [kJ/kgK]

**Returns** T – Temperature, [K]

**Return type** float

**References**

IAPWS, Revised Supplementary Release on Backward Equations for the Functions T(p,h), v(p,h) and T(p,s), v(p,s) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-Tv%28ph,ps%293-2014.pdf, Eq 7

**Examples**

```python
>>> _Backward3b_T_Ps(20,5)
640.1176443
>>> _Backward3b_T_Ps(100,5)
847.4332825
```

iapws.iapws97._Backward3_T_Ps (P, s)

**Parameters**

• **P (float)** – Pressure, [MPa]

• **s (float)** – Specific entropy, [kJ/kgK]

**Returns** T – Temperature, [K]

**Return type** float

iapws.iapws97._Backward3a_P_hs (h, s)

5.1. iapws package
Parameters

- \( h \) (float) – Specific enthalpy, [kJ/kg]
- \( s \) (float) – Specific entropy, [kJ/kgK]

Returns \( P \) – Pressure, [MPa]

Return type float

References

IAPWS, Revised Supplementary Release on Backward Equations \( p(h,s) \) for Region 3, Equations as a Function of \( h \) and \( s \) for the Region Boundaries, and an Equation \( T_{sat}(h,s) \) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 1

Examples

```python
>>> _Backward3a_P_hs(1700,3.8)
25.55703246
>>> _Backward3a_P_hs(2000,4.2)
45.40873468
>>> _Backward3a_P_hs(2100,4.3)
60.78123340
```

iapws.iapws97._Backward3b_P_hs\( (h,s) \)

Parameters

- \( h \) (float) – Specific enthalpy, [kJ/kg]
- \( s \) (float) – Specific entropy, [kJ/kgK]

Returns \( P \) – Pressure, [MPa]

Return type float

References

IAPWS, Revised Supplementary Release on Backward Equations \( p(h,s) \) for Region 3, Equations as a Function of \( h \) and \( s \) for the Region Boundaries, and an Equation \( T_{sat}(h,s) \) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 1

Examples

```python
>>> _Backward3b_P_hs(2400,4.7)
63.63924887
>>> _Backward3b_P_hs(2600,5.1)
34.34999263
>>> _Backward3b_P_hs(2700,5.0)
88.39043281
```

iapws.iapws97._Backward3_P_hs\( (h,s) \)

Parameters
• \( h(\text{float}) \) – Specific enthalpy, [kJ/kg]
• \( s(\text{float}) \) – Specific entropy, [kJ/kgK]

Returns \( P \) – Pressure, [MPa]
Return type float

iapws.iapws97._Backward3_sat_v_P(\( P, x \))

Parameters
• \( T(\text{float}) \) – Temperature, [K]
• \( P(\text{float}) \) – Pressure, [MPa]
• \( x(\text{integer}) \) – Vapor quality, [-]

Returns \( v \) – Specific volume, [m\(^3\)/kg]
Return type float

Notes
The vapor quality \( x \) can be 0 (saturated liquid) or 1 (saturated vapour)

iapws.iapws97._Backward3_v_PT(\( P, T \))

Parameters
• \( T(\text{float}) \) – Temperature, [K]
• \( P(\text{float}) \) – Pressure, [MPa]

Returns \( v \) – Specific volume, [m\(^3\)/kg]
Return type float

References
IAPWS, Revised Supplementary Release on Backward Equations for Specific Volume as a Function of Pressure and Temperature \( v(p,T) \) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-VPT3-2016.pdf, Table 2 and 10

iapws.iapws97._Backward3x_v_PT(\( P, T \))

Parameters
• \( T(\text{float}) \) – Temperature, [K]
• \( P(\text{float}) \) – Pressure, [MPa]
• \( x(\text{char}) \) – Region 3 subregion code

Returns \( v \) – Specific volume, [m\(^3\)/kg]
Return type float

References

5.1. iapws package
Examples

```python
>>> _Backward3x_v_PT(630, 50, "a")
0.001470853100
>>> _Backward3x_v_PT(670, 80, "a")
0.001503831359
>>> _Backward3x_v_PT(710, 50, "b")
0.002204728587
>>> _Backward3x_v_PT(750, 80, "b")
0.001973692940
>>> _Backward3x_v_PT(630, 20, "c")
0.001819560617
>>> _Backward3x_v_PT(650, 30, "c")
0.001761696406
>>> _Backward3x_v_PT(656, 26, "d")
0.002245587720
>>> _Backward3x_v_PT(670, 30, "d")
0.002506897702
>>> _Backward3x_v_PT(661, 26, "e")
0.002970225962
>>> _Backward3x_v_PT(675, 30, "e")
0.003004627086
>>> _Backward3x_v_PT(671, 26, "f")
0.005019029401
>>> _Backward3x_v_PT(690, 30, "f")
0.004656470142
>>> _Backward3x_v_PT(649, 23.6, "g")
0.002163198378
>>> _Backward3x_v_PT(650, 24, "g")
0.002166044161
>>> _Backward3x_v_PT(652, 23.6, "h")
0.002651081407
>>> _Backward3x_v_PT(654, 24, "h")
0.002967802335
>>> _Backward3x_v_PT(653, 23.6, "i")
0.003273916816
>>> _Backward3x_v_PT(655, 24, "i")
0.003550329864
>>> _Backward3x_v_PT(655, 23.5, "j")
0.004545001142
>>> _Backward3x_v_PT(660, 24, "j")
0.005100267704
>>> _Backward3x_v_PT(660, 23, "k")
0.006109525997
>>> _Backward3x_v_PT(670, 24, "k")
0.006427325645
>>> _Backward3x_v_PT(646, 22.6, "l")
0.002117860851
>>> _Backward3x_v_PT(646, 23, "l")
0.002062374674
>>> _Backward3x_v_PT(648.6, 22.6, "m")
0.002533063780
>>> _Backward3x_v_PT(649.3, 22.8, "m")
0.002572971781
>>> _Backward3x_v_PT(649, 22.6, "n")
0.002923432711
>>> _Backward3x_v_PT(649.7, 22.8, "n")
```

(continues on next page)
iapws Documentation, Release 1.1.3

0.002913311494
>>> _Backward3x_v_PT(649.1, 22.6, "o")
0.003131208996
>>> _Backward3x_v_PT(649.9, 22.8, "o")
0.003221160278
>>> _Backward3x_v_PT(649.4, 22.6, "p")
0.003715596186
>>> _Backward3x_v_PT(650.2, 22.8, "p")
0.003664754790
>>> _Backward3x_v_PT(640, 21.1, "q")
0.001970999272
>>> _Backward3x_v_PT(643, 21.8, "q")
0.002043919161
>>> _Backward3x_v_PT(644, 21.1, "r")
0.005251009921
>>> _Backward3x_v_PT(648, 21.8, "r")
0.005256844741
>>> _Backward3x_v_PT(635, 19.1, "s")
0.001933292079
>>> _Backward3x_v_PT(638, 20, "s")
0.00198537227
>>> _Backward3x_v_PT(626, 17, "t")
0.008483262001
>>> _Backward3x_v_PT(640, 20, "t")
0.006227528101
>>> _Backward3x_v_PT(644.6, 21.5, "u")
0.002268366647
>>> _Backward3x_v_PT(646.1, 22, "u")
0.002296350553
>>> _Backward3x_v_PT(648.6, 22.5, "v")
0.002833273260
>>> _Backward3x_v_PT(647.9, 22.3, "v")
0.002811424405
>>> _Backward3x_v_PT(647.5, 22.15, "w")
0.003694032281
>>> _Backward3x_v_PT(648.1, 22.3, "w")
0.003622226305
>>> _Backward3x_v_PT(648, 22.11, "x")
0.004528072649
>>> _Backward3x_v_PT(649, 22.3, "x")
0.004556905799
>>> _Backward3x_v_PT(646.84, 22, "y")
0.002698354719
>>> _Backward3x_v_PT(647.05, 22.064, "y")
0.002717655648
>>> _Backward3x_v_PT(646.89, 22, "z")
0.003798732962
>>> _Backward3x_v_PT(647.15, 22.064, "z")
0.003701940009

iapws.iapws97._Region4(P, x)

Basic equation for region 4

Parameters

- \( P \) (float) – Pressure, [MPa]
- \( x \) (float) – Vapor quality, [-]

5.1. iapws package

89
Returns

prop –

Dict with calculated properties. The available properties are:

- T: Saturated temperature, [K]
- P: Saturated pressure, [MPa]
- x: Vapor quality, [-]
- v: Specific volume, [m$^3$/kg]
- h: Specific enthalpy, [kJ/kg]
- s: Specific entropy, [kJ/kgK]

Return type  dict

iapws.iapws97._Backward4_T_hs(h,s)

Parameters

- h (float) – Specific enthalpy, [kJ/kg]
- s (float) – Specific entropy, [kJ/kgK]

Returns  T – Temperature, [K]

Return type  float

References

IAPWS, Revised Supplementary Release on Backward Equations p(h,s) for Region 3, Equations as a Function of h and s for the Region Boundaries, and an Equation Tsat(h,s) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, http://www.iapws.org/relguide/Supp-phs3-2014.pdf. Eq 9

Examples

```py
>>> _Backward4_T_hs(1800,5.3)
346.8475498
>>> _Backward4_T_hs(2400,6.0)
425.1373305
>>> _Backward4_T_hs(2500,5.5)
522.5579013
```

iapws.iapws97._Region5(T,P)

Basic equation for region 5

Parameters

- T (float) – Temperature, [K]
- P (float) – Pressure, [MPa]

Returns

prop –

Dict with calculated properties. The available properties are:

- v: Specific volume, [m$^3$/kg]
• h: Specific enthalpy, [kJ/kg]
• s: Specific entropy, [kJ/kgK]
• cp: Specific isobaric heat capacity, [kJ/kgK]
• cv: Specific isochoric heat capacity, [kJ/kgK]
• w: Speed of sound, [m/s]
• alfav: Cubic expansion coefficient, [1/K]
• kt: Isothermal compressibility, [1/MPa]

Return type  dict

References


Examples

```
>>> _Region5(1500,0.5)['v']
1.38455090
>>> _Region5(1500,0.5)['h']
52.19.76855
>>> _Region5(1500,0.5)['h']-500*_Region5(1500,0.5)['v']
45.27.49310
>>> _Region5(1500,30)['s']
7.72970133
>>> _Region5(1500,30)['cp']
2.72724317
>>> _Region5(1500,30)['cv']
2.19274829
>>> _Region5(2000,30)['w']
1067.36948
>>> _Region5(2000,30)['alfav']
0.000508830641
>>> _Region5(2000,30)['kt']
0.0329193892
```

iapws.iapws97.Region5_cp0(Tr, Pr)

Ideal properties for Region 5

Parameters

• Tr (float) – Reduced temperature, [-]
• Pr (float) – Reduced pressure, [-]

Returns

prop –

Array with ideal Gibbs energy partial derivatives:

• g: Ideal Specific Gibbs energy, [kJ/kg]
• gp: [g/P]T
• gpp: [g/P^2]T
iapws Documentation, Release 1.1.3

- gt: \([g/T]P\)
- gtt: \([g^2/T^2]P\)
- gpt: \([g/TP]\)

**Return type**  array

**References**


```python
iapws.iapws97._Bound_TP(T, P)
```

Region definition for input T and P

**Parameters**

- \(T (float)\) – Temperature, [K]
- \(P (float)\) – Pressure, [MPa]

