CryoSolver

Package for cryogenic cycle simulation in Python

Documentation Ver. 1.0.0

Sofiya Savelyeva*, Steffen Klöppel*, Christoph Haberstroh*
**Technische Universität Dresden, Germany*

August 25, 2020

Contents

1 Introduction **2**

2 Licensing **2**

3 Installation **2**

4 Simulation principle **3**

5 Package content **3**
5.1 Description of *refpropfunc.py* 4
5.1.1 The list of available functions of the *refpropfunc.py* 4
5.2 Description of *solver.py* 6
5.2.1 The list of available component classes of the *solver.py* 6

6 Using the package **8**
6.1 Main parts of the user program 8
6.2 Important simulation rules 11
6.2.1 Rule to define a closed cycle 11
6.2.2 Stream index definition 11
6.3 Solution failures 12

7 Examples **12**
7.1 Simple reverse Brayton cycle 12
7.2 Claude cycle 13

Acknowledgement **13**

References **14**

1 Introduction

This is the documentation for the *CryoSolver* package for calculation and optimization of cryogenic cycles in Python. The *CryoSolver* offers a collection of classes and functions to automatically determine and solve the system of equations such as mass and energy balances to define the parameters at cycle points. These equations are determined according to the cycle components and flows specified by the user. Its advantage is the possibility to add custom functions and components characteristics due to an open and flexible Python environment. The package does not offer a large variety of tools in contrast to the industrial software, but can be sufficient for research purposes.

Please note that the package is a by-product of the research activity and the authors are not responsible for any issues connected with its utilisation.

2 Licensing

The *CryoSolver* package is available under the 3-Clause BSD License (`https://opensource.org/licenses/BSD-3-Clause`) and can be used, copied, modified for free without permission on condition of an authorship acknowledgement:

Copyright 2020 Sofiya Savelyeva, Steffen Klöppel (KKT, TU Dresden)

To cite the package, please reference to this documentation.

3 Installation

The package has been developed for *Python 3.6.4* (or higher) and the *Refprop 10.0* [1] [1]. The *ctRefprop* wrapper must be installed to allow the communication between Refprop and Python (`https://pypi.org/project/ctREFPROP/`).

Other Python libraries used: *scipy, numpy, math, matplotlib.*

The simulations can be made with and without installation of the package.

Import from a path:
The simulations can be made in an input file saved directly in the parent folder of the package ("...\Cryosolver-1.0.0\").

Alternatively, to import the package in a random directory, the package path needs to be once added into the system path list:

```python
import sys
sys.path.append("Path/to/CryoSolver-1.0.0/")
```

Global installation:
Before the global installation please execute any example file from the "\Cryosolver-1.0.0\examples\" folder and make sure that the program works properly. In case of errors with Refprop please refer to [1] and modify the *refpropfunc.py* [2].

The installation can be made from the terminal using the package installer for Python (`https://pip.pypa.io/en/stable/`):

[1]Note: It is possible to use CoolProp library instead of Refprop. The *refpropfunc.py* needs to be respectively modified by the user.

[2]In case of installation on OSX or Linux some changes in the Refprop setup can be required.

```
>pip install C:/.../CryoSolver-1.0.0
```

After that the package can be imported in any Python file with the command:

```
import cryosolver
```

4 Simulation principle

The calculation of cryogenic cycles is usually based on solving the system of equations for the mass and energy balance. For each component of the cycle it is possible to write the following equations:

$$\sum \dot{m}_{in} \cdot h_{in} - \sum \dot{m}_{out} \cdot h_{out} + \sum \dot{Q}_i + \sum \dot{L}_i = 0 \tag{1}$$

$$\sum \dot{m}_{in} - \sum \dot{m}_{out} = 0, \tag{2}$$

where $\dot{m}$ – mass flow, h – enthalpy, *in* and *out* – index of inlet and outlet streams, $\dot{Q}_i$ – heat input or output, $\dot{L}_i$ – work input or output. The enthalpy can be uniquely defined as a function of temperature and entropy (or alternatively pressure for a non-ideal fluid). Then, the pressure balance of each stream can be defined:

$$p_{in} - p_{out} - \Delta p_{loss} = 0, \tag{3}$$

where p – pressure, Δp_{loss} – pressure losses. Furthermore, equations of composition equilibrium can be written for each cycle component in case if a mixture is used as refrigerant. In addition, specific equations can be added to determine different properties of cycle components or streams. All these equations form a system to uniquely define the state at cycle points. The temperature, entropy (or pressure), total mass flow and, optionally, mole fractions of mixture components for each of N streams form an array of respective variables.
By means of *CryoSolver* it is possible to automatically determine the set of necessary equations depending on the specified components and given parameters by initializing the dedicated class objects. Then they can be solved together in the *CryoSolver* by means of the existing Python algorithms. It is also possible to setup any kind of optimization or case study. The package additionally contains specific functions, such as exergy calculation or plotting the Q-T diagram of a heat exchanger. For details please refer to sections 5 - 6.

5 Package content

CryoSolver consists of two main files:

- *refpropfunc.py*
- *solver.py*

The *refpropfunc.py* contains simplified functions to calculate the fluid properties using the Refprop 10 library. The *solver.py* is the main program containing the library of functions to determine cycle components, to setup parameters of flows and initial values of calculations and to solve the system of energy, mass balance and other specified equations. Units used within the solver are shown in the Table 1. The composition Z should be given on a **mole basis** by default.

Table 1: Units of measure used in the program

Name	Designation in code	units
Temperature	T	K
Pressure	P	$bar(abs)$
Enthalpy	H	kJ/kg
Entropy	S	$kJ/(kg \cdot K)$
Density	D	kg/m^3
Isobaric and isochoric heat capacity	Cp, Cv	$kJ/(kg \cdot K)$
Mass flow	m	kg/s
Power	power	kW
Heat load	capacity	kW
Isentropic efficiency	eta_is	-

5.1 Description of *refpropfunc.py*

The *refpropfunc.py* file contains simplified functions to calculate fluid properties with Refprop 10 [1]. It was added to make the solver independent on the fluid properties database. Thus, to use the CoolProp or any other database, only the functions in *refpropfunc.py* need to be changed to access the dedicated library.

To import the *refpropfunc.py* independently on the solver, the following code can be used:

```
import cryosolver.refpropfunc as RF
```

In case if the *cryosolver.solver* is already imported, the *refpropfunc.py* module is imported automatically as *cryosolver.solver.RF*.

To setup the Refprop library, the following code has been used:

```
import os
from ctREFPROP.ctREFPROP import REFPROPFunctionLibrary
RP = REFPROPFunctionLibrary(os.environ['RPPREFIX'])
RP.SETPATHdll(os.environ['RPPREFIX'])
```

To change the Refprop path, please reference to the Refprop wrappers for Python: `https://github.com/usnistgov/REFPROP-wrappers`.

To calculate the fluid properties, the ABFLSHdll high-level Refprop subroutine has been used. The properties are calculated on mass basis, but the composition should be indicated in mole units.

The fluid setup is performed once at the beginning of the simulation using the SETUPdll routine of the Refprop. All the fluids used in cycle are specified together for faster calculations. The composition array will determine mole fractions of each fluid. In case of a pure fluid needs to be specified, other mole fractions needs to be set as 0.

The reference state is set to 300 K and 1 bar (abs) subtracting the corresponding reference value from the calculated absolute enthalpy and entropy. The internal Refprop subroutine is not used to avoid deviations in case of variable mixture composition.

