PyForFluids

Release 0.0.1-a1

Benelli, Federico E.; Arpajou, M. Candelaria

Feb 09, 2022

CONTENTS:

CHAPTER

ONE

PYFORFLUIDS

1.1 pyforfluids package

1.1.1 Subpackages

pyforfluids.models package

Module contents

Models.

Modules that contains the multiple models to estimate fluid's properties.

Submodules

pyforfluids.models.gerg2008 module

GERG2008 EoS.

class pyforfluids.models.gerg2008.**GERG2008**

Bases: object

GERG2008 equation of state.

GERG2008 Equation of state described by O. Kunz and W. Wagner [1]

The components must be those of the GERG2008 model. This class use imported methods from Fortran subroutines for high speed calculation of properties.

References

[1]

Methods

validate_components:	Check if the components belong to the EOS.
validate_ranges:	Check in which range of validity are the temperature and pressure.
set_concentration:	Normalize the composition as molar fractions.
calculate_properties:	Calculate the properties.

`calculate_properties`(*temperature*, *pressure*, *density*, *composition*)

Calculate the thermodynamic properties of the given fluid.

Calculation of the thermodynamic properties of the fluid at it's given temperature and density.

Parameters

temperature: float Fluid temperature in Kelvin degrees [K]

pressure: float Fluid pressure in Pascal [Pa]

density: float Fluid density in mol per liter [mol/L]

composition: dict Dictionary of the compounds concentrations as: `{"methane": 0.8, "ethane":0.2}` When necessary, the concentration values are normalized.

Returns

dict Dictionary of the thermodynamic properties of the given fluid calculated with GERG2008 equation of state.

`name = 'GERG2008'`

`set_concentration`(*composition*)

Verify if the sum of the molar fractions of the fluid components is 1.

If not, a warninig message is sent and the composition is normalized.

Parameters

components: dict Dictionary of the fluid compounds as keys and their molar fraction as values.

Returns

array Array of the normalized fluid composition vector.

`valid_components = {'argon', 'butane', 'carbon_dioxide', 'carbon_monoxide', 'decane', 'ethane', 'helium', 'heptane', 'hexane', 'hydrogen', 'hydrogen_sulfide', 'isobutane', 'isopentane', 'methane', 'nitrogen', 'nonane', 'octane', 'oxygen', 'pentane', 'propane', 'water'}`

`validate_components`(*components*)

Validate fluid components.

Verify if the given fluid components are valid for the GERG2008 equation of state.If not, a ValueError Exception is raised.

Parameters

components: set Set of the fluid components to be verified.

`validate_ranges`(*temperature*, *pressure*)

Validate fluid temperature and pressure.

Verify whether the fluid temperature and pressure values belong to the normal, extended or invalid use range of the GERG2008 equation of state. A warning menssage is sent if the temperature and pressure conditions are those of the extended or invalid range, and also if they take negative values.

Parameters

temperature: float Fluid temperature in Kelvin degrees [K]

pressure: float Fluid pressure in Pascal [Pa]

1.1.2 Submodules

1.1.3 pyforfluids.core module

Core module.

`class pyforfluids.core.Fluid(`*model*, *composition*, *temperature*, *pressure=None*, *density=None*`)`

Bases: `object`

Describes a fluid based on a given model and it's thermo variables.

Density and pressure can't be defined at the same time. If pressure is given, the density will be calculated with an iterative algorithm using the derivatives given by the model.

Parameters

model: pyforfluids model_like Model to use in the properties calculation.

composition [dict] Dictionary with the compounds concentrations as: `{'methane': 0.8, 'ethane': 0.1}` In some cases, as in GERG2008, the values will be normalized for the calculations but won't be modified in the Fluid attribute

temperature: float Fluid temperature in degrees Kelvin [K]

pressure: float Fluid pressure in Pascals [Pa]

density: float Fluid density in mol per liter [mol/L]

Attributes

properties [dict] Fluid properties calculated by it's model.

Methods

copy:	Returns a copy of the Fluid.
set_composition:	Change the Fluid's composition.
set_temperature:	Change the Fluid's temperature.
set_density:	Change the Fluid's density.
set_pressure:	Change the FLuid's pressure.
calculate_properties:	Calculate the Fluids properties, returns as a dictionary.
isotherm:	Calculate the Fluid properties along a density range.
density_iterator:	Calculate the Fluid's density based on a specified pressure.

`calculate_properties()`

Calculate the fluid's properties.

`copy()`

Return a copy of the fluid, taking density as independant variable.

Returns

Fluid

`density_iterator`(*objective_pressure*, *vapor_phase=True*, *liquid_phase=True*)
With a given pressure and temperature value, get the fluid density.