**Returns region** – IAPWS-97 region code

**Return type**  float

**References**

Wagner, W; Kretzschmar, H-J: International Steam Tables: Properties of Water and Steam Based on the Industrial Formulation IAPWS-IF97; Springer, 2008; doi: 10.1007/978-3-540-74234-0. Fig. 2.3

```python
iapws.iapws97._Bound_Ph(P, h)
```

Region definition for input P \& h

**Parameters**

- \(P (float)\) – Pressure, [MPa]
- \(h (float)\) – Specific enthalpy, [kJ/kg]

**Returns region** – IAPWS-97 region code

**Return type**  float

**References**

Wagner, W; Kretzschmar, H-J: International Steam Tables: Properties of Water and Steam Based on the Industrial Formulation IAPWS-IF97; Springer, 2008; doi: 10.1007/978-3-540-74234-0. Fig. 2.5

```python
iapws.iapws97._Bound_Ps(P, s)
```

Region definition for input P and s

**Parameters**

- \(P (float)\) – Pressure, [MPa]
- \(s (float)\) – Specific entropy, [kJ/kgK]

**Returns region** – IAPWS-97 region code

**Return type**  float
References

Wagner, W; Kretzschmar, H-J: International Steam Tables: Properties of Water and Steam Based on the Industrial Formulation IAPWS-IF97; Springer, 2008; doi: 10.1007/978-3-540-74234-0. Fig. 2.9

iapws.iapws97._Bound_hs(h, s)
Region definition for input h and s

Parameters

• h (float) – Specific enthalpy, [kJ/kg]
• s (float) – Specific entropy, [kJ/kgK]

Returns region – IAPWS-97 region code

Return type float

References

Wagner, W; Kretzschmar, H-J: International Steam Tables: Properties of Water and Steam Based on the Industrial Formulation IAPWS-IF97; Springer, 2008; doi: 10.1007/978-3-540-74234-0. Fig. 2.14

iapws.iapws97.prop0(T, P)
Ideal gas properties

Parameters

• T (float) – Temperature, [K]
• P (float) – Pressure, [MPa]

Returns

prop –
Dict with calculated properties. The available properties are:
• v: Specific volume, [m³/kg]
• h: Specific enthalpy, [kJ/kg]
• s: Specific entropy, [kJ/kgK]
• cp: Specific isobaric heat capacity, [kJ/kgK]
• cv: Specific isochoric heat capacity, [kJ/kgK]
• w: Speed of sound, [m/s]
• alfav: Cubic expansion coefficient, [1/K]
• kt: Isothermal compressibility, [1/MPa]

Return type dict

class iapws.iapws97.IAPWS97(**kwargs)
Class to model a state of liquid water or steam with the IAPWS-IF97

Parameters

• T (float) – Temperature, [K]
• P (float) – Pressure, [MPa]
• h (float) – Specific enthalpy, [kJ/kg]
• **s** (*float*) – Specific entropy, [kJ/kgK]
• **x** (*float*) – Vapor quality, [-]
• **l** (*float, optional*) – Wavelength of light, for refractive index, [nm]

**Notes**

Definitions options:

• T, P: Not valid for two-phases region
• P, h
• P, s
• h, s
• T, x: Only for two-phases region
• P, x: Only for two-phases region

**Returns**

`prop` –

The calculated instance has the following properties:

• P: Pressure, [MPa]
• T: Temperature, [K]
• g: Specific Gibbs free energy, [kJ/kg]
• a: Specific Helmholtz free energy, [kJ/kg]
• v: Specific volume, [m³/kg]
• rho: Density, [kg/m³]
• h: Specific enthalpy, [kJ/kg]
• u: Specific internal energy, [kJ/kg]
• s: Specific entropy, [kJ/kg·K]
• cp: Specific isobaric heat capacity, [kJ/kg·K]
• cv: Specific isochoric heat capacity, [kJ/kg·K]
• Z: Compression factor, [-]
• fi: Fugacity coefficient, [-]
• f: Fugacity, [MPa]
• gamma: Isoentropic exponent, [-]
• alfav: Isobaric cubic expansion coefficient, [1/K]
• xkappa: Isothermal compressibility, [1/MPa]
• kappas: Adiabatic compresibility, [1/MPa]
• alfap: Relative pressure coefficient, [1/K]
• betap: Isothermal stress coefficient, [kg/m³]
• joule: Joule-Thomson coefficient, [K/MPa]
• deltat: Isothermal throttling coefficient, [kJ/kg·MPa]
• region: Region
• v0: Ideal specific volume, [m³/kg]
• u0: Ideal specific internal energy, [kJ/kg]
• h0: Ideal specific enthalpy, [kJ/kg]
• s0: Ideal specific entropy, [kJ/kg·K]
• a0: Ideal specific Helmholtz free energy, [kJ/kg]
• g0: Ideal specific Gibbs free energy, [kJ/kg]
• cp0: Ideal specific isobaric heat capacity, [kJ/kg·K]
• cv0: Ideal specific isochoric heat capacity [kJ/kg·K]
• w0: Ideal speed of sound, [m/s]
• gamma0: Ideal isoentropic exponent, [-]
• w: Speed of sound, [m/s]
• mu: Dynamic viscosity, [Pa·s]
• nu: Kinematic viscosity, [m²/s]
• k: Thermal conductivity, [W/m·K]
• alfa: Thermal diffusivity, [m²/s]
• sigma: Surface tension, [N/m]
• epsilon: Dielectric constant, [-]
• n: Refractive index, [-]
• Prandt: Prandtl number, [-]
• Pr: Reduced Pressure, [-]
• Tr: Reduced Temperature, [-]
• Hvap: Vaporization heat, [kJ/kg]
• Svp: Vaporization entropy, [kJ/kg·K]

Return type  dict

Examples

```python
>>> water=IAPWS97(T=170+273.15, x=0.5)
>>> water.Liquid.cp, water.Vapor.cp, water.Liquid.w, water.Vapor.w
4.3695 2.5985 1418.3 498.78
```

```python
>>> water=IAPWS97(T=325+273.15, x=0.5)
>>> water.P, water.Liquid.v, water.Vapor.v, water.Liquid.h, water.Vapor.h
12.0505 0.00152830 0.0141887 1493.37 2684.48
```

```python
>>> water=IAPWS97(T=50+273.15, P=0.0006112127)
>>> water.cp0, water.cv0, water.h0, water.s0, water.w0
1.8714 1.4098 2594.66 9.471 444.93
```
Attributes

*calculable* Check if class is calculable by its kwargs

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><strong>call</strong></em>(self, <strong>kwargs)</strong></td>
<td>Call self as a function.</td>
</tr>
<tr>
<td><em>derivative</em>(self, z, x, y, fase)</td>
<td>Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a</td>
</tr>
</tbody>
</table>

```
calculo
fill
```

status = 0
msg = 'Unknown variables'
kwargs = {'P': 0.0, 'T': 0.0, 'h': None, 'l': 0.5893, 's': None, 'v': 0.0, 'x': None}
calculo
    Check if class is calculable by its kwargs
calculo(self)
fill(self, fase, estado)
derivative(self, z, x, y, fase)
    Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a

class iapws.iapws97.IAPWS97_PT(P, T)
    Derivated class for direct P and T input
Attributes

*calculable* Check if class is calculable by its kwargs

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><strong>call</strong></em>(self, <strong>kwargs)</strong></td>
<td>Call self as a function.</td>
</tr>
<tr>
<td><em>derivative</em>(self, z, x, y, fase)</td>
<td>Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a</td>
</tr>
</tbody>
</table>

```
calculo
fill
```

class iapws.iapws97.IAPWS97_Ph(P, h)
    Derivated class for direct P and h input
Attributes

*calculable* Check if class is calculable by its kwargs
Methods

__call__(self, **kwargs) Call self as a function.
derivative(self, z, x, y, fase) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a

class iapws.iapws97.IAPWS97_Ps(P, s)
    Derivated class for direct P and s input

Attributes

    calculable Check if class is calculable by its kwargs

Methods

__call__(self, **kwargs) Call self as a function.
derivative(self, z, x, y, fase) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a

class iapws.iapws97.IAPWS97_Px(P, x)
    Derivated class for direct P and x input

Attributes

    calculable Check if class is calculable by its kwargs

Methods

__call__(self, **kwargs) Call self as a function.
derivative(self, z, x, y, fase) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a

class iapws.iapws97.IAPWS97_Tx(T, x)
    Derivated class for direct T and x input

Attributes

    calculable Check if class is calculable by its kwargs

5.1. iapws package
**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__</code> (self, **kwargs)</td>
<td>Call self as a function.</td>
</tr>
<tr>
<td><code>derivative</code> (self, z, x, y, fase)</td>
<td>Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a</td>
</tr>
</tbody>
</table>

**class iapws.iapws97.IAPWS97(**kwargs)**

Class to model a state of liquid water or steam with the IAPWS-IF97

**Parameters**

- **T** (*float*) – Temperature, [K]
- **P** (*float*) – Pressure, [MPa]
- **h** (*float*) – Specific enthalpy, [kJ/kg]
- **s** (*float*) – Specific entropy, [kJ/kgK]
- **x** (*float*) – Vapor quality, [-]
- **l** (*float, optional*) – Wavelength of light, for refractive index, [nm]

**Notes**

Definitions options:

- T, P: Not valid for two-phases region
- P, h
- P, s
- h, s
- T, x: Only for two-phases region
- P, x: Only for two-phases region

**Returns**

prop –

The calculated instance has the following properties:

- **P**: Pressure, [MPa]
- **T**: Temperature, [K]
- **g**: Specific Gibbs free energy, [kJ/kg]
- **a**: Specific Helmholtz free energy, [kJ/kg]
- **v**: Specific volume, [m³/kg]
- **rho**: Density, [kg/m³]
- **h**: Specific enthalpy, [kJ/kg]
• u: Specific internal energy, [kJ/kg]
• s: Specific entropy, [kJ/kg·K]
• cp: Specific isobaric heat capacity, [kJ/kg·K]
• cv: Specific isochoric heat capacity, [kJ/kg·K]
• Z: Compression factor, [-]
• fi: Fugacity coefficient, [-]
• f: Fugacity, [MPa]
• gamma: Isoentropic exponent, [-]
• alfav: Isobaric cubic expansion coefficient, [1/K]
• xkappa: Isothermal compressibility, [1/MPa]
• kappas: Adiabatic compressibility, [1/MPa]
• alfap: Relative pressure coefficient, [1/K]
• betap: Isothermal stress coefficient, [kg/m³]
• joule: Joule-Thomson coefficient, [K/MPa]
• deltat: Isothermal throttling coefficient, [kJ/kg·MPa]
• region: Region
• v0: Ideal specific volume, [m³/kg]
• u0: Ideal specific internal energy, [kJ/kg]
• h0: Ideal specific enthalpy, [kJ/kg]
• s0: Ideal specific entropy, [kJ/kg·K]
• a0: Ideal specific Helmholtz free energy, [kJ/kg]
• g0: Ideal specific Gibbs free energy, [kJ/kg]
• cp0: Ideal specific isobaric heat capacity, [kJ/kg·K]
• cv0: Ideal specific isochoric heat capacity, [kJ/kg·K]
• w0: Ideal speed of sound, [m/s]
• gamma0: Ideal isoentropic exponent, [-]
• w: Speed of sound, [m/s]
• mu: Dynamic viscosity, [Pa·s]
• nu: Kinematic viscosity, [m²/s]
• k: Thermal conductivity, [W/m-K]
• alfa: Thermal diffusivity, [m²/s]
• sigma: Surface tension, [N/m]
• epsilon: Dielectric constant, [-]
• n: Refractive index, [-]
• Prandt: Prandtl number, [-]
• Pr: Reduced Pressure, [-]
iapws Documentation, Release 1.1.3