5.1.1 The list of available functions of the *refpropfunc.py*

- **fluid**(fluids) – the class to setup all the used fluids;
- **MoleComposition**(MassComposition) – Conversion of the mass composition into the mole composition;

- **MassComposition**(MoleComposition) – Conversion of the mole composition into the mass composition;
- **H_TP**(Z, T, P) – Enthalpy defined by composition, temperature and pressure;
- **S_TP**(Z, T, P) – Entropy defined by composition, temperature and pressure;
- **H_PS**(Z, P, S) – Enthalpy defined by composition, pressure and entropy;
- **P_TS**(Z, T, S) – Pressure defined by composition, temperature and entropy;
- **H_TS**(Z, T, S) – Enthalpy defined by composition, temperature and entropy;
- **D_TP**(Z, T, P) – Density defined by composition, temperature and pressure;
- **T_PS**(Z, P, S) – Temperature defined by composition, pressure and entropy;
- **D_PS**(Z, P, S) – Density defined by composition, pressure and entropy;
- **Cv_TP**(Z, T, P) – Isochoric heat capacity defined by composition, temperature and pressure;
- **Cp_TP**(Z, T, P) – Isobaric heat capacity defined by composition, temperature and pressure;
- **T_PH**(Z, P, H) – Temperature defined by composition, pressure and enthalpy;
- **D_PH**(Z, P, H) – Density defined by composition, pressure and enthalpy;
- **P_TH**(Z, T, H) – Pressure defined by composition, temperature and enthalpy;
- **P_HS**(Z, H, S) – Pressure defined by composition, enthalpy and entropy;
- **S_PH**(Z, P, H) – Entropy defined by composition, pressure and enthalpy;

Additional function to calculate fluid properties in the two-phase region:

- **xliq_TP**(Z, T, P) – Liquid phase mole composition defined by temperature and pressure;
- **xvap_TP**(Z, T, P) – Vapor phase mole composition defined by temperature and pressure;
- **xliq_TS**(Z, T, S) – Liquid phase mole composition defined by temperature and entropy ;
- **xvap_TS**(Z, T, S) – Vapor phase mole composition defined by temperature and entropy;
- **vaporstring**(T, S, m, Z) – Mass flow and mole composition of the vapor phase defined by temperature, entropy, total mass flow and composition. Returns the array of values [*mflash, flash*], where *mflash* – total mass flow of the vapor phase, *flash* – composition of the vapor phase on a mole basis;

- **liquidstring**(T, S, m, Z) – Mass flow and mole composition of the liquid phase defined by temperature, entropy, total mass flow and composition. Returns the array of values [*mflash, flashx*], where *mflash* – total mass flow of the liquid phase, *flashx* – composition of the liquid phase on a mole basis.

The detailed description of each function is available directly in the *refpropfunc.py* .

Example: To calculate the enthalpy of a nitrogen-oxygen mixture with a mole composition of [0.78, 0.22] at 250 K and 10 bar, the following code should be used:

```
import cryosolver.refpropfunc as RF

F = RF.fluid("nitrogen.FLD","oxygen.FLD")
Z = [0.78, 0.22]
Enthalpy = RF.H_TP(Z, 250, 10)
```

5.2 Description of *solver.py*

The solver functions can be grouped by their main purpose: setup, component classes, simulation and output.

The main object used for the simulation is a global variable *mainvar*, which contains lists to store system of equations (*mainvar.funclist*), stream parameters (*mainvar.streamlist*) and other necessary values, which can be accessed by the user (see *solver.py* code for more detailes). This variable has type of *mainvarsetup* class and should be created or reset at the beginning of the simulation by means of *start()* function.

The component classes are defined in such a way that depending on values specified by the user a respective number of balance equations is added into the main list of system equations (*mainvar.funclist*). Each of these equations needs to be equal to zero. Moreover, each of them takes no external variables and is only specified for stream properties defined as global variable in *mainvar.streamlist*.

To solve the system of these equations, the function *mainfunction()* is defined. It takes the full array of temperatures, pressures/entropies, mass flows and compositions (optionally) as a variable from the solver, then assigns these values to the dedicated stream properties in the *mainvar.streamlist* and solves each function from the *mainvar.funclist* for new values in *mainvar.streamlist*. The *solution()* function helps to make a proper solver setup and to find the solution using the scipy.optimize.root module.

