Alarko Holding New Ideas Competition

Model-Based Component Matching & Thermal Simulation

By Amneh Jaber

Topic:

This idea is presented under code 3 (Development and Creativity in New Products, Services, and Business Procedures/Processes).

Developed a model-based component matching system for Direct Expansion (DX) coils to calculate capacity, superheat temperature, two-phase fraction and mass flow while focusing on achieving a target superheat temperature using ACHP library and iterative methods.

Simultaneously, conducted simulations of water coils under various Eurovent test conditions for Alarko Fan Coil Units (FCU) to find water side pressure drop, water flow rate, condensate flow rate, total, sensible and latent heats. Both DX and FCU water coil calculations were done using Python while considering various factors such as water and air properties, coil geometry, and operating conditions.

These approaches significantly improve the efficiency and dynamism of thermal calculations and simulations. The resulting data can be easily visualized in graphical plots and/or exported to Excel datasheets for further analysis or reporting.

Aim:

The primary aim of this project is to create a convertible, dynamic, and reliable model-based calculation system that goes beyond its application to DX coils or FCU coils. Instead, it serves as a foundational tool for solving a wide range of R&D challenges. This system's capabilities extend to simulating thermal calculations, and generating a dynamic database that encompasses all possibilities, which can later be integrated into our selection software, enhancing its functionality across various applications.

Introduction & Implementation of the Idea:

The Research and Development (R&D) department is continuously seeking solutions to address complex project requirements and provide reliable tests and precise calculations for our products. While traditional methods and engineering practices play an important role in R&D, the integration of programming languages can be used as a powerful tool in this process.

One of the most known and widely used programming languages is Python, Python is a high-level language known for its simplicity, readability, and extensive libraries. It offers a wide range of packages and modules specifically designed for scientific computing and numerical

simulations. By integrating Python into our projects, we can create efficient, customized solutions and dynamic databases.

The initial step in implementing this idea involves identifying specific projects where a model-based approach or thermal calculations can be applied. For instance, in the DX Coil project, capacity calculations were required to select a suitable distributor (Phase 1). Also, FCU certification can be attained by ensuring that our declared values for Eurovent align with the actual test results (Phase2).

- ***Phase 1 : Model-Based Component Matching for DX Coils:***

The main objective of this project phase is to recommend suitable DX Coil distributors that can later be integrated into Airovision dll, this will be done by conducting a series of calculations on DX coils within Air Handling Units (AHU). This involves assessing their performance under various conditions, categorizing them by capacity, and finally making distributor recommendations based on predefined criteria. The code is responsible for reading more than 17000 combinations of input data related to Direct Expansion (DX) coils, executing calculations for each coil, determining suitable distributors, and storing the results in an output CSV file. Prior to this code, an initial version was developed to generate the potential P25 DX coil circuiting combinations (2288 combinations).