- Tr: Reduced Temperature, [-]
- Hvap: Vaporization heat, [kJ/kg]
- Svap: Vaporization entropy, [kJ/kg·K]

Return type: dict

Examples

```python
>>> water = IAPWS97(T=170+273.15, x=0.5)
>>> water.Liquid.cp, water.Vapor.cp, water.Liquid.w, water.Vapor.w
(4.3695, 2.5985, 1418.3, 498.78)
```

```python
>>> water = IAPWS97(T=325+273.15, x=0.5)
>>> water.P, water.Liquid.v, water.Vapor.v, water.Liquid.h, water.Vapor.h
(12.0505, 0.00152830, 0.0141887, 1493.37, 2684.48)
```

```python
>>> water = IAPWS97(T=50+273.15, P=0.0006112127)
>>> water.cp0, water.cv0, water.h0, water.s0, water.w0
(1.8714, 1.4098, 2594.66, 9.471, 444.93)
```

Methods

```python
def derivative(self, z, x, y, fase):
    Wrapper derivative for custom derived properties
    where x, y, z can be: P, T, v, u, h, s, g, a
```

5.1.5 iapws.iapws08 module

IAPWS standard for Seawater IAPWS08 and related functionality. The module include:

**SeaWater**: Global module class with all the functionality integrated

Other functionality:

- `_Tb()`: Boiling temperature of seawater
- `_Tf()`: Freezing temperature of seawater
- `_Triple()`: Triple point properties of seawater
- `_OsmoticPressure()`: Osmotic pressure of seawater
- `_ThCond_SeaWater()`: Thermal conductivity of seawater
- `_solNa2SO4()`: Solubility of sodium sulfate in aqueous mixtures of sodium chloride and sulfuric acid
- `_critNaCl()`: Critical locus of aqueous solutions of sodium chloride

```python
class iapws.iapws08.SeaWater(**kwargs):
    Class to model seawater with standard IAPWS-08
```

Parameters
• **T** *(float)* – Temperature, [K]
• **P** *(float)* – Pressure, [MPa]
• **S** *(float)* – Salinity, [kg/kg]
• **fast** *(bool, default False)* – Use the Supplementary release SR7-09 to speed up the calculation
• **IF97** *(bool, default False)* – Use the Advisory Note No. 5 with industrial formulation

**Returns**

• **rho** *(float)* – Density, [kg/m\(^3\)]
• **v** *(float)* – Specific volume, [m\(^3\)/kg]
• **h** *(float)* – Specific enthalpy, [kJ/kg]
• **s** *(float)* – Specific entropy, [kJ/kg·K]
• **u** *(float)* – Specific internal energy, [kJ/kg]
• **g** *(float)* – Specific Gibbs free energy, [kJ/kg]
• **a** *(float)* – Specific Helmholtz free energy, [kJ/kg]
• **cp** *(float)* – Specific isobaric heat capacity, [kJ/kg·K]
• **cv** *(float)* – Specific isochoric heat capacity, [kJ/kg·K]
• **gt** *(float)* – Derivative Gibbs energy with temperature, [kJ/kg·K]
• **gp** *(float)* – Derivative Gibbs energy with pressure, [m\(^3\)/kg]
• **gtt** *(float)* – Derivative Gibbs energy with temperature square, [kJ/kg·K\(^2\)]
• **gtp** *(float)* – Derivative Gibbs energy with pressure and temperature, [m\(^3\)/kg·K]
• **gpp** *(float)* – Derivative Gibbs energy with temperature square, [m\(^3\)/kg·MPa]
• **gs** *(float)* – Derivative Gibbs energy with salinity, [kJ/kg]
• **gsp** *(float)* – Derivative Gibbs energy with salinity and pressure, [m\(^3\)/kg]
• **alfav** *(float)* – Thermal expansion coefficient, [1/K]
• **betas** *(float)* – Isentropic temperature-pressure coefficient, [K/MPa]
• **xkappa** *(float)* – Isothermal compressibility, [1/MPa]
• **ks** *(float)* – Isentropic compressibility, [1/MPa]
• **w** *(float)* – Sound Speed, [m/s]
• **m** *(float)* – Molality of seawater, [mol/kg]
• **mu** *(float)* – Relative chemical potential, [kJ/kg]
• **muw** *(float)* – Chemical potential of H2O, [kJ/kg]
• **musr** *(float)* – Chemical potential of sea salt, [kJ/kg]
• **osm** *(float)* – Osmotic coefficient, [-]
• **haline** *(float)* – Haline contraction coefficient, [kg/kg]
Notes

Warning if input isn’t in limit:

- 261 ≤ T ≤ 353
- 0 < P ≤ 100
- 0 ≤ S ≤ 0.12

References


Examples

```python
>>> salt = iapws.SeaWater(T=300, P=1, S=0.04)
>>> salt.rho
1026.7785717245113
>>> salt.gs
88.56221805501536
>>> salt.haline
0.7311487666026304
```

Methods

```python
__call__(self, **kwargs) Make instance callable to can add input parameter one to one

calculo(self) Calculate procedure

derivative(self, z, x, y) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a
```

status = 0

msg = 'Undefined'

kwargs = {'IF97': True, 'P': 0.0, 'S': None, 'T': 0.0, 'fast': False}

calculo(self)
    Calculate procedure
derivative(self, z, x, y)
    Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, u, h, s, g, a
classmethod _water($T, P$)
Get properties of pure water, Table4 pag 8

classmethod _waterIF97($T, P$)

classmethod _waterSupp($T, P$)
Get properties of pure water using the supplementary release SR7-09, Table4 pag 6

classmethod _saline($T, P, S$)
Eq 4

iapws.iapws08._Tb($P, S$)
Procedure to calculate the boiling temperature of seawater

Parameters

- $P$ (float) – Pressure, [MPa]
- $S$ (float) – Salinity, [kg/kg]

Returns $Tb$ – Boiling temperature, [K]
Return type float

References

IAPWS, Advisory Note No. 5: Industrial Calculation of the Thermodynamic Properties of Seawater, http://www.iapws.org/relguide/Advise5.html, Eq 7

iapws.iapws08._Tf($P, S$)
Procedure to calculate the freezing temperature of seawater

Parameters

- $P$ (float) – Pressure, [MPa]
- $S$ (float) – Salinity, [kg/kg]

Returns $Tf$ – Freezing temperature, [K]
Return type float

References

IAPWS, Advisory Note No. 5: Industrial Calculation of the Thermodynamic Properties of Seawater, http://www.iapws.org/relguide/Advise5.html, Eq 12

iapws.iapws08._Triple($S$)
Procedure to calculate the triple point pressure and temperature for seawater

Parameters $S$ (float) – Salinity, [kg/kg]

Returns

prop –
Dictionary with the triple point properties:
- $Tt$: Triple point temperature, [K]
- $Pt$: Triple point pressure, [MPa]

Return type dict

5.1. iapws package
iapws Documentation, Release 1.1.3

References

IAPWS, Advisory Note No. 5: Industrial Calculation of the Thermodynamic Properties of Seawater, http://www.iapws.org/relguide/Advise5.html, Eq 7

iapws.iapws08._OsmoticPressure(T, P, S)
Procedure to calculate the osmotic pressure of seawater

Parameters

• T(float) – Temperature, [K]
• P(float) – Pressure, [MPa]
• S(float) – Salinity, [kg/kg]

Returns Posm – Osmotic pressure, [MPa]
Return type float

References

IAPWS, Advisory Note No. 5: Industrial Calculation of the Thermodynamic Properties of Seawater, http://www.iapws.org/relguide/Advise5.html, Eq 15

iapws.iapws08._ThCond_SeaWater(T, P, S)
Equation for the thermal conductivity of seawater

Parameters

• T(float) – Temperature, [K]
• P(float) – Pressure, [MPa]
• S(float) – Salinity, [kg/kg]

Returns k – Thermal conductivity excess relative to that of the pure water, [W/mK]
Return type float

Notes

Raise NotImplementedWarning if input isn’t in limit:

• 273.15 ≤ T ≤ 523.15
• 0 ≤ P ≤ 140
• 0 ≤ S ≤ 0.17

Examples

```python
>>> _ThCond_SeaWater(293.15, 0.1, 0.035)
-0.00418604
```
Equation for the solubility of sodium sulfate in aqueous mixtures of sodium chloride and sulfuric acid

Parameters

- \( T \) (float) – Temperature, [K]
- \( mH2SO4 \) (float) – Molality of sulfuric acid, [mol/kg(water)]
- \( mNaCl \) (float) – Molality of sodium chloride, [mol/kg(water)]

Returns \( S \) – Molal solubility of sodium sulfate, [mol/kg(water)]

Return type float

Notes

Raise \texttt{NotImplementedError} if input isn’t in limit:

- 523.15 \( T \) 623.15
- 0 \( mH2SO4 \) 0.75
- 0 \( mNaCl \) 2.25

Examples

```python
>>> _solNa2SO4(523.15, 0.25, 0.75)
2.68
```

References

IAPWS, Solubility of Sodium Sulfate in Aqueous Mixtures of Sodium Chloride and Sulfuric Acid from Water to Concentrated Solutions, http://www.iapws.org/relguide/na2so4.pdf
Notes

Raise `NotImplementedError` if input isn’t in limit:

- 0 x 0.12

Examples

```python
>>> _critNaCl(0.1)
975.571016
```

References


5.1.6 iapws.ammonia module

Module with Ammonia-water mixture properties and related properties. The module include:

- `NH3`: Multiparameter equation of state for ammonia
- `H2ONH3`: Thermodynamic properties of ammonia-water mixtures
- `Tr`: Triple point of ammonia-water mixtures

```python
class iapws.ammonia.NH3(**kwargs)
```

Multiparameter equation of state for ammonia for internal procedures, see MEoS base class

Parameters

- `T (float)` – Temperature, [K]
- `P (float)` – Pressure, [MPa]
- `rho (float)` – Density, [kg/m³]
- `v (float)` – Specific volume, [m³/kg]
- `h (float)` – Specific enthalpy, [kJ/kg]
- `s (float)` – Specific entropy, [kJ/kgK]
- `u (float)` – Specific internal energy, [kJ/kg]
- `x (float)` – Vapor quality, [-]
- `l (float, optional)` – Wavelength of light, for refractive index, [nm]
- `rho0 (float, optional)` – Initial value of density, to improve iteration, [kg/m³]
- `T0 (float, optional)` – Initial value of temperature, to improve iteration, [K]
- `x0` (Initial value of vapor quality, necessary in bad input pair definition) – where there are two valid solution (T-h, T-s)
Notes

- It needs two incoming properties of T, P, rho, h, s, u.
- \( v \) as an alternate input parameter to rho
- T-x, P-x, preferred input pair to specified a point in two phases region

The calculated instance has the following properties:

- P: Pressure, [MPa]
- T: Temperature, [K]
- x: Vapor quality, [-]
- g: Specific Gibbs free energy, [kJ/kg]
- a: Specific Helmholtz free energy, [kJ/kg]
- v: Specific volume, [m\(^3\)/kg]
- r: Density, [kg/m\(^3\)]
- h: Specific enthalpy, [kJ/kg]
- u: Specific internal energy, [kJ/kg]
- s: Specific entropy, [kJ/kg·K]
- cp: Specific isobaric heat capacity, [kJ/kg·K]
- cv: Specific isochoric heat capacity, [kJ/kg·K]
- cp_cv: Heat capacity ratio, [-]
- Z: Compression factor, [-]
- fi: Fugacity coefficient, [-]
- f: Fugacity, [MPa]
- gamma: Isoentropic exponent, [-]
- alfav: Isobaric cubic expansion coefficient, [1/K]
- kappa: Isothermal compressibility, [1/MPa]
- kappas: Adiabatic compressibility, [1/MPa]
- alfap: Relative pressure coefficient, [1/K]
- betap: Isothermal stress coefficient, [kg/m\(^3\)]
- joule: Joule-Thomson coefficient, [K/MPa]
- betas: Isoentropic temperature-pressure coefficient, [-]
- Gruneisen: Gruneisen parameter, [-]
- virialB: Second virial coefficient, [m\(^3\)/kg]
- virialC: Third virial coefficient, [m\(^6\)/kg\(^2\)]
- dpdT_rho: Derivatives, dp/dT at constant rho, [MPa/K]
- dpdrho_T: Derivatives, dp/drho at constant T, [MPa·m\(^3\)/kg]
- drhodT_P: Derivatives, drho/dT at constant P, [kg/m\(^3\)·K]
• drhodT: Derivatives, drho/dP at constant T, [kg/m$^3$·MPa]
• dhdT_rho: Derivatives, dh/dT at constant rho, [kJ/kg·K]
• dhdT_P: Isothermal throttling coefficient, [kJ/kg·MPa]
• dhdT_P: Derivatives, dh/dT at constant P, [kJ/kg·K]
• dhdrho_T: Derivatives, dh/drho at constant T, [kJ·m$^3$/kg$^2$]
• dhdrho_P: Derivatives, dh/drho at constant P, [kJ·m$^3$/kg$^2$]
• dhdp_rho: Derivatives, dh/dP at constant rho, [kJ/kg·MPa]
• kt: Isothermal Expansion Coefficient, [-]
• ks: Adiabatic Compressibility, [1/MPa]
• Ks: Adiabatic bulk modulus, [MPa]
• Kt: Isothermal bulk modulus, [MPa]
• v0: Ideal specific volume, [m$^3$/kg]
• rho0: Ideal gas density, [kg/m$^3$]
• u0: Ideal specific internal energy, [kJ/kg]
• h0: Ideal specific enthalpy, [kJ/kg]
• s0: Ideal specific entropy, [kJ/kg·K]
• a0: Ideal specific Helmholtz free energy, [kJ/kg]
• g0: Ideal specific Gibbs free energy, [kJ/kg]
• cp0: Ideal specific isobaric heat capacity, [kJ/kg·K]
• cv0: Ideal specific isochoric heat capacity, [kJ/kg·K]
• w0: Ideal speed of sound, [m/s]
• gamma0: Ideal isoentropic exponent, [-]
• w: Speed of sound, [m/s]
• mu: Dynamic viscosity, [Pa·s]
• nu: Kinematic viscosity, [m$^2$/s]
• k: Thermal conductivity, [W/m·K]
• alfa: Thermal diffusivity, [m$^2$/s]
• sigma: Surface tension, [N/m]
• epsilon: Dielectric constant, [-]
• n: Refractive index, [-]
• Prandt: Prandtl number, [-]
• Pr: Reduced Pressure, [-]
• Tr: Reduced Temperature, [-]
• Hvap: Vaporization heat, [kJ/kg]
• Svap: Vaporization entropy, [kJ/kg·K]
• Z_rho: $(Z - 1)/\rho$, [m$^3$/kg]
• IntP: Internal pressure, [MPa]
• invT: Negative reciprocal temperature, [1/K]
• hInput: Specific heat input, [kJ/kg]

References


Attributes

CP
Gruneisen
IntP
Ks
Kt
Prandt
Z
Z_rh0
a
alfa
alfap
alfav
betap
betas
calculable Check if inputs are enough to define state

cp
cp_cv
cv
dhdP_T
dhdP_rho
dhdT_P
dhdT_rh0
dhdrho_P
dhdrho_T
dpdT_rh0
dpdrho_T
drhodP_T
drhodT_P

5.1. iapws package
epsilon
f
fi
g
gamma
h
hInput
joule
k
kappa
ks
kt
mu
n
nu
rho
s
u
v
w

Methods

__call__(self, **kwargs) Make instance callable to can add input parameter one to one

calculo(self) Calculate procedure
derivative(self, z, x, y, fase) Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a

fill(self, fase, estado) Fill phase properties

name = 'ammonia'
CASNumber = '7664-41-7'
formula = 'NH3'
synonym = 'R-717'
rhoc = 225.0
Tc = 405.4
Pc = 11333.0
M = 17.03026
Tt = 195.495
\( T_b = 239.823 \)
\( f_{\text{acent}} = 0.25601 \)
\( \text{momentoDipolar} = 1.47 \)

\[ \text{Fi0} = \{ 'ao\_exp': [], 'ao\_hyp': [], 'ao\_log': [1, -1], 'ao\_pow': [-15.81502, 4.255726, 11.47434, -1.296211, 0.5706757], 'hyp': [], 'pow': [0, 1, 0.3333333333333333, -1.5, -1.75], 'titao': [] \} \]

\[ \_\text{constants} = \{'R': 8.314471, 'c2': [1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3], \} \]

\[ \_\text{melting} = \{ 'Pref': 1000, 'Tmax': 700.0, 'Tmin': 195.495, 'Tref': 195.495, 'a1': \} \]

\[ \_\text{surf} = \{ 'exp': [1.211, 5.585], 'sigma': [0.1028, 0.09453] \} \]

\[ \_\text{Pv} = \{ 'ao': [-7.0993, -2.433, 8.7591, -6.4091, -2.1185], 'eq': 5, 'exp': [1.0, 1.5, 1.7, 1.95, 4.2] \} \]

\[ \_\text{rhoL} = \{ 'ao': [34.488, -128.49, 173.82, -106.99, 30.339], 'eq': 1, 'exp': [0.58, 0.75, 0.9, 1.1, 1.3] \} \]

\[ \_\text{rhoG} = \{ 'ao': [-0.38435, -4.0846, -6.6634, -31.881, 213.06, -246.48], 'eq': 3, 'exp': [0.218, 0.55, 1.5, 3.7, 5.5, 5.8] \} \]

**Equation for the Viscosity**

**Parameters**

- \( \text{rho}(\text{float}) \) – Density [kg/m\(^3\)]
- \( T(\text{float}) \) – Temperature [K]

**Returns** \( \mu \) – Viscosity [Pa·s]

**Return type** float

**References**


**Equation for the thermal conductivity**

**Parameters**

- \( \text{rho}(\text{float}) \) – Density, [kg/m\(^3\)]
- \( T(\text{float}) \) – Temperature, [K]
- \( \text{fase}(\text{dict}) \) – phase properties

**Returns** \( k \) – Thermal conductivity [W/mK]

**Return type** float

**References**


**class** iapws.ammonia.H2ONH3

**prop**(self, rho, T, x)

Thermodynamic properties of ammonia-water mixtures
Parameters

- $T$ (float) – Temperature [K]
- $\rho$ (float) – Density [kg/m$^3$]
- $x$ (float) – Mole fraction of ammonia in mixture [mol/mol]

Returns

$prop$ – Dictionary with thermodynamic properties of ammonia-water mixtures:

- $M$: Mixture molecular mass, [g/mol]
- $P$: Pressure, [MPa]
- $u$: Specific internal energy, [kJ/kg]
- $s$: Specific entropy, [kJ/kgK]
- $h$: Specific enthalpy, [kJ/kg]
- $a$: Specific Helmholtz energy, [kJ/kg]
- $g$: Specific gibbs energy, [kJ/kg]
- $c_v$: Specific isochoric heat capacity, [kJ/kgK]
- $c_p$: Specific isobaric heat capacity, [kJ/kgK]
- $w$: Speed of sound, [m/s]
- $fug_{H2O}$: Fugacity of water, [-]
- $fug_{NH3}$: Fugacity of ammonia, [-]

Return type dict

References


$\_\phi0 (self, rho, T, x)$

Ideal gas Helmholtz energy of binary mixtures and derivatives

Parameters

- $\rho$ (float) – Density, [kg/m$^3$]
- $T$ (float) – Temperature, [K]
- $x$ (float) – Mole fraction of ammonia in mixture, [mol/mol]

Returns

$prop$ – Dictionary with ideal adimensional helmholtz energy and derivatives:

- $\tau$: the adimensional temperature variable, [-]
- $\delta$: the adimensional density variable, [-]
- $fio$,[-]
- $fiot$: $[fio/\tau]\delta$ [-]
- $fiod$: $[fio/\delta]\tau$ [-]
• fiott: \[^2 \text{fio}/\tau^2 \] \delta [-]
• fiodt: \[^2 \text{fio}/\tau \delta \] [-]
• fiodd: \[^2 \text{fio}/\delta^2 \] \tau [-]

**Return type** dict

**References**


\_phir (self, rho, T, x)

Residual contribution to the free Helmholtz energy

**Parameters**

• rho (float) – Density, \([\text{kg/m}^3]\)
• T (float) – Temperature, \([\text{K}]\)
• x (float) – Mole fraction of ammonia in mixture, \([\text{mol/mol}]\)

**Returns**

prop – dictionary with residual adimensional helmholtz energy and derivatives:

• tau: the adimensional temperature variable, [-]
• delta: the adimensional density variable, [-]
• fir, [-]
• fir: \([\text{fir}/\tau \delta] \tau, x [-]\)
• fir: \([\text{fir}/\tau^2] \delta, x [-]\)
• fir: \([\text{fir}/\delta^2] \tau, x [-]\)
• fir: \([\text{fird}/\tau^2] \delta, x [-]\)
• fir: \([\text{fird}/\delta^2] \tau, x [-]\)
• fir: \([\text{fird}/\delta] \tau, \delta [-]\)

**Return type** dict

**References**


\_Dphir (self, tau, delta, x)

Departure function to the residual contribution to the free Helmholtz energy

**Parameters**

• tau (float) – Adimensional temperature, [-]
• delta (float) – Adimensional density, [-]
• x (float) – Mole fraction of ammonia in mixture, \([\text{mol/mol}]\)
Returns

**prop** – Dictionary with departure contribution to the residual adimensional helmholtz energy and derivatives:

- **fir** [-]
- **firt**: \([\Delta \text{fir}/\tau] \delta, x [-]\)
- **fird**: \([\Delta \text{fir}/\delta] \tau, x [-]\)
- **firtt**: \([2 \Delta \text{fir}/\tau^2] \delta, x [-]\)
- **firdt**: \([2 \Delta \text{fir}/\delta \tau] x [-]\)
- **firdd**: \([2 \Delta \text{fir}/\delta^2] \tau, x [-]\)
- **firx**: \([\Delta \text{fir}/x] \tau, \delta [-]\)

