PyForFluids: A Python package for multi-component fluid thermodynamic properties and phase equilibrium calculations.

Benelli, Federico E.[1]; **Arpajou, M. Candelaria** [2]; **Cabral, Juan B.** [3]; **Cismondi, Martín** [1]

[1] Instituto de Investigación y Desarrollo en Ingeniería de Procesos y Química Aplicada (IPQA-UNC-CONICET)

[2] INGAR Instituto de Desarrollo y Diseño (CONICET-UTN)

[3] Instituto de Astronomía Teórica y Experimental (IATE-OAC-CONICET)

federico.benelli@mi.unc.edu.ar

SCAN ME

Abstract

PyForFluids (Python-Fortran-Fluids) is a Python package focused on the calculation of multicomponent fluids properties and phase equilibrium based on Equations of State (EoS). It provides a simple interface to work from a high level object oriented abstraction but also exploits the high performance Fortran code for the heavier calculations. Right now it includes the multifluid GERG-2008 EoS [1] and three cubic EoS (Peng-Robinson, Soave-Redlich-Kwong and RKPR) [2, 3, 4] are being implemented. All four equations are explicit in the Helmholtz Free Energy. PyForFluids calculates multiple thermodynamic properties like speed of sound, isobaric heat, compressibility factor, entropy, enthalpy, etc. Besides that, biphasic equilibrium calculations like flash, bubble and dew points are included, with phase envelopes tracing being in current development. To realize complex calculations, PyForFluids takes advantage of the high performance and speed of Fortran code. At the same time, it offers a user-friendly Python interface. The integration between these two programming languages is achieved thanks to numpy[5] module f2py[6]. Fortran was the chosen language due to both being faster for numerical routines and an availability of legacy projects. This package is designed with a collaborative and modular approach in mind, taking advantage of an object oriented programming approach. To both see the inner workings of it and make changes/additions proposals all the code is available on an public repository at GitHub https://github.com/fedebenelli/pyforfluids PyForFluids is made following programming good practices standards for continuous integration. At each code addition or modification, the package is tested with specific unit tests to assure the reliance of the computations. Also the documentation where each class and function is described is automatically generated and hosted at pyforfluids.readthedocs.io with each update, where also a simple tutorial with the basic usage of the package can be found.

Introduction

Nowadays there are multiple comercial simulators capable of solving phase equilibria and thermodynamic properties calculation but, due to their expensive licenses, they're unreachable for a lot of research groups as well to small enterprises. Besides that unreachability, most of those softwares are closed source so it's imposible to both inspect their inner workings as well as making desirable modifications.

`pyforfluids` aim's to be a tool that provides robust fluid thermodynamic properties and phase equilibria calculations, with focus in both simplicity in use and extensibility, with the help of abstractions obtained with an object oriented approach.

Implemented models

All the thermodynamic models included in `pyforfluids` are programmed in Fortran to assure fast performance while making calculations but, since the objects that coordinate the desired calculations are in the Python layer, nothing stops a user in implementing their own models on Python (or using models implemented in other libraries and embedded into `pyforfluids` !).

`pyforfluids` right now implements the following models, with the goal of of expanding it's capabilities to more.

GERG-2008

The GERG-2008 EoS is a multifluid equation of state, developed to be implemented in natural gas usage, but it keeps a high accuracy for the whole range from liquid to super critical conditions. It's explicit on the Helmholtz energy, with it's own ideal and residual parameters.

Ideal term

$$\alpha^o(\overline{x}, \rho, T) = \sum_{i=1}^{N} x_i \left[\alpha_i^o(\rho, T) + \ln x_i\right] \tag{1}$$

Residual term

$$\alpha^r(\overline{x}, \delta, \tau) = \sum_{i=1}^{N} x_i \alpha_i^r(\delta, \tau) + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} x_i x_j F_{ij} \alpha_{ij}^r(\delta, \tau) \tag{2}$$

$$\alpha_i^r(\delta, \tau) = \sum_{k=1}^{K_{Pol,i}} n_{i,k} \delta_{i,k}^d \tau^{t_{i,k}} + \sum_{k=K_{Pol,i}+1}^{K_{Pol,i}+K_{Exp,i}} n_{i,k} \delta_{i,k}^d \tau^{t_{i,k}} e^{-\delta^{c_{i,k}}} \tag{3}$$