5.2.1 The list of available component classes of the *solver.py*

Component	Added equations
Compressor	- Energy balance $(h_{out} - h_{in} - (1/\eta_s) \cdot (h_{outs} - h_{in}))$; - mass balance $(\dot{m}_{in} - \dot{m}_{out})$. Optionally: - composition equilibrium for each of mixture components except the last one $(z_{in}[i] - z_{out}[i])$, where i – component of the mixture; - pressure balance if pressure ratio is given $(p_{out}/p_{in} - pressure\_ratio)$; - full energy balance if power is given $(h_{out} \cdot \dot{m}_{out} - h_{in} \cdot \dot{m}_{in} - power)$.

Turbine	- Energy balance $(h_{in} - h_{out} - (1/\eta_s) \cdot (h_{in} - h_outs))$; - mass balance $(\dot{m}_{in} - \dot{m}_{out})$. Optionally: - composition equilibrium for each of mixture components except the last one $(z_{in}[i] - z_{out}[i])$, where i – component of the mixture; - pressure balance if pressure ratio is given $(p_{in}/p_{out} - pressure\_ratio)$; - power balance if power is given $(h_{in} \cdot \dot{m}_{in} - h_{out} \cdot \dot{m}_{out} - power)$.
Heat_exchanger	- Energy balance $\sum(h_{in} \cdot \dot{m}_{in} - h_{out} \cdot \dot{m}_{out})$ for all cells; - mass balance for each cell $(\dot{m}_{in} - \dot{m}_{out})$; - pressure balance for each cell $(p_{in} - p_{out})$. Optionally: - power balance for each stream if capacity is known $(h_{in} \cdot \dot{m}_{in} - h_{out} \cdot \dot{m}_{out} - capacity)$; - composition equilibrium for each of mixture components except the last one $(z_{in}[i] - z_{out}[i])$, where i – component of the mixture; - equality of Number of Transfer Units to the specified value (defined by the outlet temperatures of a cold stream for a counterflow heat exchanger).
Simple_heat_exchanger	- Pressure balance $(p_{in} - p_{out})$; - mass balance $(\dot{m}_{in} - \dot{m}_{out})$. Optionally: - energy balance if heat load (capacity) is given $(capacity - (h_{out} \cdot \dot{m}_{out} - h_{in} \cdot \dot{m}_i n))$; - composition equilibrium for each of mixture components except the last one $(z_{in}[i] - z_{out}[i])$, where i – component of the mixture; - temperature balance if outlet temperature is given $(T_{out} - value)$.
Separator	- Mass balance $(\dot{m}_{in} - \sum(\dot{m}_{out}[i]))$; - temperature equality of all outlets to the inlet $(T_{in} - T_{out}[i])$; - pressure equality of all outlets to the inlet $(p_{in} - p_{out}[i])$. Optionally: - composition equilibrium for each of mixture components except the last one $(z_{in}[i] - z_{out}[i])$, where i – component of the mixture.
Mixer	- Energy balance $(h_{out} \cdot \dot{m}_{out} - \sum(h_{in} \cdot \dot{m}_{in}))$; - mass balance $(\dot{m}_{out} - \sum(\dot{m}_{in}[i]))$; - pressure equality between all the inlets and outlet stream $(p_{out} - p_{in}[i])$. Optionally: - composition equilibrium for each of mixture components except the last one (calculated from compositions and mass flows of inlet streams).

Valve	- Energy balance ($h_{in} \cdot \dot{m}_{in} - h_{out} \cdot \dot{m}_{out}$); - mass balance ($\dot{m}_{in} - \dot{m}_{out}$); - pressure balance depending on specified parameter: for pressure drop ($p_{in} - p_{out} - value$) or for outlet pressure ($p_{out} - value$). Optionally: - composition equilibrium for each of mixture components except the last one (calculated from compositions and mass flows of inlet streams).
PhaseSeparator	- Mass balance of vapor and liquid outlet depending on the parameters of the inlet stream; - pressure equality of vapor and liquid streams to the inlet stream; - temperature equality of vapor and liquid streams to the inlet stream. Optionally: - composition balance of the vapor and liquid outlet depending on the parameters of the inlet stream (not defined for a pure fluid).