**Return type** dict

**References**


**iapws.ammonia.Ttr(x)**

Equation for the triple point of ammonia-water mixture

**Parameters**

- **x** (*float*) – Mole fraction of ammonia in mixture, [mol/mol]

**Returns**

- **Ttr** – Triple point temperature, [K]

**Return type** float

**Notes**

Raise **NotImplementedError** if input isn’t in limit:

- 0 ≤ x ≤ 1

**References**


### 5.1.7 iapws.humidAir module

Module with Air-water mixture properties and related properties. The module include:

- **_virial()**: Virial equations for humid air
- **_fugacity()**: Fugacity equation for humid air
- **MEoSBlend**: Special MEoS subclass to implement pseudocomponent blend with ancillary dew and bubble point
- **Air**: Multiparameter equation of state for Air as pseudocomponent
- **HumidAir**: Humid air mixture with complete functionality
iapws.humidAir.__virial__(T)
  Virial equations for humid air

  Parameters  
  T (float) – Temperature [K]

  Returns
  
  prop –
  
  Dictionary with critical coefficient:
  
  - Baa: Second virial coefficient of dry air, [m³/mol]
  - Baw: Second air-water cross virial coefficient, [m³/mol]
  - Bww: Second virial coefficient of water, [m³/mol]
  - Caaa: Third virial coefficient of dry air, [m⁶/mol]
  - Caaw: Third air-water cross virial coefficient, [m⁶/mol]
  - Caww: Third air-water cross virial coefficient, [m⁶/mol]
  - Cwww: Third virial coefficient of dry air, [m⁶/mol]
  - Bawt: dBaw/dT, [m³/molK]
  - Bawtt: d²Baw/dT², [m³/molK²]
  - Caawt: dCaaw/dT, [m⁶/molK]
  - Caawtt: d²Caaw/dT², [m⁶/molK²]
  - Cawwt: dCaww/dT, [m⁶/molK]
  - Cawwtt: d²Caww/dT², [m⁶/molK²]

  Return type dict

Notes

Raise Warning if T isn’t in range of validity:
  
  - Baa: 60 T 2000
  - Baw: 130 T 2000
  - Bww: 130 T 1273
  - Caaa: 60 T 2000
  - Caaw: 193 T 493
  - Caww: 173 T 473
  - Cwww: 130 T 1273

Examples

```python
>>> __virial__(200)['Baa']
-3.92722567e-5
```
iapws Documentation, Release 1.1.3

References


IAPWS, Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, Table 10, http://www.iapws.org/relguide/SeaAir.html

iapws.humidAir._fugacity(T, P, x)
Fugacity equation for humid air

Parameters

• T (float) – Temperature, [K]
• P (float) – Pressure, [MPa]
• x (float) – Mole fraction of water-vapor, [-]

Returns fv – fugacity coefficient, [MPa]

Return type float

Notes

Raise NotImplemented if input isn’t in range of validity:

• 193 T 473
• 0 P 5
• 0 x 1

Really the xmax is the xsaturation but isn’t implemented

Examples

>>> _fugacity(300, 1, 0.1)
0.0884061686

References


class iapws.humidAir.MEoSBlend(**kwargs)
Special meos class to implement pseudocomponent blend and defining its ancillary dew and bubble point

Attributes

CP
Gruneisen
IntP
Ks
Kt
Prandt
Z
Z_rhoea alfa alfap alfav betap betas
calculable Check if inputs are enough to define state
cp
cp_cv
cv
dhdP_T
dhdP_rhoe dhdT_P
dhdT_rhoe dhdrho_P
dhdrho_T dpdT_rhoe dpdrho_T
drhodP_T
drhodT_P
epsilon
f
fi
g
gamma
h
hInput
joule
k
kappa
ks
kt
mu
Methods

`__call__`(self, **kwargs)
Make instance callable to can add input parameter one to one

calculo(self)
Calculate procedure

derivative(self, z, x, y, fase)
Wrapper derivative for custom derived properties where x, y, z can be: P, T, rho, u, h, s, g, a

fill(self, fase, estado)
Fill phase properties

```python
classmethod _dewP(T)
Using ancillary equation return the pressure of dew point

classmethod _bubbleP(T)
Using ancillary equation return the pressure of bubble point
```

class iapws.humidAir.Air(**kwargs)
Multiparameter equation of state for Air as pseudocomponent for internal procedures, see MEEs base class

Parameters

- `T (float)` – Temperature, [K]
- `P (float)` – Pressure, [MPa]
- `rho (float)` – Density, [kg/m³]
- `v (float)` – Specific volume, [m³/kg]
- `h (float)` – Specific enthalpy, [kJ/kg]
- `s (float)` – Specific entropy, [kJ/kgK]
- `u (float)` – Specific internal energy, [kJ/kg]
- `x (float)` – Vapor quality, [-]
- `l (float, optional)` – Wavelength of light, for refractive index, [nm]
- `rho0 (float, optional)` – Initial value of density, to improve iteration, [kg/m³]
- `T0 (float, optional)` – Initial value of temperature, to improve iteration, [K]
- `x0` *(initial value of vapor quality, necessary in bad input pair definition)* – where there are two valid solution (T-h, T-s)

Notes

- It needs two incoming properties of T, P, rho, h, s, u.
• v as an alternate input parameter to rho
• T-x, P-x, preferred input pair to specified a point in two phases region

The calculated instance has the following properties:
• P: Pressure, [MPa]
• T: Temperature, [K]
• x: Vapor quality, [-]
• g: Specific Gibbs free energy, [kJ/kg]
• a: Specific Helmholtz free energy, [kJ/kg]
• v: Specific volume, [m³/kg]
• r: Density, [kg/m³]
• h: Specific enthalpy, [kJ/kg]
• u: Specific internal energy, [kJ/kg]
• s: Specific entropy, [kJ/kg·K]
• cp: Specific isobaric heat capacity, [kJ/kg·K]
• cv: Specific isochoric heat capacity, [kJ/kg·K]
• cp_cv: Heat capacity ratio, [-]
• Z: Compression factor, [-]
• fi: Fugacity coefficient, [-]
• f: Fugacity, [MPa]
• gamma: Isoentropic exponent, [-]
• alfav: Isobaric cubic expansion coefficient, [1/K]
• kappa: Isothermal compressibility, [1/MPa]
• kappas: Adiabatic compresibility, [1/MPa]
• alfap: Relative pressure coefficient, [1/K]
• betap: Isothermal stress coefficient, [kg/m³]
• joule: Joule-Thomson coefficient, [K/MPa]
• betas: Isoentropic temperature-pressure coefficient, [-]
• Gruneisen: Gruneisen parameter, [-]
• virialB: Second virial coefficient, [m³/kg]
• virialC: Third virial coefficient, [m⁶/kg²]
• dpdT_rhoo: Derivatives, dp/dT at constant rho, [MPa/K]
• dphrho_T: Derivatives, dp/drho at constant T, [MPa·m³/kg]
• drhodT_P: Derivatives, drho/dT at constant P, [kg/m³·K]
• drhodP_T: Derivatives, drho/dP at constant T, [kg/m³·MPa]
• dhdT_rhoo: Derivatives, dh/dT at constant rho, [kJ/kg·K]
• dhdP_T: Isothermal throttling coefficient, [kJ/kg·MPa]
• dhdT_P: Derivatives, dh/dT at constant P, [kJ/kg·K]
• dhdrho_T: Derivatives, dh/drho at constant T, [kJ·m³/kg²]
• dhdrho_P: Derivatives, dh/drho at constant P, [kJ·m³/kg²]
• dhp_rh: Derivatives, dh/dP at constant rho, [kJ/kg·MPa]
• kt: Isothermal Expansion Coefficient, [-]
• ks: Adiabatic Compressibility, [1/MPa]
• Ks: Adiabatic bulk modulus, [MPa]
• Kt: Isothermal bulk modulus, [MPa]
• v0: Ideal specific volume, [m³/kg]
• rho0: Ideal gas density, [kg/m³]
• u0: Ideal specific internal energy, [kJ/kg]
• h0: Ideal specific enthalpy, [kJ/kg]
• s0: Ideal specific entropy, [kJ/kg·K]
• a0: Ideal specific Helmholtz free energy, [kJ/kg]
• g0: Ideal specific Gibbs free energy, [kJ/kg]
• cp0: Ideal specific isobaric heat capacity, [kJ/kg·K]
• cv0: Ideal specific isochoric heat capacity, [kJ/kg·K]
• w0: Ideal speed of sound, [m/s]
• gamma0: Ideal isoentropic exponent, [-]
• w: Speed of sound, [m/s]
• mu: Dynamic viscosity, [Pa·s]
• nu: Kinematic viscosity, [m²/s]
• k: Thermal conductivity, [W/m·K]
• alfa: Thermal diffusivity, [m²/s]
• sigma: Surface tension, [N/m]
• epsilon: Dielectric constant, [-]
• n: Refractive index, [-]
• Prandt: Prandtl number, [-]
• Pr: Reduced Pressure, [-]
• Tr: Reduced Temperature, [-]
• Hvap: Vaporization heat, [kJ/kg]
• Svap: Vaporization entropy, [kJ/kg·K]
• Z_rh: (Z − 1)/ρ, [m³/kg]
• IntP: Internal pressure, [MPa]
• invT: Negative reciprocal temperature, [1/K]
• hInput: Specific heat input, [kJ/kg]
References


Attributes

- CP
- Gruneisen
- IntP
- Ks
- Kt
- Prandt
- Z
- Z_rhoe
- a
- alfa
- alfap
- alfav
- betap
- betas

*calculable* Check if inputs are enough to define state

- cp
- cp_cv
- cv
- dhdP_T
- dhdP_rhoe
- dhdT_P
- dhdT_rhoe
- dhdrhoe_P
- dhdrhoe_T
- dpdT_rhoe
- dpdrhoe_T
- drhoeP_T
- drhoeT_P
- epsilon
- f
- fi
Methods

```
# _call_(self, **kwargs)
Make instance callable to can add input parameter
one to one

calculo(self)
Calculate procedure

derivative(self, z, x, y, fase)
Wrapper derivative for custom derived properties
where x, y, z can be: P, T, v, rho, u, h, s, g, a

fill(self, fase, estado)
Fill phase properties
```

```
name = 'air'
CASNumber = '1'
formula = 'N2+Ar+O2'
synonym = 'R-729'
rhoc = 302.622436442
Tc = 132.6306
Pc = 3786.0
M = 28.96546
Tt = 59.75
Tb = 78.903
f_acent = 0.0335
momentoDipolar = 0.0
```
class _Liquid_Density(T)
    Auxiliary equation for the density or saturated liquid
    Parameters T (float) – Temperature [K]
    Returns rho – Saturated liquid density [kg/m³]
    Return type float

static _visco(rho, T, fase=None)
    Equation for the Viscosity
    Parameters
    • rho (float) – Density, [kg/m³]
    • T (float) – Temperature, [K]
    Returns μ – Viscosity, [Pa·s]
    Return type float

References


class _thermo(self, rho, T, fase=None)
    Equation for the thermal conductivity
    Parameters
    • rho (float) – Density, [kg/m³]
    • T (float) – Temperature, [K]
    • fase (dict) – phase properties
    Returns k – Thermal conductivity, [W/mK]
    Return type float

References


class iapws.humidAir.HumidAir(**kwargs)
    Humid air class with complete functionality
Parameters

- **T (float)** – Temperature, [K]
- **P (float)** – Pressure, [MPa]
- **rho (float)** – Density, [kg/m³]
- **v (float)** – Specific volume, [m³/kg]
- **A (float)** – Mass fraction of dry air in humid air, [kg/kg]
- **xa (float)** – Mole fraction of dry air in humid air, [-]
- **W (float)** – Mass fraction of water in humid air, [kg/kg]
- **xw (float)** – Mole fraction of water in humid air, [-]

Notes

- It needs two incoming properties of T, P, rho.
- v as an alternate input parameter to rho
- For composition need one of A, xa, W, xw.

