pyXSteam Documentation

Release 0.4.8

drunsinn, Magnus Holmgren

Mar 19, 2022

Contents:

Original Released by Magnus Holmgren for Matlab and Excel: <http://xsteam.sourceforge.net> and/or <http://www.x-eng.com>

XSteam provides (mostly) accurate steam and water properties from 0 - 1000 bar and from 0 - 2000 °C according to the [IAPWS release IF-97](http://www.iapws.org/relguide/IF97-Rev.pdf). For accuracy of the functions in different regions see IF-97 Page 4

Also includes thermal conductivity and viscosity, which are not part of the IF97 release. * Thermal Conductivity: (IAPWS 1998)

This Python Library is based on the original XSteam Library for Matlab and Excel from Magnus Holmgren, www.x-eng.com. We take no responsibilities for any errors in the code or damage thereby! See README.md for examples

Some effort has been made to include the refined function of more recent releases and also functions for calculations on heavy water. This includes: * IAPWS R4 * IAPWS R14

CHAPTER 1

Notes

1.1 Density (rho)

Density is calculated as 1/v. See section 1.5 Volume

1.2 Viscosity

Viscosity is not part of IAPWS Steam IF97. Equations from "Revised Release on the IAPWS Formulation 1985 for the Viscosity of Ordinary Water Substance", 2003 are used. Viscosity in the mixed region (4) is interpolated according to the density. This is not true since it will be two phases.

1.3 Thermal conductivity

Revised release on the IAPS Formulation 1985 for the Thermal Conductivity of ordinary water substance (IAPWS 1998)

1.4 Nomenclature

All Functions follow the same naming schema: First the wanted property, then a underscore _, then the wanted input properties Example: *t_ph* is temperature as a function of pressure and enthalpy. For a list of valid functions se below:

Property	Description
t	Temperature (°C or °F)
p	Pressure (bar or psi)
h	Enthalpy (kJ/kg or btu/lb)
v	Specific volume (m3/kg or ft^3/lb)|
rho	Density (kg/m3 or lb/ft^3)
s	Specific entropy (kJ/(kg °C) or btu/(lb °F))
u	Specific internal energy (kJ/kg or btu/lb)
Cp	Specific isobaric heat capacity (kJ/(kg °C) or btu/(lb °F))
Cv	Specific isochoric heat capacity (kJ/(kg °C) or btu/(lb °F))
w	Speed of sound (m/s or ft/s)
my	Viscosity (N s/m^2 or lbm/ft/hr)
tc	Thermal Conductivity (W/(m °C) or btu/(h ft °F))
st	Surface Tension (N/m or lb/ft)
x	Vapor fraction
vx	Vapor Volume Fraction

1.4.1 Available Functions

Temperature

Function	Description
tsat_p	Saturation temperature
t_ph	Temperature as a function of pressure and enthalpy
t_ps	Temperature as a function of pressure and entropy
t_hs	Temperature as a function of enthalpy and entropy

Pressure

Function	Description
psat_t	Saturation pressure
p_hs	Pressure as a function of h and s.
p_hrho	Pressure as a function of h and rho. Very inaccurate for solid water region since it's almost incompressible!

Enthalpy

Function	Description
hV_p	Saturated vapor enthalpy
hL_p	Saturated liquid enthalpy
hV_t	Saturated vapor enthalpy
hL_t	Saturated liquid enthalpy
h_pt	Enthalpy as a function of pressure and temperature
h_ps	Enthalpy as a function of pressure and entropy
h_px	Enthalpy as a function of pressure and vapor fraction
h_prho	Enthalpy as a function of pressure and density. Observe for low temperatures (liquid) this equation has 2 solutions
h_tx	Enthalpy as a function of temperature and vapor fraction

Specific volume

Function	Description
vV_p	Saturated vapor volume
vL_p	Saturated liquid volume
vV_t	Saturated vapor volume
vL_t	Saturated liquid volume
v_pt	Specific volume as a function of pressure and temperature
v_ph	Specific volume as a function of pressure and enthalpy
v_ps	Specific volume as a function of pressure and entropy

Density

Function	Description
rhoV_p	Saturated vapor density
rhoL_p	Saturated liquid density
rhoV_t	Saturated vapor density
rhoL_t	Saturated liquid density
rho_pt	Density as a function of pressure and temperature
rho_ph	Density as a function of pressure and enthalpy
rho_ps	Density as a function of pressure and entropy