Mixing rules

$$\frac{1}{\rho_r(\overline{x})} = \sum_{i=1}^{N} \sum_{j=1}^{N} x_i x_j \beta_{\nu,ij} \gamma_{\nu,ij} \cdot \frac{x_i + x_j}{\beta_{\nu,ij}^2 x_i + x_j} \cdot \frac{1}{8} \left(\frac{1}{\rho_{c,i}^{1/3}} + \frac{1}{\rho_{c,j}^{1/3}} \right)^3 \tag{4}$$

$$T_r(\overline{x}) = \sum_{i=1}^{N} \sum_{j=1}^{N} x_i x_j \beta_{T,ij} \gamma_{T,ij} \cdot \frac{x_i + x_j}{\beta_{T,ij}^2 x_i + x_j} \cdot \sqrt{(T_{c,i} \cdot T_{c,j})} \tag{5}$$

Cubic Equations of State

Cubic equations of state are implemented following the design recommended by Michelsen and Mollerup. Using a δ_1 and δ_2 parameters to use a general equation applicable to multiple EoS.

$$a_{mix} = \sum^{N} \sum^{N} \sqrt{a_i a_j}(1 - k_{ij})$$
$$b_{mix} = \sum^{N} \sum^{N} \frac{b_i + b_j}{2}(1 - l_{ij})$$

Code Implementation

`pyforfluids` implementation is focused on two kind of Python objects:

Fluid A Fluid object that contains all generic procedures and attributes that can describe a Fluid.

Model A Model object that contains all the relevant procedures related to an Equation of State. On `pyforfluids` there are included two model objects `GERG2008` and `CubicEOS`.

These objects give the user the building blocks to implement

Model Object

A `Model` object contains all the necessary procedures to make thermodynamic properties calculations and relevant constants required by other functions (like P_c).

- Model
  - Constants — R, T_c ,P_c, ω
  - Procedures
    - Components validator
    - Concentration normalizer
    - Properties calculator

All these attributes and procedures are a **must have** for every model. Since they are used by other external functions.

The most simple model object is the one related to the GERG-2008 EoS, since all the parameters are self contained

Defining a GERG2008 model

```python
import pyforfluids as pff

gerg_model = pff.models.GERG2008()
```

Defining a Peng Robinson model

```python
pr_model = pff.models.CubicEOS(
        model="PR",
        mix_rule="ClassicVdW",
        names=names,
        critical_temperature=tc,
        critical_pressure=pc,
        acentric_factor=w,
        kij_matrix=kij,
        lij_matrix=lij
)
```

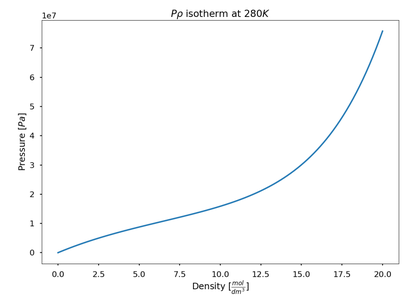
Fluid Object

A `Fluid` object contains the higher level procedures that are generic for any kind of model.

- Fluid
  - Model
  - State — composition, P, T, ρ
  - Density solver
  - Isotherms calculator
  - Properties calculation

```python
import pyforfluids as pff
import numpy as np

fluid = pff.Fluid(
    model=pff.models.GERG2008()
    composition={"methane": 0.2,
                 "ethane":  0.8},
    temperature=270 # K
    density=0.02 # mol/dm3
)
density_range = np.linspace(0.001, 20, 100)
isotherm = fluid.isotherm(density_range)
```



Equilibrium calculation

All the equilibrium related calculations are done by functions that receive a fluid and another relevant variable (like pressure and temperature for a PT Flash) and return a liquid and vapor `Fluid` objects.

```python
flash_pt = pff.equilibrium.flash_pt

composition = {'propane': 0.01, 'butane': 0.5, 'isobutane': 0.15,
               'pentane': 0.2, 'hexane': 0.14}
temperature = 366.48
pressure = 1.039e6

fluid = pff.Fluid(
    model=pff.models.GERG2008(),
    composition=composition,
    temperature=temperature,
    density=1,
)

vapor, liquid, beta, it = flash_pt(fluid, pressure, temperature)

>>> vapor
>>> Fluid(model=GERG2008, temperature=366.48, pressure=1039000.0000,
..........density=0.4236, composition={'propane': 0.02405919, 'butane': 0.59222959,
..........'isobutane': 0.22051554, 'pentane': 0.11992022, 'hexane': 0.04327129})
```

Code Quality

All the produced code is hosted online at https://github.com/fedebenelli/pyforfluids. At each code commit a set of unit tests is run that assure the correct work of the package, as well that codestyle rules are respected. Forcing that at least 90% of the code is tested.

Conclusions

References

[1] A. B. Jones and J. M. Smith. Article Title. *Journal title*, 13(52):123–456, March 2013.

[2] J. M. Smith and A. B. Jones. *Book Title*. Publisher, 7th edition, 2012.