The calculated instance has the following properties:

- **P**: Pressure, [MPa]
- **T**: Temperature, [K]
- **g**: Specific Gibbs free energy, [kJ/kg]
- **a**: Specific Helmholtz free energy, [kJ/kg]
- **v**: Specific volume, [m³/kg]
- **rho**: Density, [kg/m³]
- **h**: Specific enthalpy, [kJ/kg]
- **u**: Specific internal energy, [kJ/kg]
- **s**: Specific entropy, [kJ/kg·K]
- **cp**: Specific isobaric heat capacity, [kJ/kg·K]
- **w**: Speed of sound, [m/s]
- **alfav**: Isobaric cubic expansion coefficient, [1/K]
- **betas**: Isoentropic temperature-pressure coefficient, [-]
- **xkappa**: Isothermal Expansion Coefficient, [-]
- **ks**: Adiabatic Compressibility, [1/MPa]
- **A**: Mass fraction of dry air in humid air, [kg/kg]
- **W**: Mass fraction of water in humid air, [kg/kg]
- **xa**: Mole fraction of dry air, [-]
- **xw**: Mole fraction of water, [-]
- **xa_sat**: Mole fraction of dry air at saturation state, [-]
- **mu**: Relative chemical potential, [kJ/kg]
• muw: Chemical potential of water, [kJ/kg]
• M: Molar mass of humid air, [g/mol]
• HR: Humidity ratio, [-]
• RH: Relative humidity, [-]

Attributes

**calculable** Check if inputs are enough to define state

Methods

```python
__call__(self, **kwargs) Make instance callable to can add input parameter one to one

`calculo(self)` Calculate procedure

`derivative(self, z, x, y)` Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a

status = 0
msg = 'Undefined'
kwargs = {'A': None, 'P': 0.0, 'T': 0.0, 'W': None, 'rho': 0.0, 'v': 0.0, 'xa': None, 'xw': None}

`calculable` Check if inputs are enough to define state

`calculo(self)` Calculate procedure

`derivative(self, z, x, y)` Wrapper derivative for custom derived properties where x, y, z can be: P, T, v, rho, u, h, s, g, a

_eq(self, T, P)` Procedure for calculate the composition in saturation state

Parameters

- **T (float)** – Temperature [K]
- **P (float)** – Pressure [MPa]

Returns **Asat** – Saturation mass fraction of dry air in humid air [kg/kg]

Return type **float**

_prop(self, T, rho, fav)` Thermodynamic properties of humid air

Parameters

- **T (float)** – Temperature, [K]
- **rho (float)** – Density, [kg/m³]
- **fav (dict)** – dictionary with helmholtz energy and derivatives

Returns

prop –

5.1. iapws package 125
Dictionary with thermodynamic properties of humid air:

- **P**: Pressure, [MPa]
- **s**: Specific entropy, [kJ/kgK]
- **cp**: Specific isobaric heat capacity, [kJ/kgK]
- **h**: Specific enthalpy, [kJ/kg]
- **g**: Specific gibbs energy, [kJ/kg]
- **alfav**: Thermal expansion coefficient, [1/K]
- **betas**: Isentropic T-P coefficient, [K/MPa]
- **xkappa**: Isothermal compressibility, [1/MPa]
- **ks**: Isentropic compressibility, [1/MPa]
- **w**: Speed of sound, [m/s]

Return type  **dict**

**References**

IAPWS, Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, Table 5, http://www.iapws.org/relguide/SeaAir.html

**_coligative (self, rho, A, fav)**

Miscellaneous properties of humid air

**Parameters**

- **rho** (*float*) – Density, [kg/m³]
- **A** (*float*) – Mass fraction of dry air in humid air, [kg/kg]
- **fav** (*dict*) – dictionary with helmholtz energy and derivatives

**Returns**

**prop** –

Dictionary with calculated properties:

- **mu**: Relative chemical potential, [kJ/kg]
- **muw**: Chemical potential of water, [kJ/kg]
- **M**: Molar mass of humid air, [g/mol]
- **HR**: Humidity ratio, [-]
- **xa**: Mole fraction of dry air, [-]
- **xw**: Mole fraction of water, [-]

**Return type  **dict**
References

IAPWS, Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent
iapws.org/relguide/SeaAir.html

```python
_fav(self, T, rho, A)
Specific Helmholtz energy of humid air and derivatives

Parameters

* `T (float)` – Temperature, [K]
* `rho (float)` – Density, [kg/m³]
* `A (float)` – Mass fraction of dry air in humid air, [kg/kg]

Returns

prop –
Dictionary with helmholtz energy and derivatives:

* `fir`, [kJ/kg]
* `fira`: \( \frac{\partial f_{av}}{\partial A} \right|_{T,\rho} \), [kJ/kg]
* `firt`: \( \frac{\partial f_{av}}{\partial T} \right|_{A,\rho} \), [kJ/kgK]
* `fird`: \( \frac{\partial f_{av}}{\partial \rho} \right|_{A,T} \), [kJ/m³kg²]
* `fira`: \( \frac{\partial^2 f_{av}}{\partial A^2} \right|_{T,\rho} \), [kJ/kg]
* `firt`: \( \frac{\partial^2 f_{av}}{\partial A \partial T} \right|_{\rho} \), [kJ/kgK]
* `fird`: \( \frac{\partial^2 f_{av}}{\partial \rho \partial A} \right|_{T} \), [kJ/m³kg²]
* `firad`: \( \frac{\partial^2 f_{av}}{\partial A^2} \right|_{T,\rho} \), [kJ/kgK]
* `firtt`: \( \frac{\partial^2 f_{av}}{\partial T^2} \right|_{A,\rho} \), [kJ/kgK²]
* `firdt`: \( \frac{\partial^2 f_{av}}{\partial \rho \partial T} \right|_{A} \), [kJ/m³kg²K]
* `firdd`: \( \frac{\partial^2 f_{av}}{\partial \rho^2} \right|_{A,T} \), [kJ/m⁶kg³]

Return type `dict`

References

IAPWS, Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent
iapws.org/relguide/SeaAir.html

```python
_fmix(self, T, rho, A)
Specific Helmholtz energy of air-water interaction

Parameters

* `T (float)` – Temperature, [K]

5.1. iapws package
• **rho** (*float*) – Density, \([\text{kg/m}^3]\)
• **A** (*float*) – Mass fraction of dry air in humid air, \([\text{kg/kg}]\)

**Returns**

**prop** –

Dictionary with helmholtz energy and derivatives:

- **fir**, \([\text{kJ/kg}]\)
- **fira**:
  \[\frac{\partial f_{mix}}{\partial A} \bigg|_{T, \rho}, \quad [\text{kJ/kg}]\]
- **firt**:
  \[\frac{\partial f_{mix}}{\partial T} \bigg|_{A, \rho}, \quad [\text{kJ/kgK}]\]
- **fird**:
  \[\frac{\partial^2 f_{mix}}{\partial \rho} \bigg|_{A, T}, \quad [\text{kJ/m}^3\text{kg}^2]\]
- **fira**: \[\frac{\partial^2 f_{mix}}{\partial A^2} \bigg|_{T, \rho}, \quad [\text{kJ/kg}]\]
- **firt**: \[\frac{\partial^2 f_{mix}}{\partial A \partial T} \bigg|_{\rho}, \quad [\text{kJ/kgK}]\]
- **fird**: \[\frac{\partial^2 f_{mix}}{\partial \rho \partial T} \bigg|_{A}, \quad [\text{kJ/m}^3\text{kg}^2\text{K}]\]
- **firdd**: \[\frac{\partial^2 f_{mix}}{\partial \rho^2} \bigg|_{A, T}, \quad [\text{kJ/m}^6\text{kg}^3]\]

**Return type** dict

**References**

IAPWS, Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, Table 10, [http://www.iapws.org/relguide/SeaAir.html](http://www.iapws.org/relguide/SeaAir.html)

For a rapid usage demonstration, see these examples

**IAPWS-IF97 (see full documentation)**

```python
from iapws import IAPWS97
sat_steam=IAPWS97(P=1,x=1)  #saturated steam with known P
sat_liquid=IAPWS97(T=370, x=0)  #saturated liquid with known T
steam=IAPWS97(P=2.5, T=500)  #steam with known P and T
print(sat_steam.h, sat_liquid.h, steam.h)  #calculated enthalpies
```

**IAPWS-95 (see full documentation)**

```python
from iapws import IAPWS95
sat_steam=IAPWS95(P=1,x=1)  #saturated steam with known P
sat_liquid=IAPWS95(T=370, x=0)  #saturated liquid with known T
steam=IAPWS95(P=2.5, T=500)  #steam with known P and T
print(sat_steam.h, sat_liquid.h, steam.h)  #calculated enthalpies
```

**IAPWS-17 for Heavy water (see full documentation)**
from iapws import D2O
sat_liquid=D2O(T=370, x=0)  # saturated liquid with known $T$
print(sat_liquid.h)  # calculated enthalpy

IAPWS-06 for Ice Ih (see full documentation)

from iapws import _Ice
ice=_Ice(273.15, 0.101325)  # Ice at normal melting point
print(ice['rho'])  # calculated density

IAPWS-08 for seawater (see full documentation)

from iapws import SeaWater
state = SeaWater(T=300, P=0.101325, S=0.001)  # Seawater with 0.1% Salinity
print(state.cp)  # get cp

5.1. iapws package
• FIXME: Electrolytic conductivity
• TODO: Improve convergence in two phase region for IAPWS95 and D2O class
• TODO: Implement SBTL method for fast calculation
• TODO: Implement TTSE method for fast calculation

Ammonia-water mixture:
• FIXME: Ammonia-water mixture residual helmholtz. The values are good, bad differ by 1%
• TODO: Add equilibrium routine

I’ve tried to test all code and use all values for computer verification the standards give, but anyway the code can have hidden problem. For any suggestions, comments, bugs … you can usage the github issue section, or contact directly with me at email.
CHAPTER 7

Introduction

Python implementation of standard from IAPWS (http://www.iapws.org/release.html).

- Home: https://github.com/jjgomera/iapws
- Author: Juan José Gómez Romera <jjgomera@gmail.com>
- License: GPL-3
- Documentation: http://iapws.readthedocs.io/
CHAPTER 8

Dependences

- python 2x, 3x, compatible with both versions
- Numpy-scipy: library with mathematic and scientific tools
CHAPTER 9

Installation

In Debian you can find in official repositories in jessie, testing and sid. In Ubuntu it’s in official repositories from Ubuntu saucy (13.10). In other systems you can install using pip:

```
pip install iapws
```

or directly cloning the github repository:

```
git clone https://github.com/jjgomera/iapws.git
```

and adding the folder to a python path.
CHAPTER 10

Features

This module implements almost the full set of standards:

Releases:

- R1-76(2014): Revised Release on the Surface Tension of Ordinary Water Substance, \texttt{iapws\_iapws\_Tension()}
- R2-83(1992): Release on the Values of Temperature, Pressure and Density of Ordinary and Heavy Water Substances at their Respective Critical Points, \texttt{iapws\_iapws()}
- R4-84(2007): Revised Release on Viscosity and Thermal Conductivity of Heavy Water Substance, \texttt{iapws\_iapws\_D2O\_Viscosity(),iapws\_iapws\_D2O\_ThCond()}
- R5-85(1994): Release on Surface Tension of Heavy Water Substance, \texttt{iapws\_iapws\_D2O\_Tension()}
- R8-97: Release on the Static Dielectric Constant of Ordinary Water Substance for Temperatures from 238 K to 873 K and Pressures up to 1000 MPa, \texttt{iapws\_iapws\_Dielectric()}
- R9-97: Release on the Refractive Index of Ordinary Water Substance as a Function of Wavelength, Temperature and Pressure, \texttt{iapws\_iapws\_Refractive()}
- R11-07: Release on the Ionization Constant of H2O, \texttt{iapws\_iapws\_Kw()}
- R12-08: Release on the IAPWS Formulation 2008 for the Viscosity of Ordinary Water Substance, \texttt{iapws\_iapws\_Viscosity()}
- R13-08: Release on the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, \texttt{iapws\_iapws08()}
• R14-08(2011): Revised Release on the Pressure along the Melting and Sublimation Curves of Ordinary Water Substance, \texttt{iapws\_iapws\_Melting\_Pressure()}, \texttt{iapws\_iapws\_Sublimation\_Pressure()}

• R15-11: Release on the IAPWS Formulation 2011 for the Thermal Conductivity of Ordinary Water Substance, \texttt{iapws\_iapws\_ThCond()}

• R16-17: Release on the IAPWS Formulation 2017 for the Thermodynamic Properties of Heavy Water, \texttt{iapws\_iapws95.D2O()}

Supplementary Releases:

• SR1-86(1992): Revised Supplementary Release on Saturation Properties of Ordinary Water Substance, \texttt{iapws\_iapws95.MEoS\_Liquid\_Density()}, \texttt{iapws\_iapws95.MEoS\_Vapor\_Density()}, \texttt{iapws\_iapws95.MEoS\_Vapor\_Pressure()}

• SR2-01(2014): Revised Supplementary Release on Backward Equations for Pressure as a Function of Enthalpy and Entropy \(p(h,s)\) for Regions 1 and 2 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward1\_P\_hs()}, \texttt{iapws\_iapws97\_Backward2\_P\_hs()}

• SR3-03(2014): Revised Supplementary Release on Backward Equations for the Functions \(T(p,h), v(p,h),\) and \(T(p,s), v(p,s)\) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward3\_T\_Ph()}, \texttt{iapws\_iapws97\_Backward3\_T\_Ps()}, \texttt{iapws\_iapws97\_Backward3\_v\_Ph()}, \texttt{iapws\_iapws97\_Backward3\_v\_Ps()}

• SR4-04(2014): Revised Supplementary Release on Backward Equations \(p(h,s)\) for Region 3, Equations as a Function of \(h\) and \(s\) for the Region Boundaries, and an Equation \(T_{sat}(h,s)\) for Region 4 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward3\_P\_hs()}

• SR5-05(2016): Revised Supplementary Release on Backward Equations for Specific Volume as a Function of Pressure and Temperature \(v(p,T)\) for Region 3 of the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, \texttt{iapws\_iapws97\_Backward3\_v\_PT()}

• SR6-08(2011): Revised Supplementary Release on Properties of Liquid Water at 0.1 MPa, \texttt{iapws\_iapws\_Liquid()}

• SR7-09: Supplementary Release on a Computationally Efficient Thermodynamic Formulation for Liquid Water for Oceanographic Use, \texttt{iapws\_iapws08\_SeaWater\_waterSupp()}

Guidelines:

• G1-90: Electrolytic Conductivity (Specific Conductance) of Liquid and Dense Supercritical Water from 0°C to 800°C and Pressures up to 1000 MPa, \texttt{iapws\_iapws\_Conductivity()}

• G2-90(1994): Solubility of Sodium Sulfate in Aqueous Mixtures of Sodium Chloride and Sulfuric Acid from Water to Concentrated Solutions, from 250 °C to 350 °C, \texttt{iapws\_iapws08\_solNa2SO4()}

• G3-00(2012): Revised Guideline on the Critical Locus of Aqueous Solutions of Sodium Chloride, \texttt{iapws\_iapws08\_critNaCl()}

• G4-01: Guideline on the IAPWS Formulation 2001 for the Thermodynamic Properties of Ammonia-Water Mixtures, \texttt{iapws\_ammonia()}

• G5-01(2016): Guideline on the Use of Fundamental Physical Constants and Basic Constants of Water, \texttt{iapws\_iapws()}

• G6-03: Guideline on the Tabular Taylor Series Expansion (TTSE) Method for Calculation of Thermodynamic Properties of Water and Steam Applied to IAPWS-95 as an Example (Not implemented)

• G7-04: Guideline on the Henry’s Constant and Vapor-Liquid Distribution Constant for Gases in H2O and D2O at High Temperatures, \texttt{iapws\_iapws\_Henry()}, \texttt{iapws\_iapws\_Kvalue()}
• G8-10: Guideline on an Equation of State for Humid Air in Contact with Seawater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of Seawater, _HumidAir()

• G9-12: Guideline on a Low-Temperature Extension of the IAPWS-95 Formulation for Water Vapor, _IAPWS95._phiex()

• G10-15: Guideline on the Thermal Conductivity of Seawater, _ThCond_SeaWater()

• G11-15: Guideline on a Virial Equation for the Fugacity of H2O in Humid Air, _virial()

• G12-15: Guideline on Thermodynamic Properties of Supercooled Water, _Supercooled()

• G13-15: Guideline on the Fast Calculation of Steam and Water Properties with the Spline-Based Table Look-Up Method (SBTL) (Not implemented)

Advisory Notes:

• AN1-03: Uncertainties in Enthalpy for the IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use (IAPWS-95) and the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam (IAPWS-IF97)

• AN2-04(2013): Role of Various IAPWS Documents Concerning the Thermodynamic Properties of Ordinary Water Substance

• AN3-07(2014): Thermodynamic Derivatives from IAPWS Formulations, _deriv_G(), _deriv_H()

• AN4-09: Roles of IAPWS and CIPM Standards for the Density of Water

• AN5-13(2016): Industrial Calculation of the Thermodynamic Properties of Seawater, _waterIF97(), _Tb(), _Tf(), _Triple(), _OsmoticPressure()

• AN6-16: Relationship between Various IAPWS Documents and the International Thermodynamic Equation of Seawater - 2010 (TEOS-10)
You can navigate the full documentation of package:

For a rapid usage demostration, see this examples

IAPWS-IF97 (see full documentation)

```python
from iapws import IAPWS97
sat_steam=IAPWS97(P=1,x=1)  #saturated steam with known P
sat_liquid=IAPWS97(T=370, x=0)  #saturated liquid with known T
steam=IAPWS97(P=2.5, T=500)  #steam with known P and T
print(sat_steam.h, sat_liquid.h, steam.h)  #calculated enthalpies
```

IAPWS-95 (see full documentation)

```python
from iapws import IAPWS95
sat_steam=IAPWS95(P=1,x=1)  #saturated steam with known P
sat_liquid=IAPWS95(T=370, x=0)  #saturated liquid with known T
steam=IAPWS95(P=2.5, T=500)  #steam with known P and T
print(sat_steam.h, sat_liquid.h, steam.h)  #calculated enthalpies
```

IAPWS-17 for Heavy water (see full documentation)

```python
from iapws import D2O
sat_liquid=D2O(T=370, x=0)  #saturated liquid with known T
print(sat_liquid.h)  #calculated enthalpy
```

IAPWS-06 for Ice Ih (see full documentation)

