iapws

Release 0.5.1

Milan Skocic

Dec 26, 2025

CONTENTS

Modern Fortran

IAPWS

Light and heavy water properties according to IAPWS.

Critical values are available as constants [1]:

- **water:**
  - *Tc_H2O*
  - *rhoc_H2O*
  - *pc_H2O*
- **In heavywater**
  - *Tc_D2O*
  - *rhoc_D2O*
  - *pc_D2O*

The Henry constant *kh* and the liquid-vapor distribution constant *kd* can be computed for the following gases as defined in [2] :

- in water: He, Ne, Ar, Kr, Xe, H2, N2, O2, CO, CO2, H2S, CH4, C2H6, SF6
- in heavywater: He, Ne, Ar, Kr, Xe, D2, CH4

The available gases can be retrieved with

- *gases* which returns the available gases as a list.
- *gases2* which return the available gases as a string.
- *ngases* which returns the number of available gases.

Five regions which cover the following range of validity [3]:

- $273.15K < T < 1073.15K$ and $p < 100MPa$
- $1073.15K < T < 2273.15K$ and $p < 50MPa$

The saturation-pressure *psat* and the saturation-temperature *Tsat* computes the saturation line as shown in the plot below.

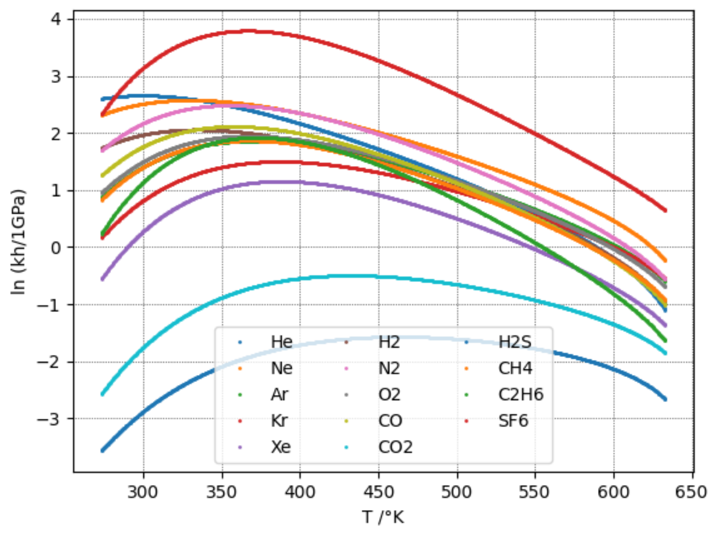

Fig. 1: k_H in H_2O

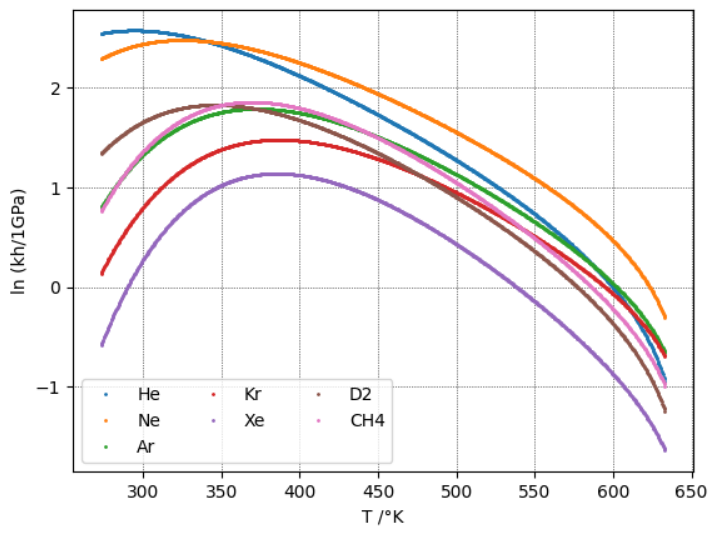

Fig. 2: k_H in D_2O

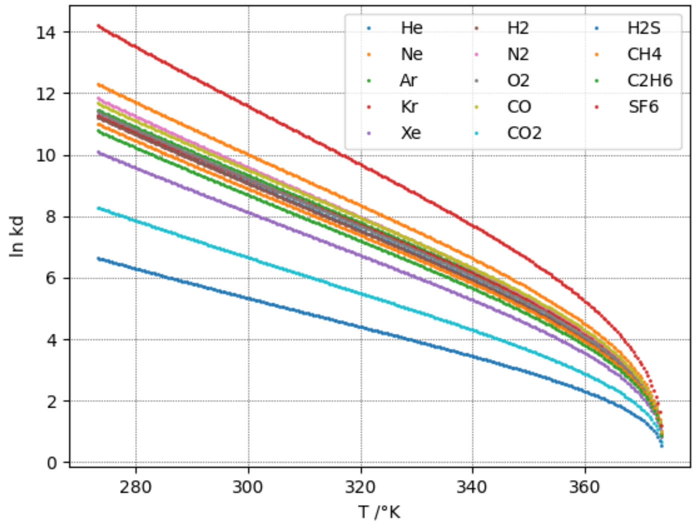

Fig. 3: k_D in H_2O

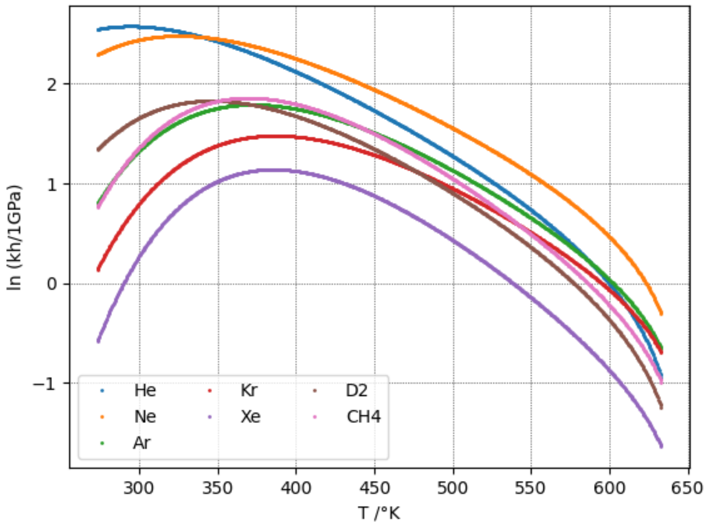

Fig. 4: k_D in D_2O

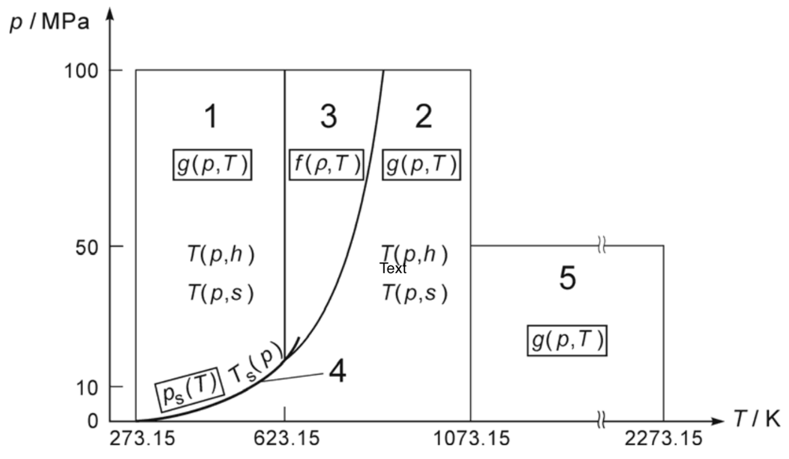

Fig. 1. Regions and equations of IAPWS-IF97.

Fig. 5: Water regions defined in R7-97

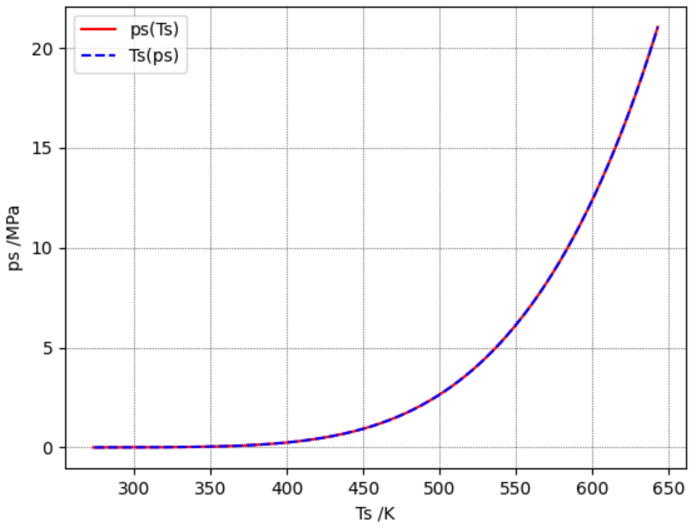

Fig. 6: Saturation line.