For more information about available classes and functions please see the code comments.

6 Using the package

All the calculations should be preferably done in a separate input program file. Examples of such files are located in the /CryoSolver-1.0.0/examples/ folder.

6.1 Main parts of the user program

The calculation is divided into the following main steps:

1. Solver import:

```
import cryosolver.solver as CS
```

2. Reset of global variables. As global variable (*mainvar* class object) is used by the solver, it needs to be created or reset before each new simulation to delete old values:

```
CS.start()
```

If necessary, by means of this function it is possible to change a preferable zero state for the exergy calculation. By default the zero state is at 300 K and 1 bar.

```
CS.start([303,1.013])
```

Furthermore, if a cycle with known mixture composition for all streams should be simulated, it is possible to put in here an array of compositions to skip the calculation of composition balances ($x1$ and $x2$ - value of mole fractions of mixture components for each of 3 streams):

```
CS.start(fixcomposition=[[x1,x2],[x1,x2],[x1,x2]])
```

3. Fluid specification:

```
CS.setup.fluidsetup("Nitrogen.FLD", "Oxygen.FLD")
```

   Please refer to the *solver.py* description and notes in the program code.

4. Assignement of cycle components by the user with class objects included into the package. Depending on given component specification the required balance functions are added by the solver into the list of equations (*cryosolver.solver.mainvar.funclist*). Thus, for example, if the turbine with 85% efficiency is assigned between the 1^{st} and 2^{nd} streams,

```
CS.Turbine(1,2,0.85)
```

   the following functions will be added to *cryosolver.solver.mainvar.funclist*:

   - Energy balance: $EnergyB() = h_{inlet} - h_{outlet} - \eta_s \cdot (h_{inlet} - h_{outlet}\big|_s)$
   - Mass balance: $MassB() = \dot{m}_{inlet} - \dot{m}_{outlet}$

   Here inlet and outlet indexes state for streams 1 and 2 respectively. In case if turbine power (here 700 kW) will be given:

```
CS.Turbine(1,2,0.85,power=700)
```

   additional function will be added:

   - Energy balance: $PowerB() = h_{inlet} \cdot \dot{m}_{inlet} - h_{outlet} \cdot \dot{m}_{outlet} - power$

5. Specification of fluid composition. This step should be done only for mixtures and if no fixed composition array is specified by *CS.start()*:

```
CS.setup.parameterset(1, "Z",[0.78,0.22])
```

6. Fixing the known parameters. The respective functions are added into the list. For example, to fix the temperature and pressure at the turbine outlet as 100 K, 1 bar, it should be written:

```
CS.setup.parameterset(2,"T",100)
CS.setup.parameterset(2,"P",1)
```

7. Additional functions can be added manually into the system. For example, fixing the temperature difference through the turbine as 100 K:

```
def userfunction():
    return CS.mainvar.streamlist[0].T - CS.mainvar.streamlist[1].T -
        100
CS.mainvar.funclist.append(userfunction)
```

   Please note, here the indexes are related to the internal solver array, so for the stream i the index $i-1$ should be used (see rules below).

8. Specification of an array of initial values. It depends on specific problem, how close they should be to the solution. Usually, for a simple case, any physically meaningful values could be assigned. The array should mandatory have 3 list elements for temperature, pressure and mass flow and an array of mole compositions except the last component of the mixture:

```
CS.setup.initvaluesset( ([300,200], [5,1], [1,1],[[0.78],[0.78]]) )
```

   Here 0.78 is the initial mole fraction of nitrogen, the one of oxygen will be calculated as (1−0.78). In case of a pure fluid or if the composition is fixed at the beginning, the initial values only for temperatures, pressures and mass flows need to be specified:

```
CS.setup.initvaluesset( ([300,200], [5,1], [1,1]) )
```

9. Finding the solution and result output. For definition of an optimization/case study functions please refer to example files.

```
s = CS.solution("TP")
CS.txtoutput(s)
```

   It is possible to perform a simulation on temperature/pressure ("TP") or on temperature/entropy ("TS") based calculation. The second one is more stable for ideal gases and 2-phase region simulations. However, the initial values should be anyway setup as temperature/pressure based and will be automatically recalculated into the entropy. Please note, that manual printing of the solution in case of "TS" simulation will return the entropy values in the middle. In the *CS.textoutput()* function they are already transformed back to show pressures.