Parameters

objective_pressure: float Fluid pressure where to calculate density.

vapor_phase: bool Find vapor phase root.

liquid_phase: bool Find liquid phase root.

Returns

liquid_density: float Calculated density in liquid phase.

vapor_density: float Calculated density in vapor phase.

single_phase: bool True if only one root was found.

`isotherm`(*density_range*)
Calculate isotherm along a density range.

Calculate the fluid's properties that it's model can give at constant temperature along a density range.

Parameters

density_range: array-like Range of values of density where to calculate the properties.

Returns

dict Dictionary with all the properties that the model can calculate along the density_range.

`set_composition`(*composition*)
Change the fluid's composition.

Parameters

composition: dict Dictionary with the fluid compounds as keys and their molar concentration as values

example: composition = {'methane': 0.1, 'ethane': 0.9}

`set_density`(*density*)
Change the fluid's density.

Parameters

density: float New density

`set_pressure`(*pressure*)
Change the fluid's pressure.

Parameters

pressure: float New pressure

`set_temperature`(*temperature*)
Change the fluid's temperature.

Parameters

temperature: float New temperature

1.1.4 Module contents

PyForFluids.

Fluid properties simulation based on Ecuations of State.

CHAPTER

TWO

INSTALLATION

For installing _PyForFluids_ you just need to:

```
pip install pyforfluids
```

Make sure to check the requirements first!

2.1 Requirements

Be sure to install *numpy* and a fortran compiler previously, since both are needed for the compilation of *Fortran* code.

2.1.1 NumPy

```
pip install numpy
```

2.1.2 Fortran Compiler

Linux

- **Debian-based** (Debian, Ubuntu, Mint,...)

```
sudo apt install gfortran
```

- **Arch-based** (Arch, Manjaro, Garuda, ...)

```
sudo pacman -S gfortran
```

Windows

We recommended using the Windows Subsystem for Linux and following the Linux instructions.

WSL

If WSL ain't being used, the native Windows wheels will be download instead, so no need to worry!

MacOS

```
brew install gfortran
```

CHAPTER

THREE

TUTORIAL

3.1 Relevant imports

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pyforfluids as pff
```

3.2 Definition of the model to be used

[2]:
```
model = pff.models.GERG2008()
```

3.3 Fluid's initial state

[3]:
```
temperature = 250  # Degrees Kelvin
density = 1  # mol/L
pressure = 101325  # Pa
composition = {'methane': 0.9, 'ethane': 0.05, 'propane': 0.05}  # Molar fractions
```

3.3.1 Definition of the fluid

The properties will be calculated at the moment of the object definition.

[4]:
```
fluid = pff.Fluid(
    model=model,
    composition=composition,
    temperature=temperature,
    density=density
)
```

Pressure as an init variable

Pressure can be used as an initial variable, in this case the method `fluid.density_iterator` will be called internally to find the root of density at the given pressure, since all models use density as an independent variable instead of temperature.

This can lead to trouble since multiple roots can be obtained in the equilibrium region!

[5]:
```
fluid = pff.Fluid(
    model=model,
    composition=composition,
    temperature=temperature,
    pressure=pressure
)
```

3.3.2 Accessing properties

Properties are stored in the Fluid attribute `Fluid.properties` as a dictionary, they can either be accessed that way or by calling them directly from the fluid.

All the properties are expressed in **International System** units, except for density that's expressed in **[mol/L]**

[6]:
```
fluid.properties
```

[6]:
```
{'density_r': 9.442772800175154,
 'temperature_r': 207.1068112975803,
 'delta': 0.005162295708090389,
 'tau': 0.8284272451903212,
 'ao': array([[-3.47583079e+01,  0.00000000e+00,  0.00000000e+00],
        [ 1.93712266e+02, -4.99964400e+01,  0.00000000e+00],
        [-3.75244420e+04, -5.41251465e+00,  0.00000000e+00]]),
 'ar': array([[-0.0041676 ,  0.        ,  0.        ],
        [-0.80651578, -0.0108937 ,  0.        ],
        [ 0.31041621, -0.00769256, -2.10957546]]),
 'z': 0.9958365270355344,
 'cv': 30.928532328981273,
 'cp': 39.39441402789975,
 'w': 380.3778978355614,
 'isothermal_thermal_coefficent': -0.2725846588975132,
 'dp_dt': 0.4072695321069243,
 'dp_drho': 2061.3266554120955,
 'dp_dv': -4.898145216338861,
 'p': 100903.24995863381,
 's': -55.41564180556873,
 'u': -86111.8177689482,
 'h': -84041.85403879466,
 'g': -70187.94358740246,
 'jt': 0.00035980611360190225,
 'k': 1.2633809737968698,
 'b': -0.08557993798318263,
 'c': 0.003479888600305259}
```

[7]:
```
fluid['cv']
```

[7]:
```
30.928532328981273
```

3.3.3 State changes

A fluid thermodynamic variable can be changed by using the methods:

- `Fluid.set_temperature`
- `Fluid.set_composition`
- `Fluid.set_density`
- `Fluid.set_pressure`

When a property is changed, the properties are not re-calculated, so it's a **must** to call the method `Fluid.calculate_properties`. This is intended to avoid useless calculations if two or more variables are to be changed. In the case of a pressure change `Fluid.density_iterator` will be called!

[8]:
```
fluid.set_temperature(280)
fluid.set_density(2)