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

ipaws is a Fortran library providing the formulas for computing light and heavy water properties. The formulas are taken from http://iapws.org. C API allows usage from C, or can be used as a basis for other wrappers. Python wrapper allows easy usage from Python.

It covers:

- R2-83 - Tc in H2O and D2O - pc in H2O and D2O - rhoc in H2O and D2O
- G7-04
  - kH
  - kD
- R7-97
  - Region 1
  - Region 2
  - Region 3
  - Region 4
  - Region 5
- R11-24:
  - Kw

To use *iapws* within your fpm project, add the following to your *fpm.toml* file:

```
[dependencies]
iapws = { git="https://github.com/MilanSkocic/iapws.git" }
```

1.2 Dependencies

```
gcc>=10.0
gfortran>=10.0
fpm>=0.8
stdlib>=0.5
```

1.3 Installation

A Makefile is provided, which uses fpm, for building the library.

- On windows, msys2 needs to be installed. Add the msys2 binary (usually C:\msys64\usr\bin) to the path in order to be able to use make.
- On Darwin, the gcc toolchain needs to be installed.

Build: the configuration file will set all the environment variables necessary for the compilation

```
chmod +x configure.sh
./configure.sh
make
make test
make install
make uninstall
```

1.4 License

MIT

CHAPTER

TWO

EXAMPLES

2.1 Fortran

```
program example_in_f
    use stdlib_kinds, only: dp, int32
    use iapws
    implicit none
    integer(int32) :: i, ngas
    real(dp) :: T(1), kh_res(1), kd_res(1), wp_res(1), p(1)
    real(dp) :: Ts(7), ps(7)
    real(dp) :: x(3), y(3)
    integer(int32) :: r(3)
    character(len=1) :: s(3)
    character(len=2) :: gas = "O2"
    integer(int32) :: heavywater = 0
    type(gas_type), pointer :: gases_list(:)
    character(len=:), pointer :: gases_str

    print *, '########################## IAPWS VERSION ##########################'
    print *, "version ", get_version()

    print *, '########################## IAPWS R2-83 ##########################'
    print "(a, f10.3, a)", "Tc in h2o=", Tc_H2O, " k"
    print "(a, f10.3, a)", "pc in h2o=", pc_H2O, " mpa"
    print "(a, f10.3, a)", "rhoc in h2o=", rhoc_H2O, " kg/m3"

    print "(a, f10.3, a)", "Tc in D2O=", Tc_D2O, " k"
    print "(a, f10.3, a)", "pc in D2O=", pc_D2O, " mpa"
    print "(a, f10.3, a)", "rhoc in D2O=", rhoc_D2O, " kg/m3"
    print *, ''

    print *, '########################## IAPWS G7-04 ##########################'
    ! Compute kh and kd in H2O
    T(1) = 25.0_dp + 273.15_dp
    call kh(T, gas, heavywater, kh_res)
    print "(A10, 1X, A10, 1X, A2, F10.1, A, 4X, A3, SP, F10.4)", "Gas=", gas, "T=", T, "K
↪", "kh=", kh_res

    call kd(T, gas, heavywater, kd_res)
    print "(A10, 1X, A10, 1X, A2, F10.1, A, 4X, A3, SP, F15.4)", "Gas=", gas, "T=", T, "K
↪", "kh=", kd_res
```

(continues on next page)