The code accompanied by comments for clarification is shown below.

```
# DXCoils

from __future__ import division, print_function, absolute_import
import CoolProp as CP
from CoolProp.CoolProp import PropsSI
from ACHP.FinCorrelations import FinInputs
from ACHP.Evaporator import EvaporatorClass
import datetime
import pandas as pd

# Get the start time of the program
start_time = datetime.datetime.now().replace(microsecond=0)

# define the path of DXCoils_P25.csv file
df = pd.read_csv(r"C:\Users\*****\Desktop\DXCoil_ACHP\DXCoils_P25_Input.csv", encoding='unicode_escape')
DXCoils_No = 5
# Adding new columns to the existing dataframe in .csv file:
df['Capacity'] = ''
df['massflow'] = ''
df['SuperheatTemp[k]'] = ''
df['2-PH_fraction'] = ''
df['Dist_No'] = ''
df['Chosen_Dist'] = ''
df['Chosen_Dist_Cap'] = ''
df['Chosen_Dist_CircNo'] = ''

FinsTubes = FinInputs()

FinsTubes.Tubes.OD = 0.01092      # Tube outer diameter in [m].
FinsTubes.Tubes.ID = 0.00983      # Tube inned diameter in [m].
FinsTubes.Tubes.Pl = 0.022        # Distance between number of tubes in flow direction in [m].
FinsTubes.Tubes.Pt = 0.0254       # Distance between center of tubes orthogonal to flow direction in [m].
FinsTubes.Tubes.kw = 398          # Tubes thermal conductivity for cupper [W/m.k].

FinsTubes.Fins.FPI = 10           # Fins per inch.
FinsTubes.Fins.Pd = 0.00157       # 2*amplitude of wavy fin in [m].
FinsTubes.Fins.xf = 0.0055        # 1/2 period of fin in [m].
FinsTubes.Fins.t = 0.0001         # Fin thickness in [m]. (This value isn't from P25 drawings, just 0.1
mm)
FinsTubes.Fins.k_fin = 237        # Fins thermal conductivity for Aluminum [W/m.k].
```

```
FinsTubes.Air.Tdb = 308.15        # Air drybulb temperature in [Kelvin].
FinsTubes.Air.p = 101325          # Air pressure in [Pa].
FinsTubes.Air.RH = 0.54           # Air relative humidity [%].
FinsTubes.Air.FanPower = 0        # Fan power in [W].

Ref = 'R410A'
# choose between: 'HEOS','TTSE&HEOS','BICUBIC&HEOS','REFPROP','SRK','PR'
Backend = 'TTSE&HEOS'
AS = CP.AbstractState(Backend, Ref)
Sat_Pressure = PropsSI('P', 'T', 280.15, 'Q', 1, Ref)

for i in range(DXCoils_No):
    # Number of tubes per row (changing values from excel.csv).
    FinsTubes.Tubes.NTubes_per_bank = df['TubesPerRow'][i]
    # Number of circuits (changing values from excel.csv).
    FinsTubes.Tubes.Ncircuits = df['Circuits'][i]
    # Number of rows (changing values from excel.csv).
    FinsTubes.Tubes.Nbank = df['Rows'][i]
    # Length between tube sheets in [m] (changing values from excel.csv).
    FinsTubes.Tubes.Ltube = df['Width'][i]/1000
    # Air volumetric flow rate in [m3/s] = 1.5m/s * ModuleWidth * ModuleHeight / 1000000
    FinsTubes.Air.Vdot_ha = 1.5*df['Width'][i]*df['Height'][i]/1000000

    kwargs = {'AS': AS,
            'mdot_r': 0.5,
            'psat_r': Sat_Pressure,
            'Fins': FinsTubes,
            'FinsType': 'WavyLouveredFins',  # WavyLouveredFins, HerringboneFins, PlainFins
            'hin_r': PropsSI('H', 'P', Sat_Pressure, 'Q', 0.15, Ref), # [J/kg], Q value is assumed 0.15
            'Verbosity': 0,
            }

    Evap = EvaporatorClass(**kwargs)
    Evap.Update(**kwargs)

    #Superheat target is 5 Kelvin.
    superheatTarget = 5
    massflow = Evap.mdot_r

    for j in range(100):
        try:
            Evap.Calculate()
            superheat = Evap.Tout_r-Evap.Tdew_r
        except Exception as e:
            if ('Not possible to determine whether pressure is inside or not' in (str(e))):
                superheat=-10
            else:
                superheat=10

        superheat_Err = superheatTarget - superheat
        if (j>70):
            massflow = massflow - 0.0005*superheat_Err
        else:
            massflow = massflow - 0.002*superheat_Err
        massflow=max(massflow, 0.0001)
        Evap.mdot_r = massflow

        if abs(superheat_Err) <= 0.1:
            break

    if Evap.Q <= 50000:
        df['Dist_No'][i] = 1
    elif Evap.Q > 50000 and Evap.Q <=100000:
        df['Dist_No'][i] = 2
    elif Evap.Q >100000 and Evap.Q <=150000:
        df['Dist_No'][i] = 3
    else:
        df['Dist_No'][i] = 4
```

```
    df['Capacity'][i] = Evap.Q / df['Dist_No'][i]
    df['massflow'][i] = massflow
    df['SuperheatTemp[k]'][i] = Evap.Tout_r - Evap.Tdew_r
    df['2-PH_fraction'][i] = Evap.w_2phase
    print("Total Capacity: ", round(Evap.Q, 2), "Dist_No: ", df['Dist_No'][i], " Massflow: ", round(
        massflow, 3), "Superheat Temp: ", round(superheat, 2), "2PH fraction: ", Evap.w_2phase, " DX Coil
Number: ", i)

# Write the calculated values to the csv file
df.to_csv("DXCoils_P25_Output.csv", index=False)

for k in range(DXCoils_No):
    #Flag to indicate if a match is found for the current k
    match_found = False
    for l in range(33):
        if ((df['Dist_Cap'][l] * 1000 * 1.1 >= df['Capacity'][k]) and (df['Capacity'][k] >=
df['Dist_Cap'][l] * 1000 * 0.9) and (abs(df['Dist_CircNo'][l] - df['Circuits'][k]) < 3)):

            df['Chosen_Dist'][k] = df['Distributor'][l]
            df['Chosen_Dist_Cap'][k] = df['Dist_Cap'][l]
            df['Chosen_Dist_CircNo'][k] = df['Dist_CircNo'][l]
            match_found = True
            #Exit the loop once a match is found
            break

    if not match_found:
        df['Chosen_Dist'][k] = "No Match"

df.to_csv("DXCoils_P25_Output.csv", index=False)

# Print the excution time
end_time = datetime.datetime.now().replace(microsecond=0)
print('Program excution time is:', end_time - start_time)
```

A brief explanation of the key components and what the code is doing step by step:

- It imports several Python libraries and modules, including CoolProp for thermophysical properties, pandas for database, and ACHP for HVAC simulations.
- It reads input data from a CSV file located at a specified path. This data contains information about the DX coils, such as their geometrical properties and operating conditions.
- It defines a set of parameters related to the geometry and operating conditions of the DX coils. These parameters are stored in the `FinsTubes` object as inputs.
- It then enters a loop to perform calculations for each DX coil specified in the input data, Inside this loop:
    - Sets various parameters in the `FinsTubes` object based on the data read from the CSV file.
    - Creates an instance of an evaporator (`Evap`) using the ACHP library and updates its properties based on the provided parameters.
    - Calculates the Evaporator (in this case, the DX Coil) performance by iteration, specifically focusing on achieving a target superheat of 5 Kelvin by adjusting the refrigerant mass flow rate using ACHP.
    - Stores various results, such as capacity, mass flow rate, superheat temperature, and two-phase fraction.
- After the calculations for all DX coils are completed, the number of distributors needed for each DX coil is calculated based in the total capacity (from 1 up to 4 distributors).

- The code later finds a suitable distributor for each DX coil from a predefined list based on capacity per distributor and circuit number.
- The final results are stored and printed to a CSV file for further analysis or reporting.

The cropped photo down below shows small part of the resulting excel data sheet (green cells are some of the calculated outputs).

Circuiting	Remove Duplicates Circuit Tol is +-2	DX Coil Capacity per Distributor [W]	Total DX Coil Capacity [W]	Ref. Massflow [Kg/s]	Superheat Temp [K]	2-Phase fractio	Chosen Distributor
2 Circuit x 2 Pass + 16 Circuit x 4 Pass	61423418114664.50702	38221.50234	114664.507	0.61417378	5.09821712	0.9358691	G5 DISTRIBUTOR 1/4" DS6 ND6.5
2 Circuit x 4 Pass + 16 Circuit x 8 Pass	61443418153092.35724	38273.08931	153092.3572	0.81998107	5.08426338	0.9533633	G5 DISTRIBUTOR 1/4" DS6 ND6.5
2 Circuit x 8 Pass + 19 Circuit x 4 Pass	81424621157208.40256	39302.10064	157208.4026	0.84198034	5.09730249	0.9396523	G5 DISTRIBUTOR 1/4" DS6 ND6.5
3 Circuit x 8 Pass + 11 Circuit x 4 Pass	61423414118849.76001	39616.58667	118849.76	0.63661901	5.08033377	0.9424845	G5 DISTRIBUTOR 1/4" DS6 ND6.5
3 Circuit x 8 Pass + 17 Circuit x 4 Pass	81424620158287.94652	39571.98663	158287.9465	0.84778062	5.09126706	0.9409286	G5 DISTRIBUTOR 1/4" DS6 ND6.5
3 Circuit x 6 Pass + 25 Circuit x 2 Pass	61423428104665.64586	34888.54862	104665.6459	0.56065287	5.08576116	0.9204291	G2 DISTRIBUTOR-1/4" DS10-01
3 Circuit x 4 Pass + 28 Circuit x 2 Pass	61423431102714.99321	34238.33107	102714.9932	0.55019861	5.09178593	0.9159695	G2 DISTRIBUTOR-1/4" DS12-03
4 Circuit x 2 Pass + 14 Circuit x 6 Pass	81424618161894.80228	40473.70057	161894.8023	0.86706539	5.09985952	0.9434117	G5 DISTRIBUTOR 1/4" DS6 ND6.5
4 Circuit x 2 Pass + 15 Circuit x 4 Pass	61423419113739.31986	37913.10662	113739.3199	0.60924852	5.0821827	0.9345407	G5 DISTRIBUTOR 1/4" DS6 ND6.5
4 Circuit x 4 Pass + 15 Circuit x 8 Pass	61443419152460.53436	38115.13359	152460.5344	0.81657416	5.09188713	0.952497	G5 DISTRIBUTOR 1/4" DS6 ND6.5
4 Circuit x 8 Pass + 15 Circuit x 4 Pass	81424619160766.1356	40191.5339	160766.1356	0.86105408	5.09028756	0.942321	G5 DISTRIBUTOR 1/4" DS6 ND6.5
4 Circuit x 10 Pass + 16 Circuit x 6 Pass	61443420150605.20728	37651.30182	150605.2073	0.8066349	5.09335305	0.9515775	G5 DISTRIBUTOR 1/4" DS6 ND6.5
4 Circuit x 2 Pass + 21 Circuit x 4 Pass	81424625153233.01972	38308.25493	153233.0197	0.82071516	5.09043331	0.9350445	G5 DISTRIBUTOR 1/4" DS6 ND6.5
5 Circuit x 8 Pass + 13 Circuit x 10 Pass	61453418162056.428	40514.107	162056.428	0.86800674	5.07753341	0.9571845	G5 DISTRIBUTOR 1/4" DS6 ND6.5
5 Circuit x 6 Pass + 14 Circuit x 10 Pass	61453419161577.53864	40394.38466	161577.5386	0.86541703	5.08496061	0.9564946	G5 DISTRIBUTOR 1/4" DS6 ND6.5
5 Circuit x 10 Pass + 15 Circuit x 8 Pass	61453420159820.16816	39955.04204	159820.1682	0.85600153	5.08639761	0.9557672	G5 DISTRIBUTOR 1/4" DS6 ND6.5
5 Circuit x 8 Pass + 16 Circuit x 6 Pass	61443421150011.78176	37502.94544	150011.7818	0.80349848	5.07961483	0.9509364	G5 DISTRIBUTOR 1/4" DS6 ND6.5
5 Circuit x 10 Pass + 20 Circuit x 6 Pass	61453425157529.172	39382.293	157529.172	0.84369368	5.09861194	0.9525641	G5 DISTRIBUTOR 1/4" DS6 ND6.5
5 Circuit x 4 Pass + 23 Circuit x 8 Pass	61463428161544.5968	40386.1492	161544.5968	0.86520313	5.09604179	0.9543813	G5 DISTRIBUTOR 1/4" DS6 ND6.5
6 Circuit x 2 Pass + 14 Circuit x 4 Pass	61423420111828.96342	37276.32114	111828.9634	0.59899914	5.09333033	0.9326026	G5 DISTRIBUTOR 1/4" DS6 ND6.5
8 Circuit x 2 Pass + 13 Circuit x 4 Pass	61423421110952.7353	36984.2451	110952.7353	0.59430386	5.09529126	0.9310324	G5 DISTRIBUTOR 1/4" DS6 ND6.5
8 Circuit x 4 Pass + 23 Circuit x 6 Pass	61453431153676.95192	38419.23798	153676.9519	0.82310032	5.08787453	0.9489261	G5 DISTRIBUTOR 1/4" DS6 ND6.5
9 Circuit x 4 Pass + 16 Circuit x 2 Pass	61423425107739.43629	35913.14543	107739.4363	0.57709139	5.09929753	0.9249231	G2 DISTRIBUTOR-1/4" DS10-01
9 Circuit x 8 Pass + 22 Circuit x 6 Pass	61463431160543.53444	40135.88361	160543.5344	0.85986018	5.0908567	0.952939	G5 DISTRIBUTOR 1/4" DS6 ND6.5
9 Circuit x 6 Pass + 24 Circuit x 2 Pass	61433433126138.69498	42046.23166	126138.695	0.67561132	5.09895472	0.931602	G2 DISTRIBUTOR-1/4" DS12-03
10 Circuit x 6 Pass + 21 Circuit x 2 Pass	61433431127299.92892	42433.30964	127299.9289	0.68184478	5.09199537	0.9336648	G2 DISTRIBUTOR-1/4" DS12-03
17 Circuit x 2 Pass + 17 Circuit x 4 Pass	61433434125601.47541	41867.15847	125601.4754	0.67276403	5.08556378	0.930833	G2 DISTRIBUTOR-1/4" DS12-03

FIGURE 1- PART OF THE RESULTING DATASHEET SHOWING THE CALCULATED OUTPUTS IN GREEN

- ***Phase 2 : Alarko FCU Thermal Test Simulation:***

In this phase of the project, the main purpose is to perform thermal calculations on Alarko-Carrier fan coil units and simulate their behavior under various Eurovent test conditions. These results will be helpful in validating the performance of our fan coils for Eurovent certification. For example, if a unit under test deviates from expected values, we can compare its performance to Python theoretical values to identify potential production issues within the water coils. In cases where testing at our factory is impractical due to high lab fault rates, the unit is sent to TSE Lab, which can be costly. In such instances, Python theoretical results can directly facilitate Eurovent certification declaration.

The code finds the required outputs for 540 different cases in under 10 minutes, the outputs include mass flowrate, water pressure drop, Total thermal capacity, latent capacity, sensible capacity and mass flowrate of condensate. These are calculated while considering various factors such as water and air properties, coil geometry, and operating conditions. A visualized plots representing the water pressure drop, total heat capacity, latent capacity and sensible capacity against the water mass flow rate are also represented.

The code accompanied by comments for clarification is shown below.

```
# Calculating water pressure drop and thermal capaciies of Alarko fan coil units.

#Import fluids library.
import iapws
#Import colebrook library to use Darcy friction factor function.
import colebrook
import CoolProp.CoolProp as CP
# Import pyplot for plotting
from matplotlib.pyplot import text
import matplotlib.pyplot as plt
# Import pandas,numpy and math to read data and operate from excelsheet.
import pandas as pd
import numpy as np
import math

import psychrolib
psychrolib.SetUnitSystem(psychrolib.SI)

# get rid of default='warn'
pd.options.mode.chained_assignment = None

# define the path to FCU Coil Database .csv file
df = pd.read_csv(r"C:\Users\*****\Desktop\Desktop\Alarko Projects\WaterPD_FCU\AllCoilsInput.csv")

# Define constant properties of water and copper at Eurovent standard testing conditions :
g = 9.81 #Gravitational energy [m/s2]
roughness = 15E-7 #Roughness of new copper [m] (value should be between 13E-7 & 15E-7)
dia = 6.4/1000 #Inner diameter of the coil [m]
Outer_dia = 7/1000 #Outer diameter of the tube. [m]
Area = (math.pi)*((dia/2)**2) #Internal area of the coil [m2]
eD = roughness/dia #mu/D describes the roughness of the pipe material compared to its inner diameter, get
f from moody chart [-]
Rb = 0.015 #The hairpin bend radius in [m]
Kb = 0.3 #Bend loss coefficient for 2 Rb/D (obtained from a graph)
fin_height = 157E-5 # fin height [m]
fin_thickness = 12E-5 # fin thickness [m]
fin_perM = 590.5511 # fins per Meter [-] (eqaul to 15 FPI)
FinnedTubeArea = 0.03183 #[m2/m]
Copper_K = 398 # Conductivity of copper [W/m.K]
Aluminuim_K = 237 # Conductivity of Aluminium [W/m.K]
# Define a list of FCU sizes
FCU_Sizes = [2, 3, 4, 5, 6, 7, 8, 10, 12, 14]
# Define modes (Cooling and Heating)
Modes = ["Cooling St.","Cooling Non St. 1","Cooling Non St. 2","Cooling Non St. 3", "Heating St.",
"Heating Non St."]
# Define row types (3, 4, 3+1)
Row_Types = ["3", "4", "3+1"]

# Define empty columns in the .csv file:

df['Mass Flow Rate']= ''
df['Pressure Drop'] = ''
df['Total Capacity_Air'] = ''
df['Total Capacity_Water'] = ''
df['Sensible Capacity'] = ''
df['Latent Capacity'] = ''
df['Calculated Air Off Temp'] = ''

# Loop through all coils in FCUCoilInfo excel sheet to and calcualte pressure drops:
for i in range(540):

    # Total length of the coil [m].
    Total_L = df['Length'][i]*df['RowNumber'][i]*10
    # Average length of the coil [m].
    avg_L = Total_L/df['CircuitNumber'][i]
    # Tube Bends in a Circuit.
```

```
    Bends = (10*df['RowNumber'][i] - df['CircuitNumber'][i]) / df['CircuitNumber'][i]

    Coil_Area = df['Height'][i] * df['Length'][i]

    # Define temperature [Kelvin] and pressure [kPa] at FCU inlet and outlet:
    Inlet_Temp_K = df['WaterInletTemp'][i] + 273.15
    Outlet_Temp_K = df['WaterOutletTemp'][i] + 273.15

    avg_Temp_K = (Inlet_Temp_K+Outlet_Temp_K)/2

    # Calculate water properties:

    # Water density [kg/m^3]
    density = iapws.IAPWS97(T=avg_Temp_K, P=0.101325).rho
    # Water dynamic viscosity in [kg/m.s] [Pa.s]
    dynamic_viscosity = iapws.IAPWS97(T=avg_Temp_K, P=0.101325).mu
    # Water thermal conductivity in [W/m.k]
    Water_K = iapws.IAPWS97(T=avg_Temp_K, P=0.101325).k
    # Water inlet enthalpy [kJ/kg]
    inlet_enthalpy = iapws.IAPWS97(T=Inlet_Temp_K, P=0.101325).h
    # Water outlet enthalpy [kJ/kg]
    outlet_enthalpy = iapws.IAPWS97(T=Outlet_Temp_K, P=0.101325).h
    # Water latent heat of vaporization [kJ/kg]
    heat_of_vaporization = 2500.4
    # Water inlet specific capacity [kJ/kg.K]
    inlet_Cp = iapws.IAPWS97(T=Inlet_Temp_K, P=0.101325).cp
    # Water outlet specific capacity [kJ/kg.K]
    outlet_Cp = iapws.IAPWS97(T=Outlet_Temp_K, P=0.101325).cp
    # Water Prandtl number [-]
    Prandtl = (dynamic_viscosity*((inlet_Cp+outlet_Cp)*0.5*1000))/Water_K

    # Calculate air properties:

    # Air inlet specific capacity [J/kg.K]
    InletAir_Cp = CP.PropsSI("C", "T", (df['AirDryBulbTemp'][i]+273.15), "P", 0.101325, "Air")
    # Air inlet density [kg/m3]
    InletAir_Density = CP.PropsSI("D", "T", (df['AirDryBulbTemp'][i]+273.15), "P", 101325, "Air")
    # Air relative humidity at FCU inlet [%]
    InletAir_RH = psychrolib.GetRelHumFromTWetBulb(df['AirDryBulbTemp'][i], df['AirWetBulbTemp'][i],
101325)
    # Dry air specific volume at FCU inlet [m3/kg]
    DryAir_SpecificVolume = 1 / (psychrolib.GetDryAirDensity(df['AirDryBulbTemp'][i], 101325))

    # ********************** #
    # Iterate through mdot here
    # ********************** #

    mdotGuess = 0.01  # Starting value
    min_difference = 999

    while mdotGuess <= 1.005:

        # Calculate QTotal_Water
        QTotal_Water_Initial = mdotGuess * ((inlet_Cp + outlet_Cp) / 2) * (Outlet_Temp_K - Inlet_Temp_K) *
1000
        deltaTAir = QTotal_Water_Initial/(InletAir_Cp * df['AirFlowRate'][i] * InletAir_Density / 3600)
        OutletAirTempGuess = df['AirDryBulbTemp'][i] - deltaTAir

        # Calculate the absolute difference between OutletAir_Temperature and the target
        if (df['Mode'][i] == "Heating St.") or (df['Mode'][i] == "Heating Non St."):
            abs_difference = abs(OutletAirTempGuess - df['AirOffTemp'][i]*1.05)
        else:
            abs_difference = abs(OutletAirTempGuess - df['AirOffTemp'][i]*0.8)
            # the 0.8 value should be variable based on the unit size, the smaller the unit, the closer
the value to 1.0

        if abs_difference < min_difference:
            # If the current absolute difference is smaller than the smallest so far, update the values
            min_difference = abs_difference
```

```
            mdot = mdotGuess
            OutletAir_Temperature = OutletAirTempGuess
            QTotal_Water = QTotal_Water_Initial

        # Increase mdot by a small increment
        mdotGuess += 0.005

    # ********************** #
    # Air side thermal capacity
    # ********************** #

    QTotal_Air = (df['AirFlowRate'][i] * InletAir_Density / 3600) * InletAir_Cp * (OutletAir_Temperature -
df['AirDryBulbTemp'][i])

    # ***************************** #
    # Water Pressure Drop Calculation
    # ***************************** #

    # water mass flow rate through 1 circuit [kg/s]
    mdotperC = mdot/df['CircuitNumber'][i]

    # fluid velocity inside the pipes [m/s].
    velocity = mdotperC/(Area*density)
    # Reynolds number of the fluid to describe type of the flow as laminar or turbulent [-]
    Re = density*velocity*dia/dynamic_viscosity
    # Darcy friction factor, used to determine friction loss in the coil (from moody chart, depends on Re
and mu/D) [-]
    factor = colebrook.sjFriction(Re, eD)

    #Pressure drop due to length of the pipes [Kpa]
    PD_Length = avg_L * factor*density*(velocity**2)/(2*dia)
    # Pressure drop due to gravitational height [Kpa]
    PD_Height = density*g*df['Height'][i]
    # Pressure drop due to fittings/Bends (Hiarpins) [Kpa]
    PD_Hairpins = ((factor*density*(velocity**2)*(math.pi) *
Rb/(2*dia))+((Kb/2)*density*(velocity**2)))*Bends
    # Total pressure drop due to all of the above (sum) [kPa]
    Pressure_Drop = (PD_Length + PD_Height + PD_Hairpins)/1000

    # ***************************** #
    # Latent and Sensible Heat Transfer
    # ***************************** #

    if (df['Mode'][i] == "Heating St.") or (df['Mode'][i] == "Heating Non St."):
        Dehumidification_Capacity = 0
        Sensible_Capacity = 0
    else:
        # Water mass flow rate condensed by the unit [kg/s]
        Condensed_mdot = df['AirFlowRate'][i] / (DryAir_SpecificVolume*(0.98 - InletAir_RH)*3600)
        # Dehumidification capacity is the latent capacity [W], using condensed mdot in [g/s]
        Dehumidification_Capacity = heat_of_vaporization * Condensed_mdot
        Sensible_Capacity = QTotal_Water - Dehumidification_Capacity

    # ********************** #
    # Print Results to database
    # ********************** #

    df['Pressure Drop'][i] = Pressure_Drop
    df['Mass Flow Rate'][i] = mdot
    df['Total Capacity_Water'][i] = QTotal_Water
    df['Total Capacity_Air'][i] = QTotal_Air
    df['Sensible Capacity'][i] = Sensible_Capacity
    df['Latent Capacity'][i] = Dehumidification_Capacity
    df['Calculated Air Off Temp'][i] = OutletAir_Temperature

df.to_csv(r"C:\Users\*****\Desktop\Desktop\Alarko Projects\WaterPD_FCU\AllTestResults.csv", index=False)
```

A brief explanation of the key components and what the code is doing:

- It reads data from an Excel file containing information about FCU coils (e.g., dimensions, coil and fin geometry, test condition).
- Constants related to water, fin material, tube material properties, as well as FCU sizes, operating modes, and row types, are defined.
- The code enters a loop that iterates through each FCU coil's data, and for each coil, it performs the following calculations:
    - Calculates various water and air properties (density, dynamic viscosity, thermal conductivity, enthalpy, etc.).
    - Iterates over different values of water mass flow rates to find that resulting to the closest outlet air temperature to the target. The code uses an iterative approach to minimize the temperature difference between the calculated and the target value.
    - Calculates the air-side thermal capacity based on the previously calculated mass flow rate and properties.
    - Calculates the pressure drop in the water side of the coil, considering factors like length of the coil, bends (fittings) and pressure drop due to gravitational energy.
    - Calculates water mass flow rate condensed by the unit, latent and sensible heat capacities.
- After processing all the FCU coils, the code saves and exports the results to a CSV file for further analysis or reporting.

An initial version of the code was developed to generate plots showing the mass flow rate ranging from 0.0 to 1.0 liter per second vs. water side pressure drop for each FCU type at different Eurovent test conditions, the code also calculates and prints the second order polynomial constants for each unit type (pressure drop as a function of mass flowrate). The plot down below is shown as an example for FCU 06 (3 rows, 4 rows and 3+1 rows) under Non-Standard heating test condition, the 2-nd order polynomial constants are presented on the right side of the plot.