Specific entropy

Function	Description
sV_p	Saturated vapor entropy
sL_p	Saturated liquid entropy
sV_t	Saturated vapor entropy
sL_t	Saturated liquid entropy
s_pt	Specific entropy as a function of pressure and temperature (Returns saturated vapor enthalpy if mixture)
s_ph	Specific entropy as a function of pressure and enthalpy

Specific internal energy

Function	Description
uV_p	Saturated vapor internal energy
uL_p	Saturated liquid internal energy
uV_t	Saturated vapor internal energy
uL_t	Saturated liquid internal energy
u_pt	Specific internal energy as a function of pressure and temperature
u_ph	Specific internal energy as a function of pressure and enthalpy
u_ps	Specific internal energy as a function of pressure and entropy

Specific isobaric heat capacity

Function	Description
CpV_p	Saturated vapor heat capacity
CpL_p	Saturated liquid heat capacity
CpV_t	Saturated vapor heat capacity
CpL_t	Saturated liquid heat capacity
Cp_pt	Specific isobaric heat capacity as a function of pressure and temperature
Cp_ph	Specific isobaric heat capacity as a function of pressure and enthalpy
Cp_ps	Specific isobaric heat capacity as a function of pressure and entropy

Specific isochoric heat capacity

Function	Description
CvV_p	Saturated vapor isochoric heat capacity
CvL_p	Saturated liquid isochoric heat capacity
CvV_t	Saturated vapor isochoric heat capacity
CvL_t	Saturated liquid isochoric heat capacity
Cv_pt	Specific isochoric heat capacity as a function of pressure and temperature
Cv_ph	Specific isochoric heat capacity as a function of pressure and enthalpy
Cv_ps	Specific isochoric heat capacity as a function of pressure and entropy

Speed of sound

Function	Description
wV_p	Saturated vapor speed of sound
wL_p	Saturated liquid speed of sound
wV_t	Saturated vapor speed of sound
wL_t	Saturated liquid speed of sound
w_pt	Speed of sound as a function of pressure and temperature
w_ph	Speed of sound as a function of pressure and enthalpy
w_ps	Speed of sound as a function of pressure and entropy

Viscosity

Function	Description
my_pt	Viscosity as a function of pressure and temperature
my_ph	Viscosity as a function of pressure and enthalpy
my_ps	Viscosity as a function of pressure and entropy

Thermal Conductivity

Function	Description
tcL_p	Saturated vapor thermal conductivity
tcV_p	Saturated liquid thermal conductivity
tcL_t	Saturated vapor thermal conductivity
tcV_t	Saturated liquid thermal conductivity
tc_pt	Thermal conductivity as a function of pressure and temperature
tc_ph	Thermal conductivity as a function of pressure and enthalpy
tc_hs	Thermal conductivity as a function of enthalpy and entropy

Surface tension

Function	Description
st_t	Surface tension for two phase water/steam as a function of T
st_p	Surface tension for two phase water/steam as a function of p

vapor fraction

Function	Description
x_ph	vapor fraction as a function of pressure and enthalpy
x_ps	vapor fraction as a function of pressure and entropy

vapor volume fraction

Function	Description
vx_ph	vapor volume fraction as a function of pressure and enthalpy
vx_ps	vapor volume fraction as a function of pressure and entropy

Pressure along the Melting and Sublimation Curves

Function	Description
pmelt_t	Pressure along the melting curve as a function of temperature
psubl_t	Pressure along the sublimation curve as a function of temperature

1.4.2 Available Functions for Heavy Water

Function	Description
my_rhoT	Viscosity as a function of density and temperature
tc_rhoT	Thermal conductivity as a function of density and temperature

1.4.3 pyXSteam package

Submodules

pyXSteam.XSteam module

pyXSteam.UnitConverter module

pyXSteam.Constants module

```
1
2  __SPECIFIC_GAS_CONSTANT__ = 0.461526  # kJ kg^-1 K^-1
3  __CRITICAL_TEMPERATURE__ = 647.096  # K
4  __CRITICAL_PRESSURE__ = 22.06395  # MPa
5  __CRITICAL_DENSITY__ = 322  # kg m^-1
6  __TRIPLE_POINT_TEMPERATURE__ = 273.16  # K (Eq9 Page 7)
7  __TRIPLE_POINT_PRESSURE__ = 0.000611657  # MPa (Eq9 Page 7)
8  __TRIPLE_POINT_SPECIFIC_ENTHALPY__ = 0.611783e-3  # kJ kg^-1 (Eq10␣
   ↪Page 7)
9  # Released September 1992, Revision of the Release of 1992
10 __CRITICAL_TEMPERATURE_H20_1992__ = 647.096  # +-0.1 K
11 __CRITICAL_PRESSURE_H20_1992__ = 22.067  # + 0.27*(+-0.1)+-0.005 MPa
12 __CRITICAL_DENSITY_H20_1992__ = 322  # +-3 kg m^-1
13
14 __CRITICAL_TEMPERATURE_D20_1992__ = 643.847  # +-0.2 K
15 __CRITICAL_PRESSURE_D20_1992__ = 21.671  # + 0.27*(+-0.2)+-0.01 MPa
16 __CRITICAL_DENSITY_D20_1992__ = 356  # +-5 kg m^-1
17 # Other common constants used in calculations
18 __ABSOLUTE_ZERO_CELSIUS__ = -273.15  # °C
```