(continued from previous page)

```
    ! Get and print available gases
    heavywater = 0
    ngas = ngases(heavywater)
    gases_list => null()
    gases_list => gases(heavywater)
    gases_str => gases2(heavywater)
    print *, "Gases in H2O: ", ngas
    print *, gases_str
    do i=1, ngas
        print *, gases_list(i)%gas
    enddo

    heavywater = 1
    ngas = ngases(heavywater)
    gases_list => null()
    gases_list => gases(heavywater)
    gases_str => gases2(heavywater)
    print *, "Gases in D2O: ", ngas
    print *, gases_str
    do i=1, ngas
        print *, gases_list(i)%gas
    enddo

    print *, '########################## IAPWS R7-97 ##########################'
    ! Compute ps from Ts.
    Ts(:) = [-1.0_dp, 25.0_dp, 100.0_dp, 200.0_dp, 300.0_dp, 360.0_dp, 374.0_dp]
    Ts(:) = Ts(:) + 273.15_dp
    call psat(Ts, ps)

    do i=1, size(Ts)
        print "(SP, F23.3, A3, 4X, F23.3, A3)", Ts(i), "K", ps(i), "MPa"
    end do

    ! Compute Ts from ps
    call Tsat(ps, Ts)
    do i=1, size(Ts)
        print "(SP, F23.3, A3, 4X, F23.3, A3)", Ts(i), "K", ps(i), "MPa"
    end do

    ! Compute water properties at 280°C/8 Mpa
    p(1) = 8.0_dp
    T(1) = 273.15_dp + 280.0_dp
    call wp(p, T, "v", wp_res)
    print "(A5, F23.16, X, A)", "v(8MPa,280°C)=", wp_res(1)*1000.0_dp, "L/kg"

    ! Compute region and phase
    x = [8.0_dp, 4.0_dp, 6.0_dp ]
    y = [553.15_dp, 1200.0_dp, 2000.0_dp]
    call wr(x, y, r)
    call wph(x, y, s)
    print *, r
```

(continues on next page)

(continued from previous page)

```
    print *, s

end program
```

2.2 C

```
#include <string.h>
#include <stdio.h>
#include "iapws.h"

int main(void){

    double T = 25.0 + 273.15; /* in C*/
    double p; /* p in Mpa */
    char *gas = "O2";
    double kh, kd, wp_res;
    char **gases_list;
    char *gases_str;
    int ngas;
    int i;
    int heavywater = 0;
    double x[3]= {8.0, 4.0, 6.0 };
    double y[3] = {553.15, 1200.0, 2000.0};
    int r[3];
    char s[3];

    printf("%s\n", "########################## IAPWS VERSION ##########################
↪");
    printf("version %s\n", iapws_get_version());

    printf("%s\n", "########################## IAPWS R2-83 ##########################");
    printf("%s %10.3f %s\n", "Tc in H2O", iapws_r283_Tc_H2O, "K");
    printf("%s %10.3f %s\n", "pc in H2O", iapws_r283_pc_H2O, "MPa");
    printf("%s %10.3f %s\n", "rhoc in H2O", iapws_r283_rhoc_H2O, "kg/m3");

    printf("%s %10.3f %s\n", "Tc in D2O", iapws_r283_Tc_D2O, "K");
    printf("%s %10.3f %s\n", "pc in D2O", iapws_r283_pc_D2O, "MPa");
    printf("%s %10.3f %s\n", "rhoc in D2O", iapws_r283_rhoc_D2O, "kg/m3");

    printf("\n");


    printf("%s\n", "########################## IAPWS G7-04 ##########################");
    /* Compute kh and kd in H2O*/
    iapws_g704_kh(&T, gas, heavywater, &kh, strlen(gas), 1);
    printf("Gas=%s\tT=%fK\tkh=%+10.4f\n", gas, T, kh);

    iapws_g704_kd(&T, gas, heavywater, &kd, strlen(gas), 1);
    printf("Gas=%s\tT=%fK\tkd=%+15.4f\n", gas, T, kd);

    /* Get and print the available gases */
```

(continues on next page)

(continued from previous page)

```
    ngas = iapws_g704_ngases(heavywater);
    gases_list = iapws_g704_gases(heavywater);
    gases_str = iapws_g704_gases2(heavywater);
    printf("Gases in H2O: %d\n", ngas);
    printf("%s\n", gases_str);
    for(i=0; i<ngas; i++){
        printf("%s\n", gases_list[i]);
    }

    heavywater = 1;
    ngas = iapws_g704_ngases(heavywater);
    gases_list = iapws_g704_gases(heavywater);
    gases_str = iapws_g704_gases2(heavywater);
    printf("Gases in D2O: %d\n", ngas);
    printf("%s\n", gases_str);
    for(i=0; i<ngas; i++){
        printf("%s\n", gases_list[i]);
    }

    printf("%s\n", "########################## IAPWS R7-97 ##########################");
    double Ts[7] =  {-1.0, 25.0, 100.0, 200.0, 300.0, 360.0, 374.0};
    double ps[7] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
    for(i=0; i<7; i++){
        Ts[i] = Ts[i] + 273.15;
    }
    iapws_r797_psat(7, Ts, ps);

    for(i=0; i<7; i++){
        printf("%+23.3f %s %+23.3f %s\n", Ts[i], "K", ps[i], "MPa");
    }

    iapws_r797_Tsat(7, ps, Ts);
    for(i=0; i<7; i++){
        printf("%+23.3f %s %+23.3f %s\n", Ts[i], "K", ps[i], "MPa");
    }

    T = 273.15 + 280.0;
    p = 8.0;
    iapws_r797_wp(&p, &T, "v", &wp_res, 1, 1);
    printf("v(8MPa,280°C) = %+23.16f L/kg\n", wp_res * 1000.0);

    iapws_r797_wr(x, y, r, 3);
    iapws_r797_wph(x, y, s, 3);
    for(i=0; i<3; i++){
        printf("%i", r[i]);
    }
    printf("\n");
    for(i=0; i<3; i++){
        printf("%c", s[i]);
    }
    printf("\n");
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

2.3 Python