```python
from iapws import _Ice
ice=_Ice(273.15, 0.101325)  #Ice at normal melting point
print(ice["rho"])  #Calculated density
```

IAPWS-08 for seawater (see full documentation)
from iapws import SeaWater

state = SeaWater(T=300, P=0.101325, S=0.001)  # Seawater with 0.1% Salinity
print(state.cp)  # Get cp
• FIXME: Electrolytic conductivity
• TODO: Improve convergence in two phase region for IAPWS95 and D2O class
• TODO: Implement SBTL method for fast calculation
• TODO: Implement TTSE method for fast calculation

Ammonia-water mixture:
• FIXME: Ammonia-water mixture residual helmholtz. The values are good, bad differ by 1%
• TODO: Add equilibrium routine

I’ve tried to test all code and use all values for computer verification the standards give, but anyway the code can have hidden problem. For any suggestions, comments, bugs . . . you can usage the github issue section, or contact directly with me at email.
CHAPTER 13

Indices and tables

• genindex
• modindex
• search
iapws._iapws, 13
iapws._utils, 26
iapws.ammonia, 106
iapws.humidAir, 114
iapws.iapws08, 100
iapws.iapws95, 29
iapws.iapws97, 60
Symbols

_Backward1_P_hs() (in module iapws.iapws97), 70
_Backward1_T_Ps() (in module iapws.iapws97), 70
_Backward2_P_hs() (in module iapws.iapws97), 78
_Backward2_T_Ps() (in module iapws.iapws97), 76
_Backward2a_P_hs() (in module iapws.iapws97), 85
_Backward2b_P_hs() (in module iapws.iapws97), 86
_Backward2c_P_hs() (in module iapws.iapws97), 87
_Backward3_P_hs() (in module iapws.iapws97), 86
_Backward3_T_Ph() (in module iapws.iapws97), 83
_Backward3b_P_hs() (in module iapws.iapws97), 86
_Backward3b_T_Ph() (in module iapws.iapws97), 83
_Backward3b_T_Ps() (in module iapws.iapws97), 85
_Backward3b_v_Ph() (in module iapws.iapws97), 82
_Backward3b_v_Ps() (in module iapws.iapws97), 84
_Backward3x_v_PT() (in module iapws.iapws97), 87
_Backward4_T_hs() (in module iapws.iapws97), 90
_Bound_Ph() (in module iapws.iapws97), 92
_Bound_Ps() (in module iapws.iapws97), 92
_Bound_TP() (in module iapws.iapws97), 92
_Bound_hs() (in module iapws.iapws97), 93
Conductivity() (in module iapws.iapws97), 22
_D2O_Melting_Pressure() (in module iapws.iapws97), 24
_D2O_Sublimation_Pressure() (in module iapws.iapws97), 24
_D2O_Tension() (in module iapws.iapws97), 23
_D2O_ThCond() (in module iapws.iapws97), 23
_D2O_Viscosity() (in module iapws.iapws97), 22
_Dielectric() (in module iapws.iapws97), 20
_Dphir() (iapws.ammonia.H2ONH3 method), 113
_Helmholtz() (iapws.iapws95.MEoS method), 35
_Henry() (in module iapws.iapws97), 25
_Ice() (in module iapws.iapws97), 14
_Kvalue() (in module iapws.iapws97), 25
_Kw() (in module iapws.iapws97), 21
_Liquid() (in module iapws.iapws97), 15
_Liquid_Density() (iapws.humidAir.Air class method), 123
_Liquid_Density() (iapws.iapws95.MEoS class method), 123

Index
method), 38
_Liquid_Enthalpy() (iapws.iapws95.IAPWS95 class method), 46
_Liquid_Entropy() (iapws.iapws95.IAPWS95 class method), 46
_Melting_Pressure() (in module iapws.iapws), 18
_OsmoticPressure() (in module iapws.iapws08), 104
_P23_T() (in module iapws.iapws97), 63
_PSat_T() (in module iapws.iapws97), 64
_PSat_h() (in module iapws.iapws97), 65
_PSat_s() (in module iapws.iapws97), 66
_P_2bc() (in module iapws.iapws97), 72
_Pv (iapws.ammonia.NH3 attribute), 111
_Pv (iapws.humidAir.Air attribute), 123
_Pv (iapws.iapws95.D2O attribute), 60
_Pv (iapws.iapws95.IAPWS95 attribute), 45
_Pv (iapws.iapws95.MEoS attribute), 35
_Reactive() (in module iapws.iapws), 20
_Reg1n1() (in module iapws.iapws97), 68
_Reg2n2() (in module iapws.iapws97), 71
_Reg3n3() (in module iapws.iapws97), 78
_Reg4n4() (in module iapws.iapws97), 89
_Reg5n5() (in module iapws.iapws97), 90
_Sublimation_Pressure() (in module iapws_iapws), 17
_Supercooled() (in module iapws_iapws), 16
_TSat_P() (in module iapws.iapws97), 65
_Tr() (in module iapws.iapws08), 103
_Tension() (in module iapws_iapws), 19
_Tf() (in module iapws.iapws08), 103
_ThCond() (in module iapws_iapws), 19
_ThCond_SeaWater() (in module iapws.iapws08), 104
_Triple() (in module iapws.iapws08), 103
_Vapor_Density() (iapws.iapws95.MEoS class method), 38
_Vapor_Enthalpy() (iapws.iapws95.IAPWS95 class method), 46
_Vapor_Entropy() (iapws.iapws95.IAPWS95 class method), 47
_Vapor_Pressure() (iapws.iapws95.MEoS class method), 38
_Viscosity() (in module iapws_iapws), 18
_alfa_sat() (iapws.iapws95.IAPWS95 class method), 45
_blend (iapws.humidAir.Air attribute), 123
_bubbleP() (iapws.humidAir.MEoSBlend class method), 118
_coligative() (iapws.humidAir.HumidAir method), 126
_constants (iapws.ammonia.NH3 attribute), 111
_constants (iapws.humidAir.Air attribute), 123
__constants (iapws.iapws95.D2O attribute), 60
__constants (iapws.iapws95.IAPWS95 attribute), 45
_critNaCl() (in module iapws.iapws08), 105
dPDt_sat() (iapws.iapws95.MEoS class method), 38
derivDimensional() (iapws.iapws95.MEoS method), 37
dewP() (iapws.humidAir.MEoSBlend class method), 118
eq() (iapws.humidAir.HumidAir method), 125
de phase (in iapws_iapws), 26
def() (iapws.humidAir.HumidAir method), 127
fmix() (iapws.humidAir.HumidAir method), 127
fugacity() (in module iapws.humidAir), 116
h13_s() (in module iapws.iapws97), 62
h1_s() (in module iapws.iapws97), 66
h2ab_s() (in module iapws.iapws97), 67
h2c3b_s() (in module iapws.iapws97), 68
h3a_s() (in module iapws.iapws97), 67
h_3ab() (in module iapws.iapws97), 79
hab_s() (in module iapws.iapws97), 73
hbc_P() (in module iapws.iapws97), 73
melting (iapws.ammonia.NH3 attribute), 111
melting (iapws.humidAir.Air attribute), 123
phi0() (iapws.ammonia.H2ONH3 method), 112
phi0() (iapws.iapws95.IAPWS95 method), 45
phi0() (iapws.iapws95.MEoS method), 36
phi_sat() (iapws.iapws95.MEoS class method), 46
phiex() (iapws.iapws95.IAPWS95 method), 45
phir() (iapws.ammonia.H2ONH3 attribute), 113
phir() (iapws.iapws95.MEoS method), 36
phir() (in module iapws.iapws95), 29
phird() (in module iapws.iapws95), 30
phirt() (in module iapws.iapws95), 30
prop() (iapws.ammonia.H2ONH3 method), 111
prop() (iapws.humidAir.HumidAir method), 125
prop0() (iapws.iapws95.MEoS method), 36
rhoG (iapws.ammonia.NH3 attribute), 111
rhoG (iapws.humidAir.Air attribute), 123
rhoG (iapws.iapws95.D2O attribute), 60
rhoG (iapws.iapws95.IAPWS95 attribute), 45
rhoG (iapws.iapws95.MEoS attribute), 35
rhoL (iapws.ammonia.NH3 attribute), 111
rhoL (iapws.iapws95.D2O attribute), 60
rhoL (iapws.iapws95.IAPWS95 attribute), 45
rhoL (iapws.iapws95.MEoS attribute), 35
saline() (iapws.iapws08.SeaWater class method), 103
saturation() (iapws.iapws95.MEoS method), 35
solNa2SO4() (in module iapws.iapws08), 105
surf() (iapws.ammonia.NH3 attribute), 111
surf() (iapws.humidAir.Air attribute), 123
surface() (iapws.iapws95.D2O method), 60
surface() (iapws.iapws95.IAPWS95 method), 47
\_surface() (iapws.iapws95.MEOS method), 37
\_t\_P() (in module iapws.iapws97), 63
\_t\_hs() (in module iapws.iapws97), 64
\_tab\_P() (in module iapws.iapws97), 79
\_tef\_P() (in module iapws.iapws97), 80
\_thermo() (iapws.ammonia.NH3 method), 111
\_thermo() (iapws.humidAir.Air method), 123
\_thermo() (iapws.iapws95.D2O method), 60
\_therm() (iapws.iapws95.IAPWS95 method), 47
\_top\_P() (in module iapws.iapws97), 79
\_txx\_P() (in module iapws.iapws97), 80
\_virial() (iapws.iapws95.MEOS method), 37
\_virial() (in module iapws.humidAir), 114
\_visco() (iapws.ammonia.NH3 method), 111
\_visco() (iapws.humidAir.Air static method), 123
\_visco() (iapws.iapws95.D2O method), 60
\_visco() (iapws.iapws95.IAPWS95 method), 47
\_water() (iapws.iapws08.SeaWater class method), 102
\_water\_IF97() (iapws.iapws08.SeaWater class method), 103
\_water\_Supp() (iapws.iapws08.SeaWater class method), 103
\nF
f\_acent (iapws.ammonia.NH3 attribute), 111
f\_acent (iapws.humidAir.Air attribute), 122
f\_acent (iapws.iapws95.D2O attribute), 60
f\_acent (iapws.iapws95.IAPWS95 attribute), 45
F10 (iapws.ammonia.NH3 attribute), 111
F10 (iapws.humidAir.Air attribute), 122
F10 (iapws.iapws95.D2O attribute), 60
F10 (iapws.iapws95.IAPWS95 attribute), 45
fill() (iapws.iapws95.MEOS method), 35
fill() (iapws.iapws97.IAPWS97 method), 96
formula (iapws.ammonia.NH3 attribute), 110
formula (iapws.humidAir.Air attribute), 122
formula (iapws.iapws95.D2O attribute), 60
formula (iapws.iapws95.IAPWS95 attribute), 45
\nG
ggetphase() (in module iapws._utils), 26
\nH
H2ONH3 (class in iapws.ammonia), 111
HumidAir (class in iapws.humidAir), 123
\nI
iapws\_iapws (module), 13
iapws\_utils (module), 26
iapws.ammonia (module), 106
iapws.humidAir (module), 114
iapws.iapws08 (module), 100
iapws.iapws95 (module), 29
iapws.iapws97 (module), 60
IAPWS95 (class in iapws.iapws95), 39
IAPWS95\_Ph (class in iapws.iapws95), 49
IAPWS95\_Ps (class in iapws.iapws95), 50
IAPWS95\_PT (class in iapws.iapws95), 47
IAPWS95\_Px (class in iapws.iapws95), 52
IAPWS97 (class in iapws.iapws97), 93, 98
IAPWS97\_Ph (class in iapws.iapws97), 96
IAPWS97\_Ps (class in iapws.iapws97), 97
IAPWS97\_PT (class in iapws.iapws97), 96
IAPWS97\_Px (class in iapws.iapws97), 97
IAPWS97\_Tx (class in iapws.iapws97), 97
\nK
kwargs (iapws.humidAir.HumidAir attribute), 125
kwargs (iapws.iapws08.SeaWater attribute), 102
kwargs (iapws.iapws95.MEOS attribute), 35
kwargs (iapws.iapws97.IAPWS97 attribute), 96
\nM
M (iapws.ammonia.NH3 attribute), 110
M (iapws.humidAir.Air attribute), 122
iapws Documentation, Release 1.1.3

M (iapws.iapws95.D2O attribute), 60
M (iapws.iapws95.IAPWS95 attribute), 45
mainClassDoc () (in module iapws.iapws95), 39
MEoS (class in iapws.iapws95), 31
MEoSBlend (class in iapws.humidAir), 116
momentoDipolar (iapws.ammonia.NH3 attribute), 111
momentoDipolar (iapws.humidAir.Air attribute), 122
momentoDipolar (iapws.iapws95.D2O attribute), 60
momentoDipolar (iapws.iapws95.IAPWS95 attribute), 45
msg (iapws.humidAir.HumidAir attribute), 125
msg (iapws.iapws08.SeaWater attribute), 102
msg (iapws.iapws95.MEoS attribute), 35
msg (iapws.iapws97.IAPWS97 attribute), 96

N
name (iapws.ammonia.NH3 attribute), 110
name (iapws.humidAir.Air attribute), 122
name (iapws.iapws95.D2O attribute), 60
name (iapws.iapws95.IAPWS95 attribute), 45
NH3 (class in iapws.ammonia), 106

P
Pc (iapws.ammonia.NH3 attribute), 110
Pc (iapws.humidAir.Air attribute), 122
Pc (iapws.iapws95.D2O attribute), 60
Pc (iapws.iapws95.IAPWS95 attribute), 45
prop0 () (in module iapws.iapws97), 93

R
Region2_cp0 () (in module iapws.iapws97), 72
Region5_cp0 () (in module iapws.iapws97), 91
rhoc (iapws.ammonia.NH3 attribute), 110
rhoc (iapws.humidAir.Air attribute), 122
rhoc (iapws.iapws95.D2O attribute), 60
rhoc (iapws.iapws95.IAPWS95 attribute), 45

S
SeaWater (class in iapws.iapws08), 100
status (iapws.humidAir.HumidAir attribute), 125
status (iapws.iapws08.SeaWater attribute), 102
status (iapws.iapws95.MEoS attribute), 35
status (iapws.iapws97.IAPWS97 attribute), 96
synonym (iapws.ammonia.NH3 attribute), 110
synonym (iapws.humidAir.Air attribute), 122
synonym (iapws.iapws95.D2O attribute), 60
synonym (iapws.iapws95.IAPWS95 attribute), 45

T
Tb (iapws.ammonia.NH3 attribute), 110
Tb (iapws.humidAir.Air attribute), 122
Tb (iapws.iapws95.D2O attribute), 60