The execution of the above code for nitrogen and oxygen (the full code is saved in /CryoSolver-1.0.0/examples/Example_0.py) will give the output:

```
Number of specified streams:  2
Specified open streams:
inlets: [0] outlets: [1]
************************************************************
name: Turbine
eta_is: 0.85
inlet: 0
outlet: 1
pressure_ratio: 15.112917789718212
power: 699.9999999999162
exergy_loss: 406.64579563013615

Temperatures:
200.0, 100.0

Pressures:
15.112917789718212, 1.0

Massflows:
5.093246893050264, 5.093246893050264

Compositions:
[0.6 0.4], [0.6 0.4]
************************************************************
Success: True
```

By the specified open streams all the inlets and outlets of the cycle are shown regardless of whether they belong to a closed loop. This output allows to find possible errors in stream index definitions.

6.2 Important simulation rules

6.2.1 Rule to define a closed cycle

As well as in most of industrial simulation software, it is not possible to directly setup a fully closed cycle. This would cause an over-definition of the equation system due to looping of the mass flow and composition balance from one stream to another. Thus, the cycle should be defined with at least **one opening** and two **closing equation of pressure and temperature balance** (Figure 1).

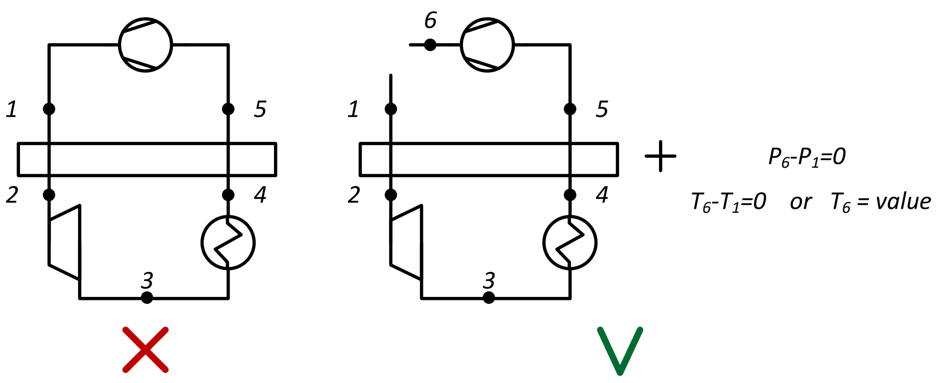

Figure 1: Rule to define close cycle.

For comfort, the additional command including the setup of the pressure equality of two streams is added to the solver:

```
CS.setup.closeloop(1,6)
```

The temperature balance is not included, so temperatures can be specified at both ends if required.

6.2.2 Stream index definition

The streams need to be **numbered starting with 1** and not to have breaks in between. This is important for the solver to find the total number of variables.
However, the solver shifts these indexes to start the assignment of internal array elements with 0. Thus, to access the stream properties from the solver (for example, defining user functions with *cryosolver.solver.mainvar.streamlist*) **the (i-1) index** should be used for each stream i.
Example: to set the temperature of the first cycle stream as 300 K using the module function:

```
CS.setup.parameterset(1,"T",300)
```

To print the temperature of the first stream from the *streamlist* array:

```
print(CS.mainvar.streamlist[0].T)
```

6.3 Solution failures

The solution will only converge if the problem statement allows to find a definitive solution. Thus, all the components and parameters need to be specified in such a way that they don't contradict with physical laws and specified functions governing all these variables. Moreover, the specification of initial values for a solver can have a high impact on the convergence. If no solution is found:

- check indexes of all specified streams and parameters (according to the stream assignment rule);
- check physical statement of the problem (for example, if solver finds a solution with negative values of variables, this most likely indicates incompatibility of set values);
- change initial values;
- for a more complex problem solver type and settings can be changed inside the *solver.py* module;
- balance equations could be multiplied by some weight coefficients to change the tolerance of solution;
- variables could be multiplied by some scaling factors before being passed to the solver and then scaled back inside the mainfunction() to change the solver sensitivity;
- in case of problems with fluid properties refer to the Refprop documentation `https://www.nist.gov/srd/refprop`.