```python
import array
import numpy as np
import matplotlib.pyplot as plt
import pyiapws

print("########################## IAPWS VERSION ##########################")
print(pyiapws.__version__)

print("########################## IAPWS R2-83 ##########################")
print("Tc in H2O", pyiapws.Tc_H2O, "K")
print("pc in H2O", pyiapws.pc_H2O, "MPa")
print("rhoc in H2O", pyiapws.rhoc_H2O, "kg/m3")

print("Tc in D2O", pyiapws.Tc_D2O, "K")
print("pc in D2O", pyiapws.pc_D2O, "MPa")
print("rhoc in D2O", pyiapws.rhoc_D2O, "kg/m3")

print("")

print("########################## IAPWS G7-04 ##########################")
gas  = "O2"
T = array.array("d", (25.0+273.15,))

# Compute kh and kd in H2O
heavywater = False
k = pyiapws.kh(T, "O2", heavywater)
print(f"Gas={gas}\tT={T[0]}K\tkh={k[0]:+10.4f}\n")

k = pyiapws.kd(T, "O2", heavywater)
print(f"Gas={gas}\tT={T[0]}K\tkd={k[0]:+10.4f}\n")

# Get and print the available gases
heavywater = False
gases_list = pyiapws.gases(heavywater)
gases_str = pyiapws.gases2(heavywater)
ngas = pyiapws.ngases(heavywater)
print(f"Gases in H2O: {ngas:}")
print(gases_str)
for gas in gases_list:
    print(gas)

heavywater = True
gases_list = pyiapws.gases(heavywater)
gases_str = pyiapws.gases2(heavywater)
ngas = pyiapws.ngases(heavywater)
print(f"Gases in D2O: {ngas:}")
```

(continues on next page)

(continued from previous page)

```
print(gases_str)
for gas in gases_list:
    print(gas)

style = {"marker":".", "ls":"", "ms":2}
T_KELVIN = 273.15
T = np.linspace(0.0, 360.0, 1000) + 273.15

solvent = {True: "D2O", False: "H2O"}

print("Generating plot for kh")
kname = "kh"
for HEAVYWATER in (False, True):
    print(solvent[HEAVYWATER])
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.grid(visible=True, ls=':')
    ax.set_xlabel("T /°K")
    ax.set_ylabel("ln (kh/1GPa)")
    gases = pyiapws.gases(HEAVYWATER)
    for gas in gases:
        k = pyiapws.kh(T, gas, HEAVYWATER) / 1000.0
        ln_k = np.log(k)
        ax.plot(T, ln_k, label=gas, **style)
    ax.legend(ncol=3)
    fig.savefig(f"../media/g704-{kname:s}_{solvent[HEAVYWATER]}.png", dpi=100, format=
↪"png")

print("Generating plot for kd")
kname = "kd"
for HEAVYWATER in (False, True):
    print(solvent[HEAVYWATER])
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.grid(visible=True, ls=':')
    ax.set_xlabel("T /°K")
    ax.set_ylabel("ln kd")
    gases = pyiapws.gases(HEAVYWATER)
    for gas in gases:
        k = pyiapws.kd(T, gas, HEAVYWATER)
        ln_k = np.log(k)
        ax.plot(T, ln_k, label=gas, **style)
    ax.legend(ncol=3)
    fig.savefig(f"../media/g704-{kname:s}_{solvent[HEAVYWATER]}.png", dpi=100, format=
↪"png")



print("########################## IAPWS R7-97 ##########################")
Ts = np.asarray([-1.0, 25.0, 100.0, 200.0, 300.0, 360.0, 374.0])
Ts = Ts + 273.15
```

(continues on next page)

(continued from previous page)

```python
ps = pyiapws.psat(Ts)
for i in range(Ts.size):
    print(f"{Ts[i]:23.3f} K {ps[i]:23.3f} MPa.")