pyXSteam.Regions module

pyXSteam.RegionBorders module

pyXSteam.RegionSelection module

pyXSteam.TransportProperties module

pyXSteam.IAPWS_R4 module

pyXSteam.IAPWS_R14 module

Module contents

1.4.4 Tutorial and Demos

Usage

Simple Example:

```
from pyXSteam.XSteam import XSteam
steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
print steam_table.hL_p(220.0)
```

By using the unitSystem Parameter, you can tell XSteam witch Unit System you are using.:

```
steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS) # m/kg/sec/°C/bar/W
steam_table = XSteam(XSteam.UNIT_SYSTEM_FLS) # ft/lb/sec/°F/psi/btu
steam_table = XSteam(XSteam.UNIT_SYSTEM_BARE) # m/kg/sec/K/MPa/W
```

To enable logging, add the following lines to your code:

```
import logging
logger = logging.getLogger('pyXSteam')
logger.setLevel(logging.DEBUG) sh = logging.StreamHandler()
sh.setFormatter(logging.Formatter('%(name)s - %(levelname)s -
↪%(message)s'))
logger.addHandler(sh)
```

Calculate single values

This is a simple example:

```
>>> from pyXSteam.XSteam import XSteam
>>> steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
>>> steam_table.hL_p(220.0)
2021.909286172027
>>> steam_table.tcV_p(1)
0.02475366759235046
```

Generate Diagrams

Diagrams based on the calculated values can easily be created using numpy and matplotlib.

Example: To draw a T(p) diagram showing the saturation curve:

```
from pyXSteam.XSteam import XSteam
import matplotlib.pyplot as pyplot
import numpy as np
steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
p = np.arange(-100.0, 250.0, 1.0)
ntsat_p = np.frompyfunc(steam_table.tsat_p, 1, 1)
tsat = ntsat_p(p)
line1, = pyplot.plot(tsat, p)
pyplot.xlabel("t")
pyplot.ylabel("p")
pyplot.setp(line1, linewidth=1, color='b')
pyplot.show()
```

For more demos, see pyXSteamDemo.py

Heavy Water functions

The functions to calculate values for heavy water are available through the class XSteamHW

```
>>> from pyXSteam.XSteamHW import XSteamHW
>>> steam_table = XSteamHW(XSteam.UNIT_SYSTEM_MKS)
>>> steam_table.my_rhoT(1.2, 300.0)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""collection of demos presenting the functionality of pyXSteam"""
import time
import logging
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as pyplot
import numpy as np
from pyXSteam.XSteam import XSteam
from pyXSteam.XSteam_HW import XSteam_HW


def demo_simpel_values():
    """calculate values and print the results"""
    steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
    # get saturated liquid enthalpy for a preasure of 220 bar
    print("hV_p(220.0) =", steam_table.hL_p(220.0))
    # get saturated vapour enthalpy for a preasure of 220 bar
    print("hV_p(220.0) =", steam_table.hV_p(220.0))
    print("tcL_p(1.0) =", steam_table.tcL_p(1.0))
    print("tcL_t(25.0) =", steam_table.tcL_t(25.0))
    print("tcV_p(1.0) =", steam_table.tcV_p(1.0))
    print("tcL_t(25.0) =", steam_table.tcV_t(25.0))
    print("tc_hs(100.0, 0.34) =", steam_table.tc_hs(100.0, 0.34))
    print("tc_ph(1.0, 100.0) =", steam_table.tc_ph(1.0, 100.0))
    print("tc_pt(1.0, 25.0) =", steam_table.tc_pt(1.0, 25.0))
    print("w_ps(1.0, 1.0) =", steam_table.w_ps(1.0, 1.0))


def demo_generate_ph_diagramm(path=None, precision=1.0):
    """Generate a p(h) Diagram showing the Saturation Line"""
    steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
    p_krit = (
        steam_table.criticalPressure() - 0.0001
    )  # minus 0.0001 or else hL_V returns NaN
    h_krit = steam_table.hL_p(p_krit)
    p = np.arange(0.0, 1000, precision)
    p2 = np.arange(0.5, p_krit, precision)
    vapor_fraction = np.arange(0.1, 1.0, 0.1)
    h = np.arange(200.0, 4500.0, 100.0)
    rho = np.arange(1, 15.0, precision * 2)
    nph_px = np.frompyfunc(steam_table.h_px, 2, 1)
    nph_pt = np.frompyfunc(steam_table.h_pt, 2, 1)
    nphL_p = np.frompyfunc(steam_table.hL_p, 1, 1)
    nphV_p = np.frompyfunc(steam_table.hV_p, 1, 1)
```

