OpenACHP notes

By Nicholas Fette, 2017-08-08

# Required libraries

## General

The code began in development for Python 2.7 or so, and has since been (mostly) converted to Python 3 with no regard for backwards compatibility. The code requires the following Python libraries for convenience:

* Python 3 standard library (hashlib, pickle, sys, traceback, json, decimal, ctypes, enum, os, collections)
* Matploblib
* Numpy
* Scipy
* Tabulate
* CoolProp
* Jupyter (for interactive notebooks)

Some additional functionality requires other software with proprietary licenses:

* NIST REFPROP database
* F-Chart Software’s EES, version ca. 2016s

## LiBr-water mixture properties

For comparison and presentation purposes, the interactive notebook file compare\_libr\_props.ipnyb is included in the src/ folder. The comparisons were presented in my dissertation prospectus, ca. April 2016. The sources of property lookups include:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reference | Implementation | Interface | Used in source files | Status |
| Patek and Klomfar | Libr\_props.py | None/native | Libr3.py | Appears to be implemented and checked |
| Patek and Klomfar | CoolProp incompressible mixture model | Libr\_props2.py | Libr.py, Libr2.py (preliminary models only) | Enthalpy correction not implemented in interface file, but in comparison notebook |
| ASHRAE | EES (LIBR.dll) | Ees\_interface2.py | Comparison only |  |
| Yuan and Herold / SSC | EES (SSCLiBr.dll) | Ees\_interface2.py | Comparison only |  |
| YUAN and Herold / SSC | Library available upon request (possibly C code) |  | None |  |

## Aqua-ammonia mixture properties

For comparison and presentation purposes, the file compare\_ammonia\_props.ipynb is included in the src/ folder. The comparisons were presented in my dissertation prospectus, ca. April 2016. The sources of property lookups include:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reference | Implementation | Interface | Used in source files | Status |
| Ibrahim and Klein | EES (nh3h2o.dlp) | Ammonia\_props.py and Ees\_interface.py | Ammonia1.py | Adds some features to the raw library file such as Gibbs energy and named functions with keyword arguments |
| Tillner-Roth and Friend | REFPROP | CoolProp | Comparison only |  |
| Patek and Klomfar | None |  |  |  |
| Melinder, Skovrup | CoolProp[[1]](#footnote-1) |  |  | Incompressible model |

# Developments

## Solvers

There are several objectives for the implementation of model solvers for the purposes of OpenACHP:

1. Obtaining (open source) property lookups and consistent reference states
2. Solving accurately and smoothly
3. Solving system model quickly
4. Solving robustly, by catching infeasible inputs and classifying the infeasibility

The first objective is the simplest from a technical perspective. The second and third (accuracy and speed) are competing objectives, and several tricks and trade-offs have been considered especially with respect to property lookups and heat exchanger equations. For example, my dissertation prospectus outlines how in solving heat exchanger problems, the use of an integral equation instead of a simplified equation (LMTD or NTU) will improve accuracy for non-linear or non-constant properties. However, multiple approaches exist to implement the integral equation. To use the integral equation with only numerical sources for stream properties, there are two basic steps:

1. Given a cold and a hot stream defined at inlets, evaluate the maximum heat rate, Q\_max, that can be feasibly transferred (such that hot stream is always hotter than cold stream)
2. Given a cold and a hot stream defined at inlets and a total amount of heat to be transferred, determine if the given heat rate is feasible (equivalently, determine effectiveness), and evaluate the overall heat exchange coefficient (and outlet states).
3. – Or – given a cold and a hot stream defined at inlets and an overall heat exchange coefficient, determine the expected heat rate (and outlet states).

The second and third steps are alternatives analogous to using, respectively, the effectiveness or NTU as the input depending on the nature of the heat exchange problem. Notably, if the streams are fixed, then the first step is only required once, and changing the heat rate requires only updating the calculations for the second step. This observation may be applied to speed up optimization. The third step does not require pre-computing the maximum feasible heat rate, since any input yields a feasible problem. However, for a problem where repeated lookup or both types of inputs are anticipated, one can use a series of evaluated points to establish a fitting spline for forward and reverse interpolation. The shape of this curve reflects the non-linearity of the problem (and presumably could be non-dimensionalized as a further study). There are some files that show these curves for a few cases typical for this project, and a brief document was compiled at some time after my dissertation proposal (TODO: give file names).

Meanwhile, here are some required choices to implement those steps:

1. To evaluate (q, T) points in each stream, use direct calls to property functions, or use a spline curve for forward and reverse lookups of (q, T). Direct calls to property functions may be slow due to the use of an iterative solvers for some sets of inputs. For a spline approach, choose the range and number of sample points.
2. Use an off-the-shelf numerical optimization/integration routine (choose which one)
3. …

Each of these choices effects the accuracy and smoothness of the model output, which can influence the performance and results of the optimizer.

The final objective is incorporated into the code design, using object-oriented models to encapsulate the cycle model and the system model (including heat exchange). To instantiate a system model, the user must first instantiate a cycle object, which will throw an exception if its inputs are infeasible. Only once the cycle is solved can the user proceed to evaluate the system, which will throw an exception if it violates the rules governing heat exchanger performance. Additionally, some constraints are only applied via the optimization routine.

## Optimization

The first pass at solving this problem was to use EES and its built-in solvers and optimization routines. However, the optimization routine would regularly abort due to infeasible inputs. In spite of this behavior, a method to generate results was carried out, using manual intervention to narrow the input search range until a boundary optimum was found, and then shifting the input search range, and so forth, until an optimum was found interior to the input search range. Using this method, I performed a few parametric studies on sensitivity to total heat exchanger area and heat rejection temperature. However, the manual approach was so slow and tedious that results could not be taken as reproducible. Using only the academic license for EES, we were not able to automate the method for manual intervention, nor able to call the model from other optimization tools via the EES command line interface.

Next, I implemented cycle and system models in Python. With these models, we are able to apply a wider variety of optimization tools. Thus the second pass at optimization was to apply the optimization library provided with the popular Scipy library, which has about 11 distinct optimization routines. The file aqua\_case\_studies2.py was used to attempt optimization using each of the available routines, calling on the aqua-ammonia chiller system modeled in system\_aqua1.py. Some of the optimization routines lack the ability to input constraints, therefore this system model includes a mapping for the input range intended to allow the same search areas to be passed to all the optimizers. For each applicable routine, the optimizer was applied over a range of heat rejection temperatures, and in each case the history of the optimizer’s attempted inputs was saved as a Python pickle file in a directory of the OpenACHP project named data, data1, etc. (However, the pickle files were not submitted for storage in the code repository.) These files can be read back in to trace the course of the optimizer and look for smoothness in the parametric study. My recollection is that the parametric studies did not look smooth and exhibited an unexpected non-monotonicity. I did not attempt to optimize the LiBr cycle. Related files include:

* System\_aqua1.py
* Aqua\_case\_studies2.py

The third pass at optimization was to use GenOpt, a generic optimization tool coded in Java and maintained by Michael Wetter at LBNL. While practicing how to use GenOpt, I identified a bug, and thanks to the package being open source, was able to proceed with my work and contribute a bug fix. Anyway, GenOpt interacts with my model via the command line, and requires as its input several files specifying how to invoke the model, how to evaluate the objective function, and solver and parameter options. Using different optimizer options, several sets of trials were created and run, which are stored in the opt1, opt2 … directories of the project repository. Again, for any given optimization trial, the output files generated by the optimizer are sufficient to playback the history of sampled inputs. However, I might not yet have a script that specializes in formatting these results for display.

1. See https://github.com/CoolProp/CoolProp/issues/341 [↑](#footnote-ref-1)