Ts = pyiapws.Tsat(ps)
for i in range(Ts.size):
    print(f"{Ts[i]:23.3f} K {ps[i]:23.3f} MPa.")

fig = plt.figure()
ax = fig.add_subplot()
ax.grid(visible=True, ls=':')
ax.set_xlabel("Ts /K")
ax.set_ylabel("ps /MPa")
Ts = np.linspace(0.0, 370.0, 500)
Ts = Ts + 273.15

ps = pyiapws.psat(Ts)
ax.plot(Ts, ps, "r-", label="ps(Ts)")

Ts = pyiapws.Tsat(ps)
ax.plot(Ts, ps, "b--", label="Ts(ps)")

ax.legend()
fig.savefig(f"../media/r797-r4.png", dpi=100, format="png")


T = 280.0 + 273.15
p = 8.0
res = pyiapws.wp(p, T, "v")*1000.0
print(f"v(8MPa,280°C) = {res:+23.16f} L/kg)")

x = np.asarray([8.0, 4.0, 6.0 ])
y = np.asarray([553.15, 1200.0, 2000.0])
r=pyiapws.wr(x, y)
s=pyiapws.wph(x, y)
print(r)
print(s)


plt.show()
```

APIS

3.1 Fortran

https://milanskocic.github.io/iapws/ford/index.html

3.2 C API

```
#ifndef IAPWS_H
#define IAPWS_H

#if _MSC_VER
#define ADD_IMPORT __declspec(dllimport)
#else
#define ADD_IMPORT
#endif

extern char* iapws_get_version(void);

ADD_IMPORT extern const double iapws_r283_Tc_H2O;
ADD_IMPORT extern const double iapws_r283_Tc_D2O;

ADD_IMPORT extern const double iapws_r283_pc_H2O;
ADD_IMPORT extern const double iapws_r283_pc_D2O;

ADD_IMPORT extern const double iapws_r283_rhoc_H2O;
ADD_IMPORT extern const double iapws_r283_rhoc_D2O;

extern void iapws_g704_kh(double *T, char *gas, int heavywater, double *k, int size_gas,␣
↪size_t size_T);
extern void iapws_g704_kd(double *T, char *gas, int heavywater, double *k, int size_gas,␣
↪size_t size_T);
extern int iapws_g704_ngases(int heavywater);
extern char **iapws_g704_gases(int heavywater);
extern char *iapws_g704_gases2(int heavywater);



extern void iapws_r797_psat(size_t N, double *Ts, double *ps);
extern void iapws_r797_Tsat(size_t N, double *ps, double *Ts);
extern void iapws_r797_wp(double *p, double  *T, char *prop, double *res, size_t N, size_
```

(continues on next page)

(continued from previous page)

```
↪t len);
extern void iapws_r797_wr(double *p, double *T, int *res, size_t N);
extern void iapws_r797_wph(double *p, double *T, char *res, size_t N);



extern void iapws_r1124_Kw(size_t N, double *T, double *rhow, double *k);