(continues on next page)

(continued from previous page)

```
    npp_hrho = np.frompyfunc(steam_table.p_hrho, 2, 1)
    # Siede und Taulinie
    hL = nphL_p(p)
    hV = nphV_p(p)
    # Dampfgehalt
    for vf in vapor_fraction:
        h_px = nph_px(p2, vf)
        (line,) = pyplot.plot(h_px, p2)
        pyplot.setp(line, linewidth=1, color="g")
    # Temperature
    for temp in range(0, 900, 30):
        h_pt = nph_pt(p, temp)
        (line,) = pyplot.plot(h_pt, p)
        pyplot.setp(line, linewidth=1, color="r")
    # Dichte
    for r in rho:
        p_hrho = npp_hrho(h, r)
        (line,) = pyplot.plot(h, p_hrho)
        pyplot.setp(line, linewidth=1, color="y")
    # Kritischer Punkt
    pyplot.plot([h_krit], [p_krit], marker="s", mfc="k", ms=8)
    (line1,) = pyplot.plot(hL, p)
    (line2,) = pyplot.plot(hV, p)
    pyplot.xlabel("h in [kJ/kg]")
    pyplot.ylabel("p in [bar]")
    pyplot.setp(line1, linewidth=2, color="b")
    pyplot.setp(line2, linewidth=2, color="r")
    pyplot.yscale("log")
    pyplot.grid()
    if path is None:
        pyplot.show()
    else:
        pyplot.savefig(path, bbox_inches="tight")


def demo_generate_Tp_diagramm():
    """Generate a T(p) Diagram showing the Saturation Curve"""
    steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
    p = np.arange(-100.0, 250.0, 1.0)
    ntsat_p = np.frompyfunc(steam_table.tsat_p, 1, 1)
    tsat = ntsat_p(p)
    (line1,) = pyplot.plot(tsat, p)
    pyplot.xlabel("t")
    pyplot.ylabel("p")
    pyplot.setp(line1, linewidth=1, color="b")
    pyplot.show()


def demo_generate_pvT_diagramm():
```

(continues on next page)

(continued from previous page)

```
    """Generate a Diagram showing the v(p,T) as a 3D survace"""
    steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
    fig = pyplot.figure()
    ax = Axes3D(fig)
    p = np.arange(-10.0, 300.0, 5.0)
    t = np.arange(-50.0, 400.0, 5.0)
    p, t = np.meshgrid(p, t)
    npv_pt = np.frompyfunc(steam_table.v_pt, 2, 1)
    v = npv_pt(p, t)
    ax.plot_surface(v, p, t, rstride=1, cstride=1, linewidth=0,␣
↪shade=True)
    ax.set_xlabel("v")
    ax.set_ylabel("p")
    ax.set_zlabel("t")
    pyplot.show()


def demo_moillier_diagramm():
    """Generate a moillier diagram"""
    steam_table = XSteam(XSteam.UNIT_SYSTEM_MKS)
    s = np.arange(2.0, 10.0, 0.01)
    pSteps = [0.006117, 0.01, 0.02, 1.0, 2.0, 3.0, 10, 100, 1000]
    nph_ps = np.frompyfunc(steam_table.h_ps, 2, 1)
    for pstep in pSteps:
        h = nph_ps(pstep, s)
        (hline,) = pyplot.plot(s, h)
        pyplot.setp(hline, linewidth=1, color="b")
    pyplot.xlabel("s in [kJ/(kg K)]")
    pyplot.ylabel("h in [kJ/kg]")
    pyplot.show()


def demo_ice_diagramm():
    """Generate a diagram showing the sublimation and melting␣
↪preasure"""
    steam_table = XSteam(XSteam.UNIT_SYSTEM_BARE)
    t_subl = np.arange(50.0, 273.16, 2.0)
    t_melt_Ih = np.arange(251.165, 273.16, 2.0)
    t_melt_III = np.arange(251.165, 256.164, 2.0)
    t_melt_V = np.arange(256.164, 273.31, 2.0)
    t_melt_VI = np.arange(273.31, 355.0, 2.0)
    t_melt_VII = np.arange(355.0, 751.0, 2.0)

    psubl_func = np.frompyfunc(steam_table.psubl_t, 1, 1)
    pmelt_func = np.frompyfunc(steam_table.pmelt_t, 2, 1)

    (line1,) = pyplot.plot(t_subl, psubl_func(t_subl))
    (line2,) = pyplot.plot(t_melt_Ih, pmelt_func(t_melt_Ih, steam_
↪table.TYPE_ICE_Ih))
```

