By David Kayanan 2020

1 Overview

This is a model of a **cooling tower** (CT), which functions as the final heat rejection device of building HVAC systems and district cooling systems. This is motivated by calculating how much of the rejected heat is sensible vs. latent heat, or for transferring CEA rejected heat results into CFD software.

<table>
<tr><td>Inputs</td><td>1. Design data = CT design specs

2. Operational data (e.g. hourly resolution)
   a. ambient temperature (dry bulb) and relative humidity
   b. Bldg/HVAC total heat rejected (kW)</td></tr>
<tr><td>Outputs</td><td>CT exhaust air data
1. thermodynamic state information of humid air (e.g. dry bulb temp, relative humidity, humidity ratio, etc.)
2. air mass flow [kg dry air/s]
3. exhaust air speed [m/s]

Other CT operational data (e.g. hot and cold water temperatures, evaporation loss, etc.)</td></tr>
</table>

By David Kayanan 2020

2 Implementation brief

Dependencies

Must haves

- psychrolib 2.4.0 for psychrometrics
- numpy, pandas, Matplotlib

Optional

- pint 0.9 for explicit units

Method

Here is the basic idea behind how the solution is carried out:

1. **Solve water circulation**
   Based on the heat load and the ambient air conditions, must determine the hot water temp (HWT), cold water temp (CWT) and the water flow rate s.t.

   - $heat\ load = (HWT\text{-}CWT) * water\ mass\ flow * water\ heat\ capacity$
   - Design values for HWT and the water flow rates are met, if possible*
   - CWT vs. wet bulb characteristic is followed, if possible*

**The CT solution covers a wide range of conditions that the CT can operate in, which can be classified as 1) no-load and very low load conditions (<20% nominal); and 2) very hot and humid ambient air conditions. To get realistic results, some variables are allowed to float within appropriate limits, which represent CT control.*

2. **Solve water to air heat transfer**

   Energy balance is applied in the water-to-air heat transfer in a CT. This allows us to solve the exhaust air state and the air flow rate (kg dry air/s). The energy balance equation is given by:

$$h_1 \cdot water\ flow + \ h_3 \cdot air\ flow = h_2 \cdot return\ water\ flow + \ h_4 \cdot air\ flow$$

   Where,

state 1	Entering water at the HWT
state 2	Returning water at the CWT
state 3	Ambient air
state 4	Exhaust air, with assumed relative humidity

3. **Solve the exhaust air flow**
   Some CFD software need the speed of the air stream, and not just the mass flow rate, as in the case of AnsysFluent. Auxiliary steps are carried out to calculate additional information.

As an example, the air speed is determined by getting the specific volume after determining state 4, multiplying the air flow and then dividing by the exhaust area.

3 Workflow / minimal example

The entire simulation is implemented in the function **simulate_CT()**. It calculates all CT units and all time periods at once (vectorized). It requires the following arguments:

heat_load	defines the CT heat load (2D ndarray; axis 0 is interpreted as time; axis 1 is interpreted as the CTs defined in *CT_design*
CT_design	defines the CT units and their specs
air_i	defines the ambient air conditions (for every time-step)

The workflow can be summarized as:

1) Load the CT units

CT design

```
In [2]: CT_design = read_CTcatalog('CT catalog v2.pkl')
        CT_design
```

Out[2]:

CT	Capacity [kW]	HWT [°C]	CWT [°C]	WBT [°C]	approach [C°]	range [C°]	water flow [kg/s]	air flow [kg/s]	L/G	evap loss [%]	Fan diameter [m]	CT perf slope	CT
100	100	38	32.8	29.734008	3.065992	5.2	4.596264	2.277240	2.018349	0.782234	0.9	0.647737	13.5
150	150	38	32.8	29.734008	3.065992	5.2	6.894396	3.415860	2.018349	0.782234	0.9	0.647737	13.5
200	200	38	32.8	29.734008	3.065992	5.2	9.192528	4.554480	2.018349	0.782234	1.2	0.647737	13.5
300	300	38	32.8	29.734008	3.065992	5.2	13.788792	6.831720	2.018349	0.782234	1.2	0.647737	13.5
400	400	38	32.8	29.734008	3.065992	5.2	18.385057	9.108960	2.018349	0.782234	1.6	0.647737	13.5
500	500	38	32.8	29.734008	3.065992	5.2	22.981321	11.386200	2.018349	0.782234	1.6	0.647737	13.5
800	800	38	32.8	29.734008	3.065992	5.2	36.770113	18.217921	2.018349	0.782234	2.2	0.647737	13.5

In this example, we loaded a catalog-style CT design (i.e. CTs are unique and are arranged in increasing size). CT units need not be unique and can be arranged as the CT units serving each building in a group of buildings.