#endif
```

3.3 Python

Python wrapper of the (Modern Fortran) iapws library.

`Kw`(*T: ndarray*, *rhow: ndarray*)

Compute the ionization constant of water Kw. Validity range 273.13 K <= T <= 1273.15 K and 0 <= p <= 1000 MPa.

Parameters

T: int, float or 1d-array.
Temperature in K.

rhow: int, float or 1d-array.
Mass density in g.cm-3.

Returns

k: float or 1d-array
Ionization constant.

`Tsat`(*ps*)

Compute the saturation temperature at pressure ps. Validity range 611.213 Pa <= ps <= 22.064 MPa.

Parameters

ps: int, float or 1d-array.
Saturation pressure in MPa.

Returns

Ts: float or 1d-array
Saturation temperatur in K. Is NaN if Ts is out of range of validity.

`gases`(*heavywater: bool = False*) → List[str]

Get the list of available gases.

Parameters

heavywater: bool, optional.
Flag for indicating if solvent is heavywater (True) or water (False). Default to False.

Returns

gases: list of str
List of available gases.

`gases2`(*heavywater: bool = False*) → str

Get the available gases as a string.

Parameters

heavywater: bool, optional.
Flag for indicating if solvent is heavywater (True) or water (False). Default to False.

Returns

gases: str
Available gases as comma separated string.

`kd`(*T: ndarray*, *gas: str*, *heavywater: bool = False*) → ndarray | float

Get the vapor-liquid constant for gas in H2O or D2O at T. If gas not found returns NaNs.

Parameters

T: int, float, or 1d-array.
Temperature in K.

gas: str
Gas.

heavywater: bool, optional.
Flag for indicating if solvent is heavywater (True) or water (False). Default to False.

Returns

kd: float or 1d-array
Adimensional liquid-vapor constant.

`kh`(*T: ndarray*, *gas: str*, *heavywater: bool = False*) → ndarray | float

Get the Henry constant for gas in H2O or D2O at T. If gas not found returns NaNs.

Parameters

T: int, float or 1d-array.
Temperature in K.

gas: str
Gas.

heavywater: bool, optional.
Flag for indicating if solvent is heavywater (True) or water (False). Default to False.

Returns

kh: float or 1d-array
Henry constant in MPa.

`ngases`(*heavywater: bool = False*) → int

Get the number of available gases.

Parameters

heavywater: bool, optional.
Flag for indicating if solvent is heavywater (True) or water (False). Default to False.

Returns

n: int
Number of available gases in water or heavywater.

`psat(Ts)`

Compute the saturation pressure at temperature Ts. Validity range 273.13 K <= Ts <= 647.096 K.

Parameters

Ts: int, float or 1d-array.
Saturation temperature in K.

Returns

ps: float or 1d-array
Saturation pressure in MPa. Is NaN if Ts is out of range of validity.

`wp(p, T, prop)`

Compute water properties at pressure p in MPa and temperature T in Kelvin. The adequate region is selected according to p and T.

Available properties:

- v: specific volume in m3/kg
- u: specific internal energy in kJ/kg
- s: specific entropy in kJ/kg
- h: specific enthalpy in kJ/kg/K
- cp: specific isobaric heat capacity in kJ/kg/K
- cv: specific isochoric heat capacity in kJ/kg/K
- w: speed of sound in m/s

Parameters

p: int, float or 1d-array.
Pressure in MPa.

T: int, float or 1d-array.
Temperature in K.

prop: str

Water property:

- v: specific volume in m3/kg
- u: specific internal energy in kJ/kg
- s: specific entropy in kJ/kg
- h: specific enthalpy in kJ/kg/K
- cp: specific isobaric heat capacity in kJ/kg/K
- cv: specific isochoric heat capacity in kJ/kg/K
- w: speed of sound in m/s

Returns
----
res: float or 1d-array
Computed property. Filled with NaN if no adequate region is found.

wph(*p*, *T*)

Get the water phase corresponding to p and T.

Parameters

p: int, float or 1d-array.
Pressure in MPa.

T: int, float or 1d-array.
Temperature in K.

Returns

region: str or 1d-array
Phases: l(liquid), v(VAPOR), c(SUPER CRITICAL), s(SATURATION), n(UNKNOWN).

wr(*p*, *T*)

Get the water region corresponding to p and T.

Parameters

p: int, float or 1d-array.
Pressure in MPa.

T: int, float or 1d-array.
Temperature in K.

Returns

region: int or 1d-array
Regions 1 to 5 or -1 when not found.

CHAPTER

FOUR

CHANGELOG

4.1 0.5.1

- Refactoring the `configure.sh` script.
- Remove support for 3.14t. No official release on python.org.
- If binaries for Python 3.14t are needed you need to compile them by yourself.

4.2 0.5.0

- Remove support for Python 3.9 and add support for Python 3.14(t).

Full changelog

4.3 0.4.1

- Drop support for python 3.8 and add support for python 3.13.
- Code cleaning in python C extensions.
- Code refactoring in pure python modules for encapsulating C extensions.
- Implementation of region 1 from R797.
- Implementation of `Kw` from R1124.
- Add API for water properties `wp`, water region `wr` and water phase `wp`.
- Documentation update.

Full changelog available at github

4.4 0.4.0

- Implementation of the region 4 in R7-97.
- API break for kh and kd in g704. The temperature must be provided in Kelvin instead of degrees Celsius.
- Add dependency to numpy for python wrapper.
- Add pure python modules for encapsulating C extensions.
- Refactoring and code cleaning.
- Documentation update.

Full changelog available at github

4.5 0.3.0

- API break: functions for the Fortran code were renamed:
  - They do not contain the package+module in the name for the sake of simplicity
  - The package is only added in the functions for the C API in order to have a namespace-like behavior.
  - If needed for solving conflicts with other packages, the functions can be aliased.
- Separate sources files for the C API code for each module.
- Implement tests with the test-drive framework.
- Add version extension in the pywrapper.
- Implement version module with its getter.
- Documentation update.

Full changelog available at github

4.6 0.2.2

- Implementation of report R283 for critical constants of water.
- Switch to pyproject.toml for python wrapper.
- Code refactoring and clean up.
- Documentation update.

Full changelog available at github

4.7 0.2.1

- Comlete missing documentation of private functions.
- Minor fixes in C API code as well in python wrapper.
- Remove unecessary dependency in Makefile.

Full changelog available at github

4.8 0.2.0

- New structure with modules corresponding to the IAPWS papers.
- Compatible with fpm.
- fpm module naming convention.
- API break for iapws_g704_kh and iapws_g704_kd functions:
  - only 1d-arrays as inputs in Fortran and C API.
  - only objects with buffer protocol as inputs in python wrapper.
  - python wrappers return memoryviews.
- New functions:
  - providing the number of gases in H2O and D2O.

  - providing the available of gases in H2O and D2O as list of strings.
  - providing the available of gases in H2O and D2O as a unique string.
- Cleanup old app code not needed anymore.
- Fix memory allocation in pywrapper.
- Completed tests.
- Documentation improvements:
  - Add conversion equations from molar fractions to solubilities.
  - Add plots for visualizing kh and kd.

Full changelog available at github

4.9 0.1.1

- Logo creation
- Error handling in python wrapper for arrays with rank greater than 1
- Tests in python wrapper for expected failures with rank-n arrays

Full changelog available at github

4.10 0.1.0

- Implementation of kH and kD from IAPWS G7-04 in fortran + C API
- Python wrapper for kH and kD.
- Documentation with sphinx.

Full changelog available at github

BIBLIOGRAPHY

[1] IAPWS. Release on the Values of Temperature, Pressure and Density of Ordinary and Heavy Water Substances at Their Respective Critical Points. Technical Report R2-83, IAPWS, 1992.

[2] IAPWS. Guideline on the Henry's Constant and Vapor-Liquid Distribution Constant for Gases in H2O and D2O at High Temperatures. Technical Report G7-04, IAPWS, 2004.

[3] IAPWS. Revised Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam. Technical Report R7-97, IAPWS, 2012.

[4] Wolfgang Wagner and A. Pruss. International Equations for the Saturation Properties of Ordinary Water Substance. Revised According to the International Temperature Scale of 1990. Addendum to J. Phys. Chem. Ref. Data 16, 893 (1987). *Journal of Physical and Chemical Reference Data*, 22(3):783–787, 1993-05-01. URL: https://doi.org/10.1063/1.555926, doi:10.1063/1.555926.

[5] Allan H. Harvey and Eric W. Lemmon. Correlation for the Vapor Pressure of Heavy Water From the Triple Point to the Critical Point. *Journal of Physical and Chemical Reference Data*, 31(1):173–181, 2002-03-01. URL: https://doi.org/10.1063/1.1430231, doi:10.1063/1.1430231.

[6] W. Wagner and A. Pruß. The IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use. *Journal of Physical and Chemical Reference Data*, 31(2):387–535, 2002-06-07. URL: https://doi.org/10.1063/1.1461829, doi:10.1063/1.1461829.

[7] R. Fernandez-Prini, J.L. Alvarez, and A.H. Harvey. Henry's Constants and Vapor–Liquid Distribution Constants for Gaseous Solutes in H2O and D2O at High Temperatures. *Journal of Physical Chemistry Reference Data*, 32(2):903–916, 2003.

[8] Andrei V. Bandura and Serguei N. Lvov. The ionization constant of water over wide ranges of temperature and density. *Journal of Physical and Chemical Reference Data*, 35(1):15–30, 12 2005. URL: https://doi.org/10.1063/1.1928231, arXiv:https://pubs.aip.org/aip/jpr/article-pdf/35/1/15/16717791/15_1_online.pdf, doi:10.1063/1.1928231.

[9] IAPWS. Revised Release on the IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use. Technical Report R6-95, IAPWS, 2018.

[10] IAPWS. Revised release on the ionization constant of h2o. Technical Report R11-24, IAPWS, 2012.

PYTHON MODULE INDEX

INDEX