Additionally, there can be failures in the calculation due to insufficient accuracy of solver solution or optimisation of a function with many peaks. These issues need to be solved separately for each specific problem.

7 Examples

In the /CryoSolver-1.0.0/examples/ folder two simple calculation examples are shown. More complicated cycles were simulated by the authors within their research work and will be published in dedicated theses. The syntax of example programs is explained directly in the code.

7.1 Simple reverse Brayton cycle

Please refer to the *Example_1.py* for the simple calculation of one specific state, *Example_1_optimizer.py* for an example of efficiency optimisation definition and *Example_1_case study.py* for a case study based on variation of a turbine outlet pressure. The cycle flow diagram is shown on the Figure 2.

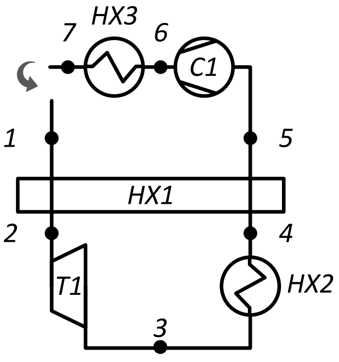

Figure 2: Flow diagram of the Example 1. T1 – turbine, C1 – compressor, HX1 – inner counter-flow heat exchanger, HX2 – simple heat exchanger, HX3 – aftercooler.

7.2 Claude cycle

Example_2.py illustrates the simulation of a Claude cycle. A nitrogen-oxygen mixture with mole composition equivalent to their presence in the air is used to show the result of a phase separator simulation for this mixture. The flow diagram of the cycle with respective stream indexes is shown in Figure 3.

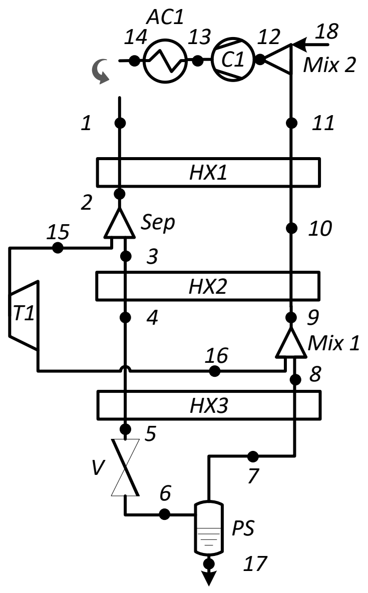

Figure 3: Flow diagram of the Example 2. Sep - separator, T1 – turbine, C1 – compressor, HX1, HX2, HX3 – inner counter-flow heat exchangers, AC1 – aftercooler, Mix 1 and Mix 2 - mixers, PS - phase separator, V - valve.

Please note: For a successful simulation of a liquefaction cycle it is necessary to input initial parameters of an inlet flow of a phase separator in such a way, that it is already in 2-phase region. Otherwise, the convergence is not guaranteed.

Acknowledgement

This package was developed at the TU Dresden within the Work Package 4 of the EASITrain Project. EASITrain – European Advanced Superconductivity Innovation and Training. This Marie Sklodowska-Curie Action (MSCA) Innovative Training Networks (ITN) receives funding from the European Union's H2020 Framework Programme under grant agreement no. 764879, `http://easitrain.web.cern.ch/`.

References

[1] E. W. Lemmon, I. H. Bell, M. L. Huber, and M. O. McLinden, "NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP, Version 10.0, National Institute of Standards and Technology," 2018.