fluid.calculate_properties()
```

[9]:
```
fluid['cv']
```

[9]:
```
34.71399971808808
```

3.4 Calculating isotherms

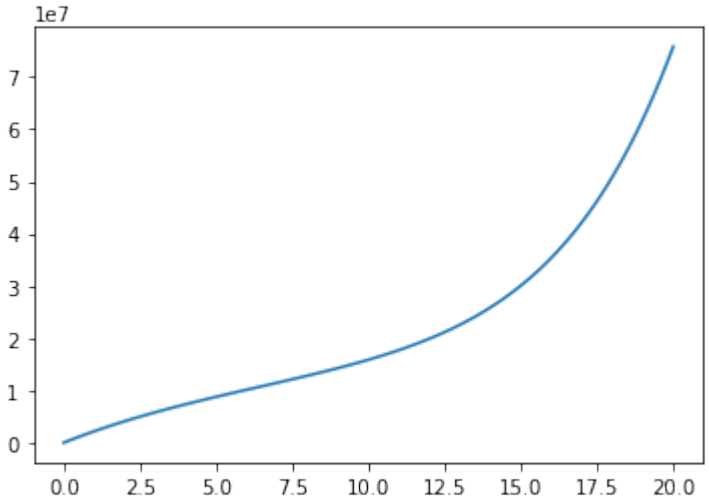
Isotherms at the fluid temperature can be calculated along a density range with the method `Fluid.isotherm`. This will return a dictionary equivalent to the `Fluid.properties` one, but each value will be a list instead of a single value.

[10]:
```
density_range = np.linspace(0.001, 20, 100)
isotherm = fluid.isotherm(density_range)
```

[11]:
```
isotherm['cv'][:5]
```

[11]:
```
[33.34361845195672,
 33.48240604268746,
 33.622421620895594,
 33.76309070522664,
 33.903893177656855]
```

[12]:
```
plt.plot(density_range, isotherm['p'])
plt.show()
```



[ ]:

PyForFluids (Python-Fortran-Fluids) is a Python package focused in the calculation of Fluid properties based on Ecuations of State (EoS). It provides a simple interface to work from Python but also exploits the high performance Fortran code for the more heavy calculations.

It's designed with modularity in mind, in a way that new thermodyinamic models are easy to add, they even can be written either in Python or Fortran.

CHAPTER

FOUR

AVAILABLE PROPERTIES

- Reduced Temperature and Density
- Ideal Helmholtz Energy (Ao)
- Residual Helmholtz Energy (Ar)
- Compresibility Factor (Z)
- Isochoric Heat (Cv)
- Isobaric Heat (Cp)
- Speed of sound (w)
- Isothermal throttling coefficent ()
- **Pressure derivatives:**
  - Temperature
  - Density
  - Volume
- Pressure (P)
- Entropy (S)
- Gibbs Free Energy (G)
- Enthalpy (H)
- Joule-Thompson coefficent
- Isoentropic exponent
- **Virial Terms:**
  - B
  - C

CHAPTER

FIVE

MOTIVATION

While nowadays there are a lot of tools for calculation of thermodynamic properties of fluids, most of them either are hard to mantain and don't have an integrated testing system or are embeded to other softwares (as spredsheat software) limiting the things that can be done to that enviroment.

PyForFluids aims to be a tool:

- With high performance, since most of it's calculations are done in Fortran
- Easy to scale due to it's modular design using the power of Python objects.
- Continuosly tested (at every *push*)to spot any problems as soon as possible.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] O. Kunz and W. Wagner, "The GERG-2008 Wide-Range Equation of State for Natural Gases and Other Mixtures: An Expansion of GERG-2004", J. Chem. Eng. Data 2012, 57, 11, 3032–3091. doi:10.1021/je300655b. https://pubs.acs.org/doi/10.1021/je300655b

PYTHON MODULE INDEX

p

INDEX