(continues on next page)

(continued from previous page)

```
    (line3,) = pyplot.plot(t_melt_III, pmelt_func(t_melt_III, steam_
↪table.TYPE_ICE_III))
    (line4,) = pyplot.plot(t_melt_V, pmelt_func(t_melt_V, steam_
↪table.TYPE_ICE_V))
    (line5,) = pyplot.plot(t_melt_VI, pmelt_func(t_melt_VI, steam_
↪table.TYPE_ICE_VI))
    (line6,) = pyplot.plot(t_melt_VII, pmelt_func(t_melt_VII, steam_
↪table.TYPE_ICE_VII))

    pyplot.xlabel("T in [K]")
    pyplot.ylabel("p in [MPa]")
    pyplot.setp(line1, linewidth=1, color="b")
    pyplot.setp(line2, linewidth=1, color="g")
    pyplot.setp(line3, linewidth=1, color="r")
    pyplot.setp(line4, linewidth=1, color="y")
    pyplot.setp(line5, linewidth=1, color="g")
    pyplot.setp(line6, linewidth=1, color="r")
    pyplot.show()


def demo_simpel_values_heavy_water():
    """calculate values for heavy water and print the results"""
    steam_table_hw = XSteam_HW(XSteam_HW.UNIT_SYSTEM_MKS)
    print("my_rhoT(1.0, 320.0) =", steam_table_hw.my_rhoT(1.0, 320.
↪0))
    print("my_rhoT(1.0, 320.0) =", steam_table_hw.tc_rhoT(1.0, 320.
↪0))


if __name__ == "__main__":
    logger = logging.getLogger("pyXSteam")
    logger.setLevel(logging.DEBUG)
    sh = logging.StreamHandler()
    sh.setFormatter(logging.Formatter("%(name)s - %(levelname)s -
↪%(message)s"))
    logger.addHandler(sh)

    print("Select which demo to run:")
    print("1. Run simple calculations")
    print("2. generate ph diagram")
    print("3. generate Tp diagram")
    print("4. generate pvT diagram")
    print("5. generate moillier diagram")
    print("6. generate ice metling and sublimation diagram")
    print("7. Run sinple calculations for heavy water")

    selection = str(input("Please enter selection [1-7]: "))
    print("You selected " + selection)
```

(continues on next page)

(continued from previous page)

```
    if selection == "1":
        start = time.process_time()
        demo_simpel_values()
        print("Demo took", time.process_time() - start, "seconds to␣
↪complete")
    elif selection == "2":
        start = time.process_time()
        demo_generate_ph_diagramm()
        print("Demo took", time.process_time() - start, "seconds to␣
↪complete")
    elif selection == "3":
        start = time.process_time()
        demo_generate_Tp_diagramm()
        print("Demo took", time.process_time() - start, "seconds to␣
↪complete")
    elif selection == "4":
        start = time.process_time()
        demo_generate_pvT_diagramm()
        print("Demo took", time.process_time() - start, "seconds to␣
↪complete")
    elif selection == "5":
        start = time.process_time()
        demo_moillier_diagramm()
        print("Demo took", time.process_time() - start, "seconds to␣
↪complete")
    elif selection == "6":
        start = time.process_time()
        demo_ice_diagramm()
        print("Demo took", time.process_time() - start, "seconds to␣
↪complete")
    elif selection == "7":
        start = time.process_time()
        demo_simpel_values_heavy_water()
        print("Demo took", time.process_time() - start, "seconds to␣
↪complete")
    else:
        print("Unknown selection")
```

1.5 Indices and tables

- genindex
- modindex
- search