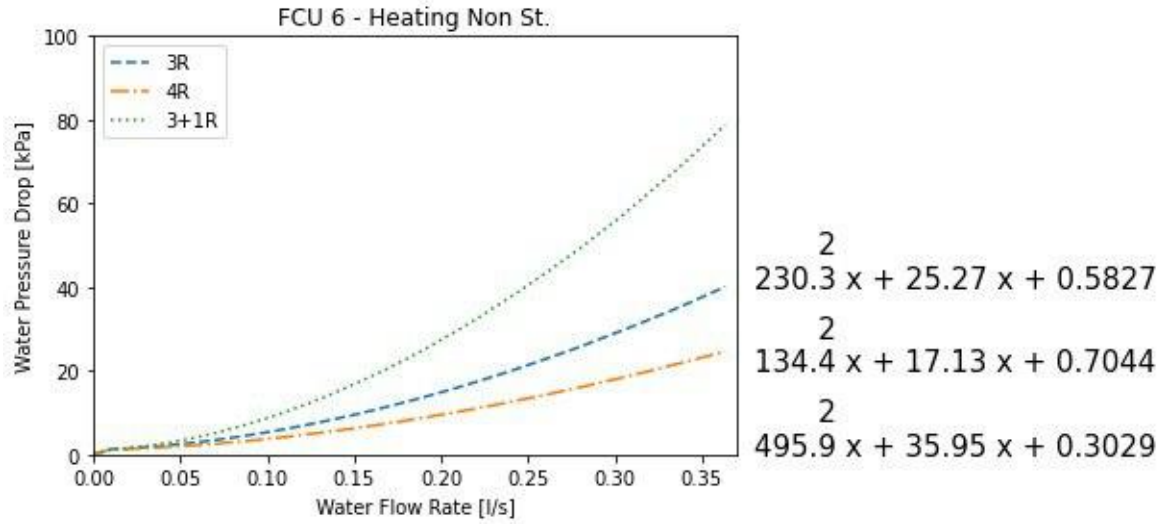

FIGURE 2-FCU 06 (3R, 4R, 3+1R) PRESSURE DROP VS. MASSFLOWRATE WITH 2-ND ORDER POLYNOMIAL CONSTANTS

Benefit or Savings Provided:

Using model-based approaches to solve R&D projects while integrating programming languages promises several benefits and savings. The extensive libraries provided can significantly reduce the time required for thermal calculations and simulations, enabling faster product development. Furthermore, mathematical libraries such as ACHP enhance the accuracy and confidence of our calculations, resulting in fewer errors, reduced rework, and eventually less labor demands.

While traditional calculation methods remain essential, they lack scalability. Therefore, adapting to the programming languages can help in accommodating the growing complexity of our HVAC system designs.

At Alarko, we currently rely on third-party technical software's, such as WinCoil and FCUProSelector, resulting to annual costs deducted from the R&D budget. However, Alarko-Carrier have the engineering expertise and capabilities necessary to develop customized solutions for our customers' requirements, eliminating the need for costly outsourced option.

Developing an in-house software with dynamic database structures can offer significant advantages in terms of time and cost efficiency. Currently, we schedule weekly meetings with third-party software developers to explain our customer requirements. This process can be quite time-consuming, as it requires extensive communication and creation of detailed reports to convey every aspect of the desired database structure and process. Instead, we could allocate this time towards building our own dynamic databases and performing the necessary calculations.

Furthermore, our FCU laboratory system and software were developed by an external company. Currently, we are facing several issues with our lab setup. There is a lack of PID control improvements, along with equipment sizing, selection and positioning problems. As a result, our testing process has become challenging, time consuming, and, in some cases, impossible, especially for non-standard heating tests on relatively large units. Some FCUs require an entire day or more for testing due to faults in the software's PID logic and control. These issues are causing significant time, cost, and labor inefficiencies. Implementing a logical PID system control or model-based approach that can simulate FCU performance within minutes could reduce these problems and lead to notable savings.

Evaluation & Conclusion:

The DX Coil code was used in calculating the capacities and choosing suitable distributors for the AHU DX Coil project, DX Coils and their distributors will be later implemented in Airovision Builder selection software, which is part of "39HQ Optimization Phase 2" project that aims to decrease 10% of 39HQ AHU price. On the other hand, the FCU code can be used to perform theoretical thermal tests to help in attaining Eurovent yearly certification process by declaring the correct test values.

When a unit under test deviates from expected values, we compare its performance to simulated theoretical values to identify potential production issues. In cases where factory testing is impractical due to high lab faults or cost constraints, simulation results can also assist in this regard.

In conclusion, the implementation of a model-based component matching system and the integration of programming languages have already enhanced Alarko's R&D capabilities. This approach proves to be efficient, and cost-effective.

The integration of an open-sourced programing language such as Python, if adapted, can possibly eliminate the need for costly third-party technical software, offering substantial long-term cost savings while providing greater control and flexibility in R&D initiatives.

With faster and more accurate simulations, Alarko can accelerate product development, respond effectively to market demands, and maintain its competition in the market. Additionally, as calculations become more automated and dependable, labor demands are expected to decrease, allowing the R&D team to focus on creative and complex aspects of their work.