Tutor

Dr. Carles Fité Piquer
Departament d'Enginyeria Química i Química Analítica

Grau en Enginyeria Química

Treball Final de Grau

Software to design a cooling tower

Marc Terés Labrid

June 2022

UNIVERSITAT DE BARCELONA

CONTENTS

SUMMARY

Cooling tower is a unit operation widely used in the chemistry industry. It is used to cool water that has lost its utility as a refrigerant because it has been used in condensers and heat exchangers. After cooling, water can be reused or send back to the environment without thermal damage.

Although cooling towers can generate large steam plumes, which causes many people to associate this steam with smoke and pollution, it is a clean operation that takes advantage of the principle of cooling by water evaporation.

In this project, a tool is created to facilitate the design for this type of operation. That is why an executable application has been made, using the python programming language. The program calculates the height of a counterflow water cooling tower for a given set of parameters: water and air inlet conditions, water outlet conditions and individual transfer coefficients. The solution is obtained by solving the mass and heat balances and differential transfer equations through the gas and liquid film along the cooling tower.

This tool is though for a user that can be specialized or not in cooling towers, in order that the user can get design parameters easily and quickly.

Keyboards: Cooling tower, dehumidification, software design, python.

RESUM

La torre de refrigeració és una operació unitària altament utilitzada a la indústria química per refredar aigua que ha estat utilitzada en condensadors o intercanviadors de calor, i ha perdut la seva capacitat de refrigeració. Així aquesta pot ser reutilitzada o tornada al medi sense generar danys tèrmics.

Tot i que les torres de refrigeració poden generar grans columnes de vapor, fet que fa que molta gent relacioni aquest vapor amb fum i contaminació, és una operació neta que aprofita el principi de refredament per evaporació.

En aquest projecte es crea una eina per facilitar el disseny per aquest tipus d'operació. Així doncs, s'ha generat un aplicació executable, a partir de d'un software que utilitza el llenguatge de programació python. El programa calcula l'altura d'una torre de refrigeració d'aigua en contracorrent donats els següents paràmetres: condicions d'entrada de l'aire i l'aigua, condicions de sortida de l'aigua i coeficients de transferència individuals. La solució s'obté resolent els balanços de massa i energia, i les equacions diferencials de transferència a través de la pel·lícula de gas i líquid al llarg de la torre.

Aquesta eina està pensada perquè l'usuari, tant si està especialitzat o no amb les torres de refrigeració, pugui obtenir paràmetres de disseny amb facilitat i rapidesa.

Keyboards: Torre de refrigeració, deshumidificació, disseny de software, python.

1. INTRODUCTION

In the chemistry industry, there are lots of processes that generates big amounts of heat. In order to dissipate this heat, water is usually used as a refrigerant in heat exchangers and condensers. The resulting water is hot and loses its utility as a refrigerant. Cooling towers are used to reduce the temperature of this water, in order that water can be reused or send back to the environment without thermal damage.

Cooling towers are based on the principle of evaporation. When a warm liquid is brought in contact with an unsaturated gas, a little portion of the liquid is vaporized.[1] The energy to vaporize the liquid comes from the liquid so it is cooled.

A cooling tower is a gas-liquid contact unit operation. This gas and liquid are usually air and water. The heat-transfer process involves latent heat due the vaporization of a small portion of the water and sensitive heat due the difference of temperature between air and water. Most of the heat transfer is due the latent heat.[2]

Gas-liquid contactors have more applications than cooling. A cooling tower is also a humidifier because the liquid is cooling while the gas is humidifying. Gas-liquid contactors can also be used as dehumidifiers. When we brought in contact a warm saturated gas with a cold liquid, part of the vapor condenses, and the humidity of the gas decreases. Then, if it is reheated, the gas has the same temperature but less humidity than before. This dehumidifying process entails an increase of the temperature of the liquid. Dehumidification of air is a common unit operation in many processes.

There are several types of cooling towers, with many sizes and models of each type. Depending in the relative directions of the water there are counterflow and crossflow towers. In counterflow operation, the air goes vertically upward against the downward fall of the water. In crossflow the air goes perpendicularly through the downward fall of the water.

In a general classification based on the air draft, there is natural and mechanical draft. In natural draft the air flow is produced by the density difference between the heated air inside the tower and the cool air of the ambient. The hot air is less dense than the cool air, that's why the hot air with the water vapor goes upwards the tower making the characteristic steam column that it is seen in this kind of tower. In Figure 1 natural draft towers are shown. The first one operates in counterflow and the second in crossflow.

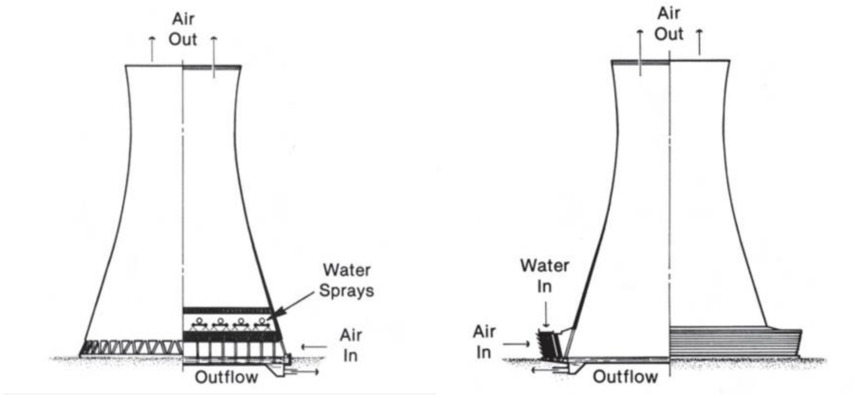


Figure 1. Natural draft towers.[3]

In mechanical draft towers the air flow is produced by one or more fans. Depending on the fan location, these towers are categorized as forced draft, if the fan is in the ambient air entering, or induced draft, if the fan is in the air discharge. In Figure 2, a forced draft tower is shown.

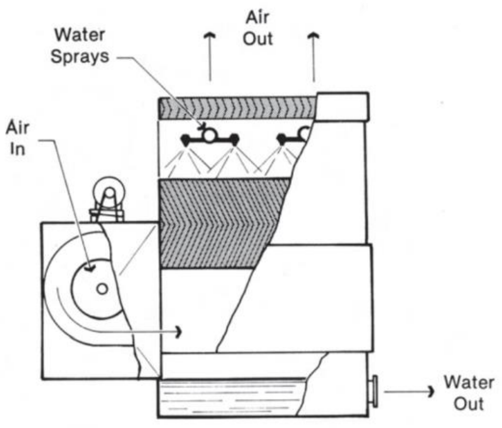


Figure 2. Counterflow forced draft tower.[3]

Cooling towers usually use nozzles, water sprays or other devices to distribute the water across the section of the tower. Drift eliminators are usually used to reduce the escape of water droplets entrained in the discharge air stream. Cooling towers often have a packing material. The propose of the packing is to allow a higher contact between air and water, because a large surface area for evaporative cooling is provided, to enhance the tower efficiency. Several types of packing can be used. Figure 3 shows V-shaped bars packing, which is used in counterflow towers.

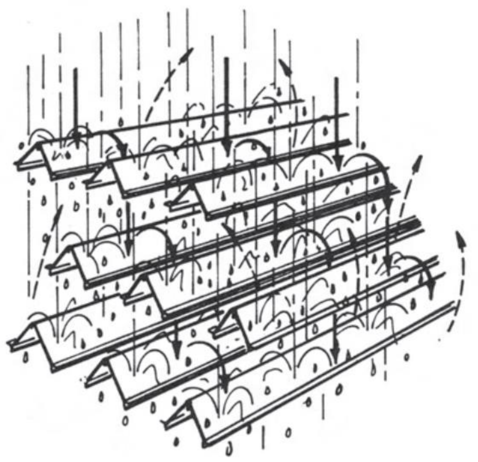


Figure 3. Typical splash-type fill.[3]

1.1. PSYCHOMETRICS OF EVAPORATION

Some general knowledge of psychometrics is necessary to understand cooling towers performance. Psychometry is the relation between physic and thermodynamic properties of a gas mixture composed by an incondensable gas and a condensable vapor. This mixture can be in contact with the liquid of the vapour compound. In most of cooling towers, the gas is air, and the liquid is water. Some basic definition and equations of psychometrics will be explained. The following equations show the psychometrics of air-water system. In case that the cooling tower uses another mixture, some equations may not be valid.

Dry temperature (T): Temperature indicated directly by a thermometer exposed to the air.

Absolut humidity (H): Vapor mass (A) carried by dry gas (B).

Humid heat (c_s): Heat energy necessary to increase 1ºC the temperature of 1 kg of the gas plus the contained vapor.[1] The specific heat of air is 1.005 kJ/kg·ºC and the specific heat of water vapor is 1.884 kJ/kg ºC.[4]

$$c_s = c_{PB} + c_{PA} \cdot H \quad [1]$$

$$c_s \left[{}^{kJ}/_{kg\ ºC}\right] = 1.005 + 1.884 \cdot H \quad [2]$$

Humid enthalpy (i_G): Enthalpy of a unit mass of gas plus whatever vapor it may contain. The total enthalpy is the sum of the sensible heat of the vapor, the latent heat of the liquid in the gas, and the sensible heat of the vapor-free gas.[1] The reference temperature (T*) chosen for both components is 0 ºC. The latent heat of water at 0 ºC (λ*) is 2502 kJ/kg.[4]

$$i_G = c_s \cdot (T - T^*) + \lambda^* \cdot H \quad [3]$$

$$i_G \left[{}^{kJ}/_{kg}\right] = c_s \cdot T[ºC] + 2502 \cdot H \quad [4]$$

Saturated enthalpy of wet air (i_{Gs}): The enthalpy of the gas plus the vapor when the equilibrium between phases is stablished.[1] This i_{Gs} is tabulated. These are the empiric equations found fitting the tabulated data in some ranges.[4]

For a temperature between -15 ºC and 20 ºC:

$$i_{Gs} \left[{}^{kJ}/_{kg}\right] = \frac{9.36 + 1.613 \cdot T[ºC]}{1 - 0.01265 \cdot T[ºC] + 6.0 \cdot 10^{-5} \cdot T[ºC]^2} \quad [5]$$

For a temperature between 10 ºC and 60 ºC:

$$i_{Gs} \left[{}^{kJ}/_{kg}\right] = \frac{10.42 + 1.37 \cdot T[ºC]}{1 - 0.019 \cdot T[ºC] + 9.5 \cdot 10^{-5} \cdot T[ºC]^2} \quad [6]$$

For a temperature between 50 ºC and 90 ºC:

$$i_{Gs} \left[{}^{kJ}/_{kg}\right] = \frac{2.28 \cