
iapws Documentation

Release 0.2.0

M. Skocic

Jun 19, 2023

CONTENTS:

1	Getting Started	1
2	IAPWS - Theoretical background	7
3	Release Notes	11
4	API	15
	Bibliography	39
	Python Module Index	41
	Index	43

GETTING STARTED

Sources: <https://github.com/MilanSkocic/iapws>

1.1 iapw



ipaws is a Fortran library providing the formulas for computing light and heavy water properties. It also provides a API for the C language. The formulas are taken from <http://iapws.org>.

1.1.1 How to install

A Makefile is provided which uses `fpm` for building the library with additional options:

- compile the source generator and generate the sources
- copy needed sources into the python wrapper folder
- build a shared library
- install the C headers
- uninstall the library and headers

On windows, `msys2` needs to be installed and use the `mingw64` or `mingw32` terminals.

On Darwin, the `gcc` toolchain needs to be installed.

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

```
source configuration
make
```

Run tests

```
fpm test
```

Install

```
make install
```

Uninstall

```
make uninstall
```

If building the python wrapper is needed:

```
cd pywrapper
python setup.py bdist_wheel
```

1.1.2 Dependencies

```
gcc>=9.0
gfortran>=9.0
fpm>=0.7
```

1.1.3 License

GNU General Public License v3 (GPLv3)

1.2 pyiapws

Python wrapper around the [Fortran iapws library](#). The Fortran library does not need to be installed, the python wrapper embeds all needed dependencies. On linux, you might have to install *libgfortran* if it is not distributed with your linux distribution.

1.2.1 How to install

```
pip install pyiapws
```

1.2.2 Dependencies

1.2.3 License

GNU General Public License v3 (GPLv3)

1.3 Examples

1.3.1 Example in Fortran

```
program example_in_f
  use iso_fortran_env
  use iapws_g704
  implicit none
  integer(int32) :: i, ngas
  real(real64) :: T(1), kh(1), kd(1)
  character(len=2) :: gas = "O2"
  integer(int32) :: heavywater = 0
  type(iapws_g704_gas_t), pointer :: gases_list(:)
  character(len=:), pointer :: gases_str
```

(continues on next page)

(continued from previous page)

```

! Compute kh and kd in H2O
T(1) = 25.0d0
call iapws_g704_kh(T, gas, heavywater, kh)
print "(A10, 1X, A10, 1X, A2, F10.1, A, 4X, A3, SP, F10.4)", "Gas=", gas, "T=", T,
↪ "C", "kh=", kh

call iapws_g704_kd(T, gas, heavywater, kd)
print "(A10, 1X, A10, 1X, A2, F10.1, A, 4X, A3, SP, F15.4)", "Gas=", gas, "T=", T,
↪ "C", "kh=", kd

! Get and print available gases
heavywater = 0
ngas = iapws_g704_ngases(heavywater)
gases_list => null()
gases_list => iapws_g704_gases(heavywater)
gases_str => iapws_g704_gases2(heavywater)
print *, "Gases in H2O: ", ngas
print *, gases_str
do i=1, ngas
    print *, gases_list(i)%gas
enddo

heavywater = 1
ngas = iapws_g704_ngases(heavywater)
gases_list => null()
gases_list => iapws_g704_gases(heavywater)
gases_str => iapws_g704_gases2(heavywater)
print *, "Gases in D2O: ", ngas
print *, gases_str
do i=1, ngas
    print *, gases_list(i)%gas
enddo

end program

```

1.3.2 Example in C

```

#include <string.h>
#include <stdio.h>
#include "iapws_g704.h"

int main(void){

    double T = 25.0; /* in C*/
    char *gas = "O2";
    double kh, kd;
    char **gases_list;
    char *gases_str;
    int ngas;
    int i;
    int heavywater = 0;

    /* Compute kh and kd in H2O*/

```

(continues on next page)

(continued from previous page)

```

iapws_g704_capi_kh(&T, gas, heavywater, &kh, strlen(gas), 1);
printf("Gas=%s\tT=%fC\tkh=%+10.4f\n", gas, T, kh);

iapws_g704_capi_kd(&T, gas, heavywater, &kd, strlen(gas), 1);
printf("Gas=%s\tT=%fC\tkd=%+15.4f\n", gas, T, kd);

/* Get and print the available gases */
ngas = iapws_g704_capi_ngases(heavywater);
gases_list = iapws_g704_capi_gases(heavywater);
gases_str = iapws_g704_capi_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_capi_ngases(heavywater);
gases_list = iapws_g704_capi_gases(heavywater);
gases_str = iapws_g704_capi_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]);
}

return 0;
}

```

1.3.3 Example in Python

```

r"""Example in python"""
import array
import pyiapws

gas = "O2"
T = array.array("d", (25.0,))

# Compute kh and kd in H2O
heavywater = False
m = pyiapws.g704.kh(T, "O2", heavywater)
k = array.array("d", m)
print(f"Gas={gas}\tT={T[0]}C\tkh={k[0]:+10.4f}\n")

m = pyiapws.g704.kd(T, "O2", heavywater)
k = array.array("d", m)
print(f"Gas={gas}\tT={T[0]}C\tkh={k[0]:+10.4f}\n")

# Get and print the available gases
heavywater = False
gases_list = pyiapws.g704.gases(heavywater)
gases_str = pyiapws.g704.gases2(heavywater)
ngas = pyiapws.g704.ngases(heavywater)
print(f"Gases in H2O: {ngas}")

```

(continues on next page)

(continued from previous page)

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

heavywater = True
gases_list = pyiapws.g704.gases(heavywater)
gases_str = pyiapws.g704.gases2(heavywater)
ngas = pyiapws.g704.ngases(heavywater)
print(f"Gases in D20: {ngas}")
print(gases_str)
for gas in gases_list:
    print(gas)
```


IAPWS - THEORETICAL BACKGROUND

2.1 IAPWS G7-04

The computation is based on the parameters provided by the IAPWS 2004 [1].

2.1.1 Henry Contant: k_H

The Henry constant k_H is defined as shown in equation Eq.2.1.1. k_H is expressed in MPa.

$$k_H = \lim_{x_2 \rightarrow 0} f_2/x_2 \quad (2.1.1)$$

- f_2 : liquid-phase fugacity
- x_2 : mole fraction of the solute

The Henry's constant k_H is given as a function of temperature by:

$$\ln \left(\frac{k_H}{p_1^*} \right) = A/T_R + \frac{B \cdot \tau^{0.355}}{T_R} + C \cdot T_R^{-0.41} \cdot \exp \tau \quad (2.1.2)$$

- $\tau = 1 - T_R$
- $T_R = T/T_{c1}$
- T_{c1} : critical temperature of the solvent as recommended by IAPWS [2]
- p_1^* is the vapor pressure of the solvent at the temperature of interest and is calculated from the correlation of Wagner and Pruss for H_2O [3] and from the correlation of Harvey and Lemmon for D_2O [4].

Both equations have the form:

$$\ln (p_1^*/p_{c1}) = T_R^{-1} \sum_{i=1}^n a_i \tau^{b_i} \quad (2.1.3)$$

- n is 6 for H_2O and 5 for D_2O
- p_{c1} is the critical pressure of the solvent recommended by IAPWS [2]

2.1.2 Vapor-Liquid Distribution Constant: k_D

The liquid-vapor distribution constant k_D is defined as shown in equation Eq.2.1.4. K_D is adimensional.

$$k_D = \lim_{x_2 \rightarrow 0} y_2/x_2 \quad (2.1.4)$$

- x_2 : mole fraction of the solute
- y_2 is the vapor-phase solute mole fraction in equilibrium with the liquid

The vapor-liquid distribution constant k_D is given as a function of temperature by:

$$\ln K_D = qF + \frac{E}{T(K)} f(\tau) + (F + G\tau^{2/3} + H\tau) \exp\left(\frac{273.15 - T(K)}{100}\right) \quad (2.1.5)$$

- q : -0.023767 for H_2O and -0.024552 for D_2O .
- $f(\tau)$ [3] for H_2O and [5] for D_2O .

In both cases, $f(\tau)$ has the following form:

$$f(\tau) = \sum_{i=1}^n c_i \cdot \tau^{d_i} \quad (2.1.6)$$

- n is 6 for H_2O and 4 for D_2O

2.1.3 Molar fractions

The molar fractions x_2 and y_2 can be expressed from the equations Eq.2.1.1 and Eq.2.1.4 as shown in Eq.2.1.7.

$$\begin{aligned} x_2 &= \frac{f_2}{k_H} \\ \frac{x_2}{f_2} &= \frac{1}{k_H} \\ y_2 &= \frac{k_D}{k_H} \cdot f_2 \\ \frac{y_2}{f_2} &= \frac{k_D}{k_H} \end{aligned} \quad (2.1.7)$$

By fixing f_2 at 1.0 it comes that the molar fractions x_2 and y_2 are then expressed per unit of pressure as shown in equation Eq.2.1.8 .

$$\begin{aligned} x_2 &= \frac{1}{k_H} \\ y_2 &= \frac{k_D}{k_H} \end{aligned} \quad (2.1.8)$$

The molar fractions can be converted to solubilities in ppm or cm³/kg as shown in equation Eq.2.1.9 by considering dilute solutions. X is the considered gas and the solvent is either H_2O or D_2O .

$$\begin{aligned} S_X[mg.kg^{-1}.bar^{-1}] &= x_2[bar^{-1}] \cdot \frac{M_X[g.mol^{-1}]}{M_{solvent}[g.mol^{-1}]} \cdot 10^6 \\ S_X[cm^3.kg^{-1}.bar^{-1}] &= \frac{S_X[mg.kg^{-1}.bar^{-1}]}{M_X[g.mol^{-1}]} \cdot V_m[mol.L^{-1}] \end{aligned} \quad (2.1.9)$$

Available gases

kh and kd can be computed for the following gases:

- 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

Plots

The evolution of kh in H_2O and D_2O , between 0°C and 360°C , are shown in figures Fig. 2.1.1 and Fig. 2.1.2.

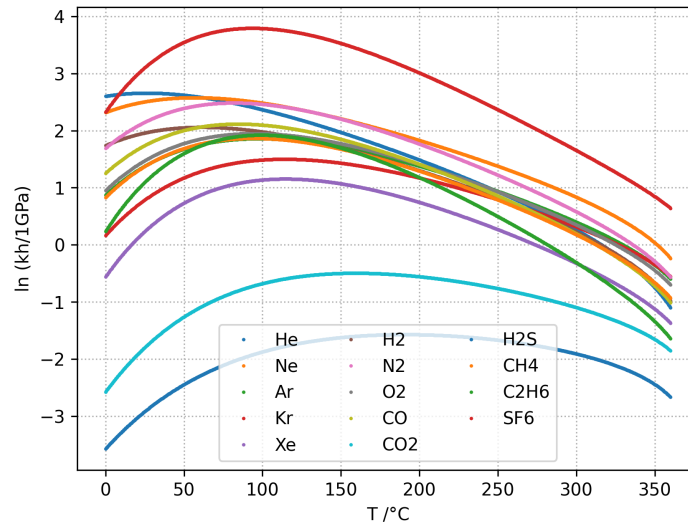


Fig. 2.1.1: kh in H_2O

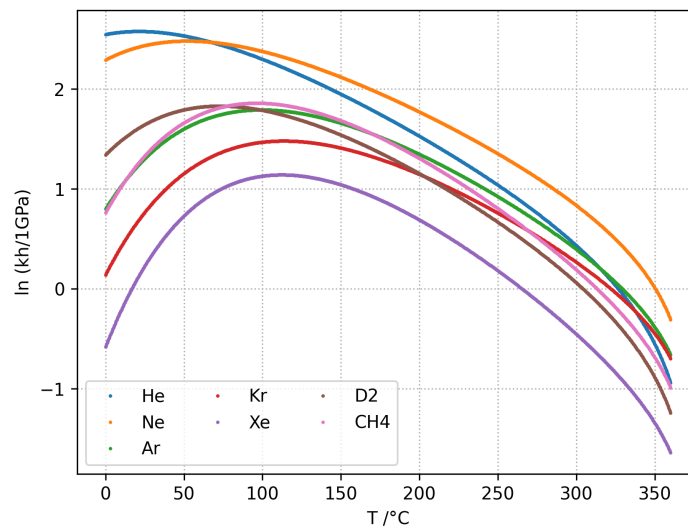
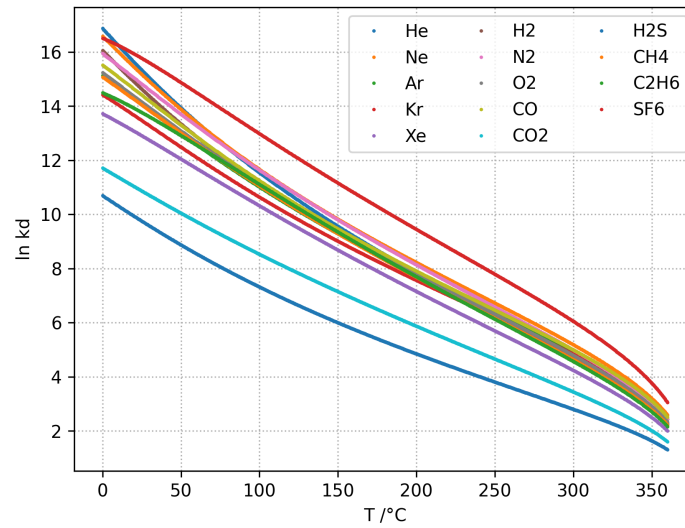
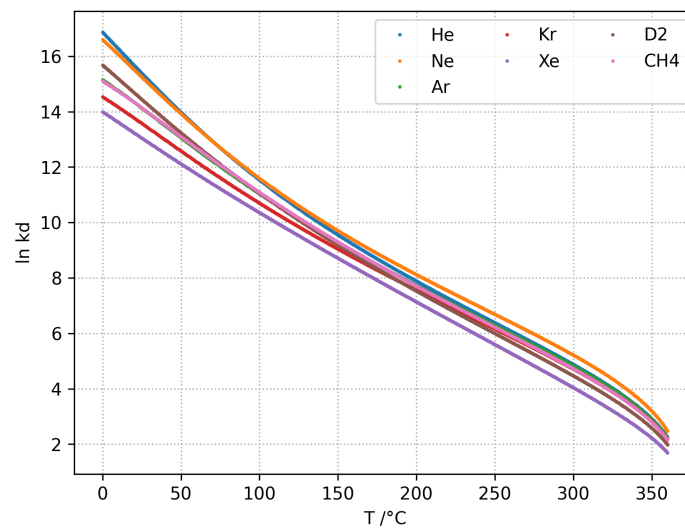


Fig. 2.1.2: kh in D_2O

The evolution of kd in H_2O and D_2O , between 0°C and 360°C , are shown in figures Fig. 2.1.3 and Fig. 2.1.4.

Fig. 2.1.3: k_h in H_2O Fig. 2.1.4: k_d in D_2O

RELEASE NOTES

3.1 iapws 0.2.0 Release Note

3.1.1 Summary

- 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.

3.1.2 Download

iapws

pyiapws

3.1.3 Contributors

Milan Skocic

3.1.4 Commits

Full Changelog: <https://github.com/MilanSkocic/pyiapws/compare/0.1.1...0.2.0>

3.2 iapws 0.1.1 Release Note

3.2.1 Summary

- 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

3.2.2 Download

iapws

pyiapws

3.2.3 Contributors

Milan Skocic

3.2.4 Commits

Full Changelog: <https://github.com/MilanSkocic/pyiapws/compare/0.1.0...0.1.1>

3.3 iapws 0.1.0 Release Note

3.3.1 Changes

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

3.3.2 Download

iapws

pyiapws

3.3.3 Contributors

Milan Skocic

3.3.4 Commits

Full Changelog: <https://github.com/MilanSkocic/pyiapws/compare/...0.1.0>

4.1 iapws

4.1.1 Fortran

- *iapws.f90*: Main module for the whole library.

```
module iapws_g704
  !! Module for IAPWS G7-04
  use iso_fortran_env
  use ieee_arithmetic
  implicit none
  private

  integer(int32), parameter :: lengas = 5
  integer(int32), parameter :: ngas_H2O = 14
  integer(int32), parameter :: ngas_D2O = 7

  type :: iapws_g704_gas_t
    !! Derived type containing a allocatable string for representing a gas.
    character(len=:), allocatable :: gas !! Gas
  end type
  type(iapws_g704_gas_t), allocatable, target :: f_gases(:)
  character(len=:), allocatable, target :: f_gases_str

  !> Absolute temperature in KELVIN
  real(real64), parameter :: T_KELVIN = 273.15d0

  !! Parameters from IAPWS G7-04
  !> critical temperature of water in K
  real(real64), parameter :: Tc1_H2O = 647.096d0
  !> critical pressure of the water in K
  real(real64), parameter :: pc1_H2O = 22.064d0
  !> critical temperature of heavy water MPa
  real(real64), parameter :: Tc1_D2O = 643.847d0
  !> critical pressure of heavywater MPa
  real(real64), parameter :: pc1_D2O = 21.671d0

  !> solvent coefficient for kd in water
  real(real64), parameter :: q_H2O = -0.023767d0
  !> solvent coefficient for kd in heavywater
  real(real64), parameter :: q_D2O = -0.024552d0

  !! ABC coefficients for gases in water.
```

(continues on next page)

(continued from previous page)

```

type :: abc_t
  character(len=lengas) :: gas
  real(real64) :: A
  real(real64) :: B
  real(real64) :: C
end type

type :: efgh_t
  character(len=lengas) :: gas
  real(real64) :: E
  real(real64) :: F
  real(real64) :: G
  real(real64) :: H
end type

!> ai and bi coefficients for water
real(real64), dimension(6, 2), parameter :: aibi_H2O = reshape([&
-7.85951783d0, 1.84408259d0, -11.78664970d0, 22.68074110d0, -15.96187190d0, 1.
↪80122502d0,&
1.000d0, 1.500d0, 3.000d0, 3.500d0, 4.000d0, 7.500d0], [6,2])

!> ai and bi coefficients for heavywater
real(real64), dimension(5, 2), parameter :: aibi_D2O = reshape([&
-7.8966570d0, 24.7330800d0, -27.8112800d0, 9.3559130d0, -9.2200830d0, &
1.00d0, 1.89d0, 2.00d0, 3.00d0, 3.60d0], [5, 2])

!> ABC constants water.
type(abc_t), dimension(ngas_H2O), parameter :: abc_H2O = &
[abc_t("He", -3.52839d0, 7.12983d0, 4.47770d0),&
abc_t("Ne", -3.18301d0, 5.31448d0, 5.43774d0),&
abc_t("Ar", -8.40954d0, 4.29587d0, 10.52779d0),&
abc_t("Kr", -8.97358d0, 3.61508d0, 11.29963d0),&
abc_t("Xe", -14.21635d0, 4.00041d0, 15.60999d0),&
abc_t("H2", -4.73284d0, 6.08954d0, 6.06066d0),&
abc_t("N2", -9.67578d0, 4.72162d0, 11.70585d0),&
abc_t("O2", -9.44833d0, 4.43822d0, 11.42005d0),&
abc_t("CO", -10.52862d0, 5.13259d0, 12.01421d0),&
abc_t("CO2", -8.55445d0, 4.01195d0, 9.52345d0),&
abc_t("H2S", -4.51499d0, 5.23538d0, 4.42126d0),&
abc_t("CH4", -10.44708d0, 4.66491d0, 12.12986d0),&
abc_t("C2H6", -19.67563d0, 4.51222d0, 20.62567d0),&
abc_t("SF6", -16.56118d0, 2.15289d0, 20.35440d0)]

!> ABC constants for heavywater
type(abc_t), dimension(ngas_D2O), parameter :: abc_D2O = &
[abc_t("He", -0.72643d0, 7.02134d0, 2.04433d0),&
abc_t("Ne", -0.91999d0, 5.65327d0, 3.17247d0),&
abc_t("Ar", -7.17725d0, 4.48177d0, 9.31509d0),&
abc_t("Kr", -8.47059d0, 3.91580d0, 10.69433d0),&
abc_t("Xe", -14.46485d0, 4.42330d0, 15.60919d0),&
abc_t("D2", -5.33843d0, 6.15723d0, 6.53046d0),&
abc_t("CH4", -10.01915d0, 4.73368d0, 11.75711d0)]

!> ci and di coefficients for water
real(real64), dimension(6, 2), parameter :: cidi_H2O = reshape([&
1.99274064d0, 1.09965342d0, -0.510839303d0, -1.75493479d0, -45.5170352d0, -6.

```

(continues on next page)

(continued from previous page)

```

↪ 7469445d5,&
1.0d0/3.0d0, 2.0d0/3.0d0, 5.0d0/3.0d0, 16.0d0/3.0d0, 43.0d0/3.0d0, 110.0d0/3.0d0], [6,
↪ 2])

!> ci and di coefficients for heavywater
real(real64), dimension(4, 2), parameter :: cidi_D20 = reshape([&
2.7072d0, 0.58662d0, -1.3069d0, -45.663d0, &
0.374d0, 1.45d0, 2.6d0, 12.3d0], [4,2])

!> EFGH constants for water
type(efgh_t), dimension(ngas_H2O), parameter :: efgh_H2O = &
[efgh_t("He", 2267.4082d0, -2.9616d0, -3.2604d0, 7.8819d0),&
efgh_t("Ne", 2507.3022d0, -38.6955d0, 110.3992d0, -71.9096d0),&
efgh_t("Ar", 2310.5463d0, -46.7034d0, 160.4066d0, -118.3043d0),&
efgh_t("Kr", 2276.9722d0, -61.1494d0, 214.0117d0, -159.0407d0),&
efgh_t("Xe", 2022.8375d0, 16.7913d0, -61.2401d0, 41.9236d0),&
efgh_t("H2", 2286.4159d0, 11.3397d0, -70.7279d0, 63.0631d0),&
efgh_t("N2", 2388.8777d0, -14.9593d0, 42.0179d0, -29.4396d0),&
efgh_t("O2", 2305.0674d0, -11.3240d0, 25.3224d0, -15.6449d0),&
efgh_t("CO", 2346.2291d0, -57.6317d0, 204.5324d0, -152.6377d0),&
efgh_t("CO2", 1672.9376d0, 28.1751d0, -112.4619d0, 85.3807d0),&
efgh_t("H2S", 1319.1205d0, 14.1571d0, -46.8361d0, 33.2266d0),&
efgh_t("CH4", 2215.6977d0, -0.1089d0, -6.6240d0, 4.6789d0),&
efgh_t("C2H6", 2143.8121d0, 6.8859d0, -12.6084d0, 0.0d0),&
efgh_t("SF6", 2871.7265d0, -66.7556d0, 229.7191d0, -172.7400d0)]

!> EFGH constants for heavywater
type(efgh_t), dimension(ngas_D2O), parameter :: efgh_D2O = &
[efgh_t("He", 2293.2474d0, -54.7707d0, 194.2924d0, -142.1257), &
efgh_t("Ne", 2439.6677d0, -93.4934d0, 330.7783d0, -243.0100d0),&
efgh_t("Ar", 2269.2352d0, -53.6321d0, 191.8421d0, -143.7659d0),&
efgh_t("Kr", 2250.3857d0, -42.0835d0, 140.7656d0, -102.7592d0),&
efgh_t("Xe", 2038.3656d0, 68.1228d0, -271.3390d0, 207.7984d0),&
efgh_t("D2", 2141.3214d0, -1.9696d0, 1.6136d0, 0.0d0),&
efgh_t("CH4", 2216.0181d0, -40.7666d0, 152.5778d0, -117.7430d0)]

public :: iapws_g704_gas_t
public :: iapws_g704_kh, iapws_g704_kd
public :: iapws_g704_ngases
public :: iapws_g704_gases, iapws_g704_gases2

contains

!> @brief Find the index of the gas in the ABC table.
!! @param[in] gas Gas.
!! @param[in] abc ABC table.
pure function findgas_abc(gas, abc)result(value)
  implicit none
  !! arguments
  character(len=*) intent(in) :: gas
  type(abc_t), dimension(:), intent(in) :: abc
  !! returns
  integer(int32) :: value
  !! local variables
  integer(int32) :: i

```

(continues on next page)

(continued from previous page)

```

    value = 0

    do i=1, size(abc)
        if(trim(gas) .eq. abc(i)%gas)then
            value = i
            exit
        endif
    end do
end function

!> @brief Find the index of the gas in the ABC table.
!! @param[in] gas Gas.
!! @param[in] efgh ABC table.
pure function findgas_efgh(gas, efgh)result(value)
    implicit none
    !! arguments
    character(len=*) , intent(in) :: gas
    type(efgh_t), dimension(:), intent(in) :: efgh
    !! returns
    integer(int32) :: value
    !! local variables
    integer(int32) :: i

    value = 0

    do i=1, size(efgh)
        if(trim(gas) .eq. efgh(i)%gas)then
            value = i
            exit
        endif
    end do
end function

pure elemental function f_p1star_H2O(T)result(value)
    implicit none
    !! arguments
    real(real64), intent(in) :: T
    !! return
    real(real64) :: value
    !! variables
    real(real64) :: Tr
    real(real64) :: tau

    Tr = (T+T_KELVIN)/Tc1_H2O
    tau = 1 - Tr
    value = exp(1/(Tr) * sum(aibi_H2O(:,1)*tau**(aibi_H2O(:,2)))) * pc1_H2O
end function

pure elemental function f_p1star_D2O(T)result(value)
    implicit none
    !! arguments
    real(real64), intent(in) :: T
    !! return
    real(real64) :: value
    !! variables
    real(real64) :: Tr

```

(continues on next page)

(continued from previous page)

```

real(real64) :: tau

Tr = (T+T_KELVIN)/Tc1_D20
tau = 1 - Tr
value = exp(1/(Tr) * sum(aibi_D20(:,1)*tau**(aibi_D20(:,2)))) * pc1_D20
end function

pure elemental function f_kh_p1star_H2O(T, abc)result(value)
  !! arguments
  real(real64), intent(in) :: T
  type(abc_t), intent(in) :: abc
  !! return
  real(real64) :: value
  !! variables
  real(real64) :: Tr
  real(real64) :: tau

  Tr = (T+T_KELVIN)/Tc1_H2O
  tau = 1 - Tr
  value = exp(abc%A/Tr + abc%B*(tau**0.355d0)/Tr + abc%C*exp(tau)*Tr**(-0.41d0))
end function

pure elemental function f_kh_p1star_D2O(T, abc)result(value)
  !! arguments
  real(real64), intent(in) :: T
  type(abc_t), intent(in) :: abc
  !! return
  real(real64) :: value
  !! variables
  real(real64) :: Tr
  real(real64) :: tau

  Tr = (T+T_KELVIN)/Tc1_D20
  tau = 1 - Tr
  value = exp(abc%A/Tr + abc%B*(tau**0.355d0)/Tr + abc%C*exp(tau)*Tr**(-0.41d0))
end function

pure elemental function ft_H2O(tau)result(value)
  implicit none
  !! arguments
  real(real64), intent(in) :: tau
  !! return
  real(real64) :: value
  value = sum(cidi_H20(:,1) * tau**(cidi_H20(:,2)))
end function

pure elemental function ft_D2O(tau)result(value)
  implicit none
  !! arguments
  real(real64), intent(in) :: tau
  !! return
  real(real64) :: value
  value = sum(cidi_D20(:,1) * tau**(cidi_D20(:,2)))
end function

pure elemental function f_kh_H2O(T, abc)result(value)

```

(continues on next page)

(continued from previous page)

```

implicit none
!! arguments
real(real64), intent(in) :: T
type(abc_t), intent(in) :: abc
!! returns
real(real64) :: value
value = f_kh_p1star_H2O(T, abc) * f_p1star_H2O(T)
end function

pure elemental function f_kh_D20(T, abc) result(value)
implicit none
!! arguments
real(real64), intent(in) :: T
type(abc_t), intent(in) :: abc
!! returns
real(real64) :: value
value = f_kh_p1star_D20(T, abc) * f_p1star_D20(T)
end function

pure elemental function f_kd_H2O(T, efgh) result(value)
implicit none
!! arguments
real(real64), intent(in) :: T
type(efgh_t), intent(in) :: efgh
!! returns
real(real64) :: value
!! local variables
real(real64) :: Tr
real(real64) :: tau
real(real64) :: p1
real(real64) :: p2
real(real64) :: p3
real(real64) :: p4

Tr = (T+T_KELVIN)/Tc1_H2O
tau = 1-Tr

p1 = q_H2O*efgh%F
p2 = efgh%E/(T+T_KELVIN)*ft_H2O(tau)
p3 = (efgh%F + efgh%G*tau**(2.0d0/3.0d0) + efgh%H*tau)
p4 = exp(-T/100.0d0)

value = exp(p1 + p2 + p3 * p4)
end function

pure elemental function f_kd_D20(T, efgh) result(value)
implicit none
!! arguments
real(real64), intent(in) :: T
type(efgh_t), intent(in) :: efgh
!! returns
real(real64) :: value
!! local variables
real(real64) :: Tr
real(real64) :: tau

```

(continues on next page)

(continued from previous page)

```

real(real64) :: p1
real(real64) :: p2
real(real64) :: p3
real(real64) :: p4

Tr = (T+T_KELVIN)/Tc1_D20
tau = 1-Tr

p1 = q_D20*efgh%F
p2 = efgh%E/(T+T_KELVIN)*ft_D20(tau)
p3 = (efgh%F + efgh%G*tau**(2.0d0/3.0d0) + efgh%H*tau)
p4 = exp(-T/100.0d0)

value = exp(p1 + p2 + p3 * p4)
end function

pure subroutine iapws_g704_kh(T, gas, heavywater, k)
  !! Compute the henry constant for a given temperature.
  implicit none

  ! arguments
  real(real64), intent(in) :: T(:)
  !! Temperature in °C.
  character(len=*), intent(in) :: gas
  !! Gas.
  integer(int32), intent(in) :: heavywater
  !! Flag if D2O (1) is used or H2O(0).
  real(real64), intent(out) :: k(:)
  !! Henry constant. Filled with NaNs if gas not found.

  ! variables
  integer(int32) :: i

  if(heavywater > 0)then
    i = findgas_abc(gas, abc_D20)
    if(i==0)then
      k = ieee_value(1.0d0, ieee_quiet_nan)
    else
      k = f_kh_D20(T, abc_D20(i))
    endif
  else
    i = findgas_abc(gas, abc_H2O)
    if(i==0)then
      k = ieee_value(1.0d0, ieee_quiet_nan)
    else
      k = f_kh_H2O(T, abc_H2O(i))
    endif
  endif
end subroutine

pure subroutine iapws_g704_kd(T, gas, heavywater, k)
  !! Compute the vapor-liquid constant for a given temperature.
  implicit none

```

(continues on next page)

(continued from previous page)

```

! arguments
real(real64), intent(in) :: T(:)
!! Temperature in °C.
character(len=*), intent(in) :: gas
!! Gas.
integer(int32), intent(in) :: heavywater
!! Flag if D2O (1) is used or H2O(0).
real(real64), intent(out) :: k(:)
!! Vapor-liquid constant. Filled with NaNs if gas not found.

! variables
integer(int32) :: i

if(heavywater > 0)then
  i = findgas_efgh(gas, efgh_D20)
  if(i==0)then
    k = ieee_value(1.0d0, ieee_quiet_nan)
  else
    k = f_kd_D20(T, efgh_D20(i))
  endif
else
  i = findgas_efgh(gas, efgh_H2O)
  if(i==0)then
    k = ieee_value(1.0d0, ieee_quiet_nan)
  else
    k = f_kd_H2O(T, efgh_H2O(i))
  endif
endif

end subroutine

pure function iapws_g704_ngases(heavywater)result(n)
!! Returns the number of gases.
implicit none

! arguments
integer(int32), intent(in) :: heavywater
!! Flag if D2O (1) is used or H2O(0).
integer(int32) :: n
!! Number of gases.

if(heavywater > 0)then
  n = ngas_D20
else
  n = ngas_H2O
endif

end function

function iapws_g704_gases(heavywater)result(gases)
!! Returns the list of available gases.
implicit none

! arguments
integer(int32), intent(in) :: heavywater
!! Flag if D2O (1) is used or H2O(0).
type(iapws_g704_gas_t), pointer :: gases(:)

```

(continues on next page)

(continued from previous page)

```

    !! Available gases.

    ! variables
    integer(int32) :: i, n

    if(allocated(f_gases))then
        deallocate(f_gases)
    endif

    if(heavywater > 0)then
        allocate(f_gases(ngas_D20))
        do i=1, ngas_D20
            if(allocated(f_gases(i)%gas))then
                deallocate(f_gases(i)%gas)
            endif
            n = len(trim(abc_D20(i)%gas))
            allocate(character(len=n) :: f_gases(i)%gas)
            f_gases(i)%gas = trim(abc_D20(i)%gas)
        enddo
    else
        allocate(f_gases(ngas_H2O))
        do i=1, ngas_H2O
            if(allocated(f_gases(i)%gas))then
                deallocate(f_gases(i)%gas)
            endif
            n = len(trim(abc_H2O(i)%gas))
            allocate(character(len=n) :: f_gases(i)%gas)
            f_gases(i)%gas = trim(abc_H2O(i)%gas)
        enddo
    endif
    gases => f_gases
end function

function iapws_g704_gases2(heavywater)result(gases)
    !! Returns the available gases as a string.
    implicit none

    ! arguments
    integer(int32), intent(in) :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
    character(len=:), pointer :: gases
    !! Available gases

    ! variables
    integer(int32) :: i, j, k, ngas
    type(iapws_g704_gas_t), pointer :: f_gases(:)

    f_gases => iapws_g704_gases(heavywater)
    ngas = size(f_gases)

    k = 0
    do i=1, ngas
        k = k + len(f_gases(i)%gas)
    enddo

    if(allocated(f_gases_str))then

```

(continues on next page)

(continued from previous page)

```

        deallocate(f_gases_str)
    endif
    allocate(character(len=k+ngas) :: f_gases_str)

    i = 1
    j = 1
    k = 1
    do i=1, ngas
        do j=1, len(f_gases(i)%gas)
            f_gases_str(k:k) = f_gases(i)%gas(j:j)
            k = k + 1
        enddo
        f_gases_str(k:k) = ","
        k = k + 1
    enddo
    f_gases_str(len(f_gases_str):len(f_gases_str)) = ""
    gases => f_gases_str

end function

end module

```

IAPWS G704: Gas solubilities

- *iapws_g704.f90*: Module for IAPWS G7-04

```

module iapws_g704
    !! Module for IAPWS G7-04
    use iso_fortran_env
    use ieee_arithmetic
    implicit none
    private

    integer(int32), parameter :: lengas = 5
    integer(int32), parameter :: ngas_H2O = 14
    integer(int32), parameter :: ngas_D2O = 7

    type :: iapws_g704_gas_t
        !! Derived type containing a allocatable string for representing a gas.
        character(len=:), allocatable :: gas !! Gas
    end type
    type(iapws_g704_gas_t), allocatable, target :: f_gases(:)
    character(len=:), allocatable, target :: f_gases_str

    !> Absolute temperature in KELVIN
    real(real64), parameter :: T_KELVIN = 273.15d0

    !! Parameters from IAPWS G7-04
    !> critical temperature of water in K
    real(real64), parameter :: Tc1_H2O = 647.096d0
    !> critical pressure of the water in K
    real(real64), parameter :: pc1_H2O = 22.064d0
    !> critical temperature of heavy water MPa
    real(real64), parameter :: Tc1_D2O = 643.847d0
    !> critical pressure of heavywater MPa

```

(continues on next page)

(continued from previous page)

```

real(real64), parameter :: pc1_D20 = 21.671d0

!> solvent coefficient for kd in water
real(real64), parameter :: q_H2O = -0.023767d0
!> solvent coefficient for kd in heavywater
real(real64), parameter :: q_D2O = -0.024552d0

!! ABC coefficients for gases in water.
type :: abc_t
    character(len=lengas) :: gas
    real(real64) :: A
    real(real64) :: B
    real(real64) :: C
end type

type :: efgh_t
    character(len=lengas) :: gas
    real(real64) :: E
    real(real64) :: F
    real(real64) :: G
    real(real64) :: H
end type

!> ai and bi coefficients for water
real(real64), dimension(6, 2), parameter :: aibi_H2O = reshape([&
-7.85951783d0, 1.84408259d0, -11.78664970d0, 22.68074110d0, -15.96187190d0, 1.
↪80122502d0,&
1.000d0, 1.500d0, 3.000d0, 3.500d0, 4.000d0, 7.500d0], [6,2])

!> ai and bi coefficients for heavywater
real(real64), dimension(5, 2), parameter :: aibi_D2O = reshape([&
-7.8966570d0, 24.7330800d0, -27.8112800d0, 9.3559130d0, -9.2200830d0, &
1.00d0, 1.89d0, 2.00d0, 3.00d0, 3.60d0], [5, 2])

!> ABC constants water.
type(abc_t), dimension(ngas_H2O), parameter :: abc_H2O = &
[abc_t("He", -3.52839d0, 7.12983d0, 4.47770d0),&
abc_t("Ne", -3.18301d0, 5.31448d0, 5.43774d0),&
abc_t("Ar", -8.40954d0, 4.29587d0, 10.52779d0),&
abc_t("Kr", -8.97358d0, 3.61508d0, 11.29963d0),&
abc_t("Xe", -14.21635d0, 4.00041d0, 15.60999d0),&
abc_t("H2", -4.73284d0, 6.08954d0, 6.06066d0),&
abc_t("N2", -9.67578d0, 4.72162d0, 11.70585d0),&
abc_t("O2", -9.44833d0, 4.43822d0, 11.42005d0),&
abc_t("CO", -10.52862d0, 5.13259d0, 12.01421d0),&
abc_t("CO2", -8.55445d0, 4.01195d0, 9.52345d0),&
abc_t("H2S", -4.51499d0, 5.23538d0, 4.42126d0),&
abc_t("CH4", -10.44708d0, 4.66491d0, 12.12986d0),&
abc_t("C2H6", -19.67563d0, 4.51222d0, 20.62567d0),&
abc_t("SF6", -16.56118d0, 2.15289d0, 20.35440d0)]

!> ABC constants for heavywater
type(abc_t), dimension(ngas_D2O), parameter :: abc_D2O = &
[abc_t("He", -0.72643d0, 7.02134d0, 2.04433d0),&
abc_t("Ne", -0.91999d0, 5.65327d0, 3.17247d0),&
abc_t("Ar", -7.17725d0, 4.48177d0, 9.31509d0),&

```

(continues on next page)

(continued from previous page)

```

    abc_t("Kr", -8.47059d0, 3.91580d0, 10.69433d0),&
    abc_t("Xe", -14.46485d0, 4.42330d0, 15.60919d0),&
    abc_t("D2", -5.33843d0, 6.15723d0, 6.53046d0),&
    abc_t("CH4", -10.01915d0, 4.73368d0, 11.75711d0)]

!> ci and di coefficients for water
real(real64), dimension(6, 2), parameter :: cidi_H2O = reshape([&
1.99274064d0, 1.09965342d0, -0.510839303d0, -1.75493479d0, -45.5170352d0, -6.
↪ 7469445d5,&
1.0d0/3.0d0, 2.0d0/3.0d0, 5.0d0/3.0d0, 16.0d0/3.0d0, 43.0d0/3.0d0, 110.0d0/3.0d0], [6,
↪ 2])

!> ci and di coefficients for heavywater
real(real64), dimension(4, 2), parameter :: cidi_D2O = reshape([&
2.7072d0, 0.58662d0, -1.3069d0, -45.663d0, &
0.374d0, 1.45d0, 2.6d0, 12.3d0], [4,2])

!> EFGH constants for water
type(efgh_t), dimension(ngas_H2O), parameter :: efgh_H2O = &
[efgh_t("He", 2267.4082d0, -2.9616d0, -3.2604d0, 7.8819d0),&
efgh_t("Ne", 2507.3022d0, -38.6955d0, 110.3992d0, -71.9096d0),&
efgh_t("Ar", 2310.5463d0, -46.7034d0, 160.4066d0, -118.3043d0),&
efgh_t("Kr", 2276.9722d0, -61.1494d0, 214.0117d0, -159.0407d0),&
efgh_t("Xe", 2022.8375d0, 16.7913d0, -61.2401d0, 41.9236d0),&
efgh_t("H2", 2286.4159d0, 11.3397d0, -70.7279d0, 63.0631d0),&
efgh_t("N2", 2388.8777d0, -14.9593d0, 42.0179d0, -29.4396d0),&
efgh_t("O2", 2305.0674d0, -11.3240d0, 25.3224d0, -15.6449d0),&
efgh_t("CO", 2346.2291d0, -57.6317d0, 204.5324d0, -152.6377d0),&
efgh_t("CO2", 1672.9376d0, 28.1751d0, -112.4619d0, 85.3807d0),&
efgh_t("H2S", 1319.1205d0, 14.1571d0, -46.8361d0, 33.2266d0),&
efgh_t("CH4", 2215.6977d0, -0.1089d0, -6.6240d0, 4.6789d0),&
efgh_t("C2H6", 2143.8121d0, 6.8859d0, -12.6084d0, 0.0d0),&
efgh_t("SF6", 2871.7265d0, -66.7556d0, 229.7191d0, -172.7400d0)]

!> EFGH constants for heavywater
type(efgh_t), dimension(ngas_D2O), parameter :: efgh_D2O = &
[efgh_t("He", 2293.2474d0, -54.7707d0, 194.2924d0, -142.1257), &
efgh_t("Ne", 2439.6677d0, -93.4934d0, 330.7783d0, -243.0100d0),&
efgh_t("Ar", 2269.2352d0, -53.6321d0, 191.8421d0, -143.7659d0),&
efgh_t("Kr", 2250.3857d0, -42.0835d0, 140.7656d0, -102.7592d0),&
efgh_t("Xe", 2038.3656d0, 68.1228d0, -271.3390d0, 207.7984d0),&
efgh_t("D2", 2141.3214d0, -1.9696d0, 1.6136d0, 0.0d0),&
efgh_t("CH4", 2216.0181d0, -40.7666d0, 152.5778d0, -117.7430d0)]

public :: iapws_g704_gas_t
public :: iapws_g704_kh, iapws_g704_kd
public :: iapws_g704_ngases
public :: iapws_g704_gases, iapws_g704_gases2

contains

!> @brief Find the index of the gas in the ABC table.
!! @param[in] gas Gas.
!! @param[in] abc ABC table.
pure function findgas_abc(gas, abc)result(value)
    implicit none

```

(continues on next page)

(continued from previous page)

```

!! arguments
character(len=*), intent(in) :: gas
type(abc_t), dimension(:), intent(in) :: abc
!! returns
integer(int32) :: value
!! local variables
integer(int32) :: i

value = 0

do i=1, size(abc)
    if(trim(gas) .eq. abc(i)%gas)then
        value = i
        exit
    endif
end do
end function

!> @brief Find the index of the gas in the ABC table.
!! @param[in] gas Gas.
!! @param[in] efgh ABC table.
pure function findgas_efgh(gas, efgh)result(value)
    implicit none
    !! arguments
    character(len=*), intent(in) :: gas
    type(efgh_t), dimension(:), intent(in) :: efgh
    !! returns
    integer(int32) :: value
    !! local variables
    integer(int32) :: i

    value = 0

    do i=1, size(efgh)
        if(trim(gas) .eq. efgh(i)%gas)then
            value = i
            exit
        endif
    end do
end function

pure elemental function f_p1star_H2O(T)result(value)
    implicit none
    !! arguments
    real(real64), intent(in) :: T
    !! return
    real(real64) :: value
    !! variables
    real(real64) :: Tr
    real(real64) :: tau

    Tr = (T+T_KELVIN)/Tc1_H2O
    tau = 1 - Tr
    value = exp(1/(Tr) * sum(aibi_H2O(:,1)*tau**(aibi_H2O(:,2)))) * pc1_H2O
end function

```

(continues on next page)

(continued from previous page)

```

pure elemental function f_p1star_D20(T)result(value)
  implicit none
  !! arguments
  real(real64), intent(in) :: T
  !! return
  real(real64) :: value
  !! variables
  real(real64) :: Tr
  real(real64) :: tau

  Tr = (T+T_KELVIN)/Tc1_D20
  tau = 1 - Tr
  value = exp(1/(Tr) * sum(aibi_D20(:,1)*tau**(aibi_D20(:,2)))) * pc1_D20
end function

pure elemental function f_kh_p1star_H2O(T, abc)result(value)
  !! arguments
  real(real64), intent(in) :: T
  type(abc_t), intent(in) :: abc
  !! return
  real(real64) :: value
  !! variables
  real(real64) :: Tr
  real(real64) :: tau

  Tr = (T+T_KELVIN)/Tc1_H2O
  tau = 1 - Tr
  value = exp(abc%A/Tr + abc%B*(tau**0.355d0)/Tr + abc%C*exp(tau)*Tr**(-0.41d0))
end function

pure elemental function f_kh_p1star_D20(T, abc)result(value)
  !! arguments
  real(real64), intent(in) :: T
  type(abc_t), intent(in) :: abc
  !! return
  real(real64) :: value
  !! variables
  real(real64) :: Tr
  real(real64) :: tau

  Tr = (T+T_KELVIN)/Tc1_D20
  tau = 1 - Tr
  value = exp(abc%A/Tr + abc%B*(tau**0.355d0)/Tr + abc%C*exp(tau)*Tr**(-0.41d0))
end function

pure elemental function ft_H20(tau)result(value)
  implicit none
  !! arguments
  real(real64), intent(in) :: tau
  !! return
  real(real64) :: value
  value = sum(cidi_H20(:,1) * tau**(cidi_H20(:,2)))
end function

pure elemental function ft_D20(tau)result(value)
  implicit none

```

(continues on next page)

(continued from previous page)

```

    !! arguments
    real(real64), intent(in) :: tau
    !! return
    real(real64) :: value
    value = sum(cidi_D20(:,1) * tau**(cidi_D20(:,2)))
end function

pure elemental function f_kh_H20(T, abc) result(value)
    implicit none
    !! arguments
    real(real64), intent(in) :: T
    type(abc_t), intent(in) :: abc
    !! returns
    real(real64) :: value
    value = f_kh_p1star_H20(T, abc) * f_p1star_H20(T)
end function

pure elemental function f_kh_D20(T, abc) result(value)
    implicit none
    !! arguments
    real(real64), intent(in) :: T
    type(abc_t), intent(in) :: abc
    !! returns
    real(real64) :: value
    value = f_kh_p1star_D20(T, abc) * f_p1star_D20(T)
end function

pure elemental function f_kd_H20(T, efgh) result(value)
    implicit none
    !! arguments
    real(real64), intent(in) :: T
    type(efgh_t), intent(in) :: efgh
    !! returns
    real(real64) :: value
    !! local variables
    real(real64) :: Tr
    real(real64) :: tau
    real(real64) :: p1
    real(real64) :: p2
    real(real64) :: p3
    real(real64) :: p4

    Tr = (T+T_KELVIN)/Tc1_H20
    tau = 1-Tr

    p1 = q_H20*efgh%F
    p2 = efgh%E/(T+T_KELVIN)*ft_H20(tau)
    p3 = (efgh%F + efgh%G*tau**(2.0d0/3.0d0) + efgh%H*tau)
    p4 = exp(-T/100.0d0)

    value = exp(p1 + p2 + p3 * p4)
end function

pure elemental function f_kd_D20(T, efgh) result(value)
    implicit none

```

(continues on next page)

(continued from previous page)

```

!! arguments
real(real64), intent(in) :: T
type(efgh_t), intent(in) :: efgh
!! returns
real(real64) :: value
!! local variables
real(real64) :: Tr
real(real64) :: tau
real(real64) :: p1
real(real64) :: p2
real(real64) :: p3
real(real64) :: p4

Tr = (T+T_KELVIN)/Tc1_D20
tau = 1-Tr

p1 = q_D20*efgh%F
p2 = efgh%E/(T+T_KELVIN)*ft_D20(tau)
p3 = (efgh%F + efgh%G*tau**(2.0d0/3.0d0) + efgh%H*tau)
p4 = exp(-T/100.0d0)

value = exp(p1 + p2 + p3 * p4)
end function

pure subroutine iapws_g704_kh(T, gas, heavywater, k)
!! Compute the henry constant for a given temperature.
implicit none

! arguments
real(real64), intent(in) :: T(:)
!! Temperature in °C.
character(len=*), intent(in) :: gas
!! Gas.
integer(int32), intent(in) :: heavywater
!! Flag if D2O (1) is used or H2O(0).
real(real64), intent(out) :: k(:)
!! Henry constant. Filled with NaNs if gas not found.

! variables
integer(int32) :: i

if(heavywater > 0)then
  i = findgas_abc(gas, abc_D20)
  if(i==0)then
    k = ieee_value(1.0d0, ieee_quiet_nan)
  else
    k = f_kh_D20(T, abc_D20(i))
  endif
else
  i = findgas_abc(gas, abc_H2O)
  if(i==0)then
    k = ieee_value(1.0d0, ieee_quiet_nan)
  else
    k = f_kh_H20(T, abc_H20(i))
  endif
endif

```

(continues on next page)

(continued from previous page)

```

endif

end subroutine

pure subroutine iapws_g704_kd(T, gas, heavywater, k)
  !! Compute the vapor-liquid constant for a given temperature.
  implicit none

  ! arguments
  real(real64), intent(in) :: T(:)
  !! Temperature in °C.
  character(len=*), intent(in) :: gas
  !! Gas.
  integer(int32), intent(in) :: heavywater
  !! Flag if D2O (1) is used or H2O(0).
  real(real64), intent(out) :: k(:)
  !! Vapor-liquid constant. Filled with NaNs if gas not found.

  ! variables
  integer(int32) :: i

  if(heavywater > 0)then
    i = findgas_efgh(gas, efgh_D20)
    if(i==0)then
      k = ieee_value(1.0d0, ieee_quiet_nan)
    else
      k = f_kd_D20(T, efgh_D20(i))
    endif
  else
    i = findgas_efgh(gas, efgh_H2O)
    if(i==0)then
      k = ieee_value(1.0d0, ieee_quiet_nan)
    else
      k = f_kd_H2O(T, efgh_H2O(i))
    endif
  endif
endif

end subroutine

pure function iapws_g704_ngases(heavywater)result(n)
  !! Returns the number of gases.
  implicit none

  ! arguments
  integer(int32), intent(in) :: heavywater
  !! Flag if D2O (1) is used or H2O(0).
  integer(int32) :: n
  !! Number of gases.

  if(heavywater > 0)then
    n = ngas_D20
  else
    n = ngas_H2O
  endif
end function

```

(continues on next page)

(continued from previous page)

```

function iapws_g704_gases(heavywater)result(gases)
  !! Returns the list of available gases.
  implicit none

  ! arguments
  integer(int32), intent(in) :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
  type(iapws_g704_gas_t), pointer :: gases(:)
    !! Available gases.

  ! variables
  integer(int32) :: i, n

  if(allocated(f_gases))then
    deallocate(f_gases)
  endif

  if(heavywater > 0)then
    allocate(f_gases(ngas_D20))
    do i=1, ngas_D20
      if(allocated(f_gases(i)%gas))then
        deallocate(f_gases(i)%gas)
      endif
      n = len(trim(abc_D20(i)%gas))
      allocate(character(len=n) :: f_gases(i)%gas)
      f_gases(i)%gas = trim(abc_D20(i)%gas)
    enddo
  else
    allocate(f_gases(ngas_H2O))
    do i=1, ngas_H2O
      if(allocated(f_gases(i)%gas))then
        deallocate(f_gases(i)%gas)
      endif
      n = len(trim(abc_H2O(i)%gas))
      allocate(character(len=n) :: f_gases(i)%gas)
      f_gases(i)%gas = trim(abc_H2O(i)%gas)
    enddo
  endif
  gases => f_gases
end function

function iapws_g704_gases2(heavywater)result(gases)
  !! Returns the available gases as a string.
  implicit none

  ! arguments
  integer(int32), intent(in) :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
  character(len=:), pointer :: gases
    !! Available gases

  ! variables
  integer(int32) :: i, j, k, ngas
  type(iapws_g704_gas_t), pointer :: f_gases(:)

  f_gases => iapws_g704_gases(heavywater)

```

(continues on next page)

(continued from previous page)

```

ngas = size(f_gases)

k = 0
do i=1, ngas
    k = k + len(f_gases(i)%gas)
enddo

if(allocated(f_gases_str))then
    deallocate(f_gases_str)
endif
allocate(character(len=k+ngas) :: f_gases_str)

i = 1
j = 1
k = 1
do i=1, ngas
    do j=1, len(f_gases(i)%gas)
        f_gases_str(k:k) = f_gases(i)%gas(j:j)
        k = k + 1
    enddo
    f_gases_str(k:k) = ","
    k = k + 1
enddo
f_gases_str(len(f_gases_str):len(f_gases_str)) = ""
gases => f_gases_str

end function

end module

```

- *iapws_g704.f90*: C API for the IAPWS module.

```

module iapws_g704_capi
    !! C API for the IAPWS module.
    use iso_fortran_env
    use iso_c_binding
    use iapws_g704
    implicit none
    private

    type, bind(C) :: c_char_p
        type(c_ptr) :: p
    end type
    type :: capi_gas_t
        character(kind=c_char, len=1), allocatable :: gas(:)
    end type
    type(capi_gas_t), allocatable, target :: c_gases(:)
    type(c_char_p), allocatable, target :: char_pp(:)
    character(len=:), allocatable, target :: c_gases_str

    public :: iapws_g704_capi_kh, iapws_g704_capi_kd
    public :: iapws_g704_capi_ngases
    public :: iapws_g704_capi_gases

contains

```

(continues on next page)

(continued from previous page)

```

subroutine iapws_g704_capi_kh(T, gas, heavywater, k, size_gas, size_T)bind(C)
  !! Compute the henry constant for a given temperature.
  implicit none

  ! arguments
  type(c_ptr), value :: T
    !! Temperature in °C.
  type(c_ptr), intent(in), value :: gas
    !! Gas.
  integer(c_int), intent(in), value :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
  type(c_ptr), intent(in), value :: k
    !! Henry constant. Filled with NaNs if gas not found.
  integer(c_int), intent(in), value :: size_gas
    !! Size of the gas string.
  integer(c_size_t), intent(in), value :: size_T
    !! Size of T and k.

  ! variables
  character, pointer, dimension(:) :: c2f_gas
  real(real64), pointer :: f_T(:)
  character(len=size_gas) :: f_gas
  real(real64), pointer :: f_k(:)
  integer(int32) :: i

  call c_f_pointer(gas, c2f_gas, shape=[size_gas])
  call c_f_pointer(T, f_T, shape=[size_T])
  call c_f_pointer(k, f_k, shape=[size_T])

  do i=1, size_gas
    f_gas(i:i) = c2f_gas(i)
  enddo
  call iapws_g704_kh(f_T, f_gas, heavywater, f_k)
end subroutine

subroutine iapws_g704_capi_kd(T, gas, heavywater, k, size_gas, size_T)bind(C)
  !! Compute the vapor-liquid constant for a given temperature.
  implicit none

  ! arguments
  type(c_ptr), value :: T
    !! Temperature in °C.
  type(c_ptr), intent(in), value :: gas
    !! Gas.
  integer(c_int), intent(in), value :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
  type(c_ptr), intent(in), value :: k
    !! Vapor-liquid constant. Filled with NaNs if gas not found.
  integer(c_int), intent(in), value :: size_gas
    !! Size of the gas string.
  integer(c_size_t), intent(in), value :: size_T
    !! Size of T and k.

  ! variables
  character, pointer, dimension(:) :: c2f_gas
  real(real64), pointer :: f_T(:)

```

(continues on next page)

(continued from previous page)

```

character(len=size_gas) :: f_gas
real(real64), pointer :: f_k(:)
integer(int32) :: i

call c_f_pointer(gas, c2f_gas, shape=[size_gas])
call c_f_pointer(T, f_T, shape=[size_T])
call c_f_pointer(k, f_k, shape=[size_T])

do i=1, size_gas
    f_gas(i:i) = c2f_gas(i)
enddo
call iapws_g704_kd(f_T, f_gas, heavywater, f_k)
end subroutine

pure function iapws_g704_capi_ngases(heavywater)bind(C)result(n)
    !! Returns the number of gases.
    implicit none

    ! arguments
    integer(c_int), intent(in), value :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
    integer(c_int) :: n
    !! Number of gases.

    n = iapws_g704_ngases(heavywater)
end function

function iapws_g704_capi_gases(heavywater)bind(C)result(gases)
    !! Returns the list of available gases.
    implicit none

    ! arguments
    integer(c_int), intent(in), value :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
    type(c_ptr) :: gases
    !! Available gases.

    ! variables
    integer(int32) :: i, j, ngas, n

    type(iapws_g704_gas_t), pointer :: f_gases(:) => null()
    f_gases => iapws_g704_gases(heavywater)
    ngas = size(f_gases)

    if(allocated(c_gases))then
        deallocate(c_gases)
    endif
    allocate(c_gases(ngas))

    if(allocated(char_pp))then
        deallocate(char_pp)
    endif
    allocate(char_pp(ngas))

    do i=1, ngas
        if(allocated(c_gases(i)%gas))then

```

(continues on next page)

(continued from previous page)

```

        deallocate(c_gases(i)%gas)
    endif
    n = len(f_gases(i)%gas)
    allocate(c_gases(i)%gas(n+1))
    do j=1, n
        c_gases(i)%gas(j) = f_gases(i)%gas(j:j)
    enddo
    c_gases(i)%gas(n+1) = c_null_char
    char_pp(i)%p = c_loc(c_gases(i)%gas)
enddo
gases = c_loc(char_pp)
end function

function iapws_g704_capi_gases2(heavywater)bind(C)result(gases)
    !! Returns the available gases as a string.
    implicit none

    ! arguments
    integer(c_int), intent(in), value :: heavywater
    !! Flag if D2O (1) is used or H2O(0).
    type(c_ptr) :: gases
    !! Available gases.

    ! variables
    character(len=:), pointer :: f_gases_str => null()
    f_gases_str => iapws_g704_gases2(heavywater)

    if(allocated(c_gases_str))then
        deallocate(c_gases_str)
    endif
    allocate(character(len=len(f_gases_str)) :: c_gases_str)

    c_gases_str = f_gases_str
    c_gases_str(len(f_gases_str):len(f_gases_str)) = c_null_char

    gases = c_loc(c_gases_str)
end function

end module

```

4.1.2 C

- *iapws.h*: Main C header for the whole library.

```

/**
 * @file iapws.h
 * @brief Main C header for the IAPWS library.
 */
#ifndef IAPWS_H
#define IAPWS_H
#include "iapws_g704.h"
#endif

```


IAPWS G704: Gas solubilities

- `iapws_g704.h`: C header.

```
/**
 * @file iapws_g704.h
 * @brief C header for the module iapws_g704.
 */

#ifndef IAPWS_G704_H
#define IAPWS_G704_H

extern void iapws_g704_capi_kh(double *T, char *gas, int heavywater, double *k, int_
↪size_gas, size_t size_T);
extern void iapws_g704_capi_kd(double *T, char *gas, int heavywater, double *k, int_
↪size_gas, size_t size_T);
extern int iapws_g704_capi_ngases(int heavywater);
extern char **iapws_g704_capi_gases(int heavywater);
extern char *iapws_g704_capi_gases2(int heavywater);

#endif
```

4.2 pyipaws

4.2.1 IAPWS G704: Gas solubilities

C extension wrapping the `iapws_g704` module of the Fortran `iapws` library.

`pyiapws.g704.gases()`

`gases(heavywater: bool) → tuple`

Get the available gases.

`pyiapws.g704.gases2()`

`gases(heavywater: bool) → str`

Get the available gases as a string.

`pyiapws.g704.kd()`

`kd(T: array, gas, heavywater: bool) → mview`

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

`pyiapws.g704.kh()`

`kh(T: array, gas: str, heavywater: bool) → mview`

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

`pyiapws.g704.ngases()`

`gases(heavywater: bool) → int`

Get the number of available gases.

BIBLIOGRAPHY

- [1] IAPWS. Guideline on the Henry's Constant and Vapor-Liquid Distribution Constant for Gases in H_2O and D_2O at High Temperatures. Technical Report G7-04, IAPWS, Kyoto, Japan, 2004.
- [2] IAPWS. Revised Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam. Technical Report R7-97, IAPWS, Lucerne, Switzerland, 2007.
- [3] 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, May 1993. doi:10.1063/1.555926.
- [4] 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, March 2002. doi:10.1063/1.1430231.
- [5] R. Fernandez-Prini, J.L. Alvarez, and A.H. Harvey. Henry's Constants and Vapor–Liquid Distribution Constants for Gaseous Solutes in H_2O and D_2O at High Temperatures. *Journal of Physical Chemistry Reference Data*, 32(2):903–916, 2003.

PYTHON MODULE INDEX

p

`pyiapws.g704`, [37](#)

INDEX

G

`gases()` (*in module `pyiapws.g704`*), 37

`gases2()` (*in module `pyiapws.g704`*), 37

K

`kd()` (*in module `pyiapws.g704`*), 37

`kh()` (*in module `pyiapws.g704`*), 37

M

module

`pyiapws.g704`, 37

N

`ngases()` (*in module `pyiapws.g704`*), 37

P

`pyiapws.g704`

module, 37