2) Define the ambient air conditions

1) Set ambient air conditions

```
In [ ]: air_i, WBT = set_ambient(Tamb, RH)
```

By David Kayanan 2020

Pass iterables of the ambient dry bulb temperature and relative humidity, and humidair instances are returned (class of humid air states), along with the wet bulb temperature of each.

3) Set up the heat load

This would come from the heat rejected (kW heat) by CEA. In the following example, the CTs are just assumed to operate at full-load during the entire period:

2) Set heat load

In [9]:

```
HL_pu = 1
CT_load = Q_(np.tile(CT_design['Capacity [kW]'].values*HL_pu, (len(air_i), 1)), 'kW')
CT_load
```

Out[9]:

100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000
100	150	200	300	400	500	800	1000	1700	2500	5000

kilowatt

4) Simulate

Simply call **simulate_CT()**, and the results are returned in a dictionary.

3) Simulate

In [ ]:

```
res = simulate_CT(CT_load, CT_design, air_i, pump_ctrl='Range limit', fan_ctrl=True)

# Returned object is a dict of results
airflow = res['air flow']
waterflow = res['water flow']
HWT = res['HWT']
ret_waterflow = res['return water flow']
air_o = res['air_o']
```

Most of the results are 2D structures with axis 0 : time, axis 1: CTs (as defined in the 2nd argument to **simulate_CT()**). The outputs would look like:

In [11]:
```
air_o
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	10
0	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH
1	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH
2	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH	Humid air at 35.82 °C, 0.950 RH
3	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH	Humid air at 35.80 °C, 0.950 RH

In [13]:
```
airflow.round(2)
```

Out[13]:

$$\begin{pmatrix}
2.21 & 3.31 & 4.42 & 6.63 & 8.84 & 11.05 & 17.68 & 22.1 & 37.57 & 55.25 & 110.5 \\
2.21 & 3.31 & 4.42 & 6.63 & 8.84 & 11.05 & 17.68 & 22.09 & 37.56 & 55.24 & 110.47 \\
2.21 & 3.31 & 4.42 & 6.63 & 8.84 & 11.05 & 17.68 & 22.1 & 37.56 & 55.24 & 110.48 \\
2.21 & 3.31 & 4.42 & 6.63 & 8.84 & 11.05 & 17.67 & 22.09 & 37.56 & 55.23 & 110.47 \\
2.21 & 3.31 & 4.42 & 6.62 & 8.83 & 11.04 & 17.67 & 22.08 & 37.54 & 55.21 & 110.41 \\
2.21 & 3.31 & 4.42 & 6.62 & 8.83 & 11.04 & 17.66 & 22.08 & 37.54 & 55.2 & 110.4 \\
2.21 & 3.31 & 4.42 & 6.62 & 8.83 & 11.04 & 17.66 & 22.08 & 37.53 & 55.19 & 110.38 \\
2.21 & 3.32 & 4.42 & 6.63 & 8.85 & 11.06 & 17.69 & 22.11 & 37.59 & 55.28 & 110.57 \\
2.21 & 3.32 & 4.43 & 6.64 & 8.86 & 11.07 & 17.72 & 22.15 & 37.65 & 55.37 & 110.74 \\
2.22 & 3.33 & 4.43 & 6.65 & 8.87 & 11.09 & 17.74 & 22.17 & 37.69 & 55.43 & 110.86 \\
2.22 & 3.33 & 4.44 & 6.66 & 8.88 & 11.1 & 17.76 & 22.2 & 37.74 & 55.49 & 110.99 \\
2.22 & 3.33 & 4.44 & 6.66 & 8.88 & 11.1 & 17.77 & 22.21 & 37.76 & 55.52 & 111.04 \\
2.22 & 3.33 & 4.44 & 6.66 & 8.88 & 11.1 & 17.75 & 22.19 & 37.73 & 55.48 & 110.96 \\
2.22 & 3.33 & 4.44 & 6.65 & 8.87 & 11.09 & 17.74 & 22.18 & 37.7 & 55.45 & 110.89
\end{pmatrix} \mathit{kilogram/second}$$

4 Final notes

What we haven't seen so far is how the CT designs are produced. I developed my own CT design method, which can also be implemented. Alternatively, if users do have design specs they want to use, then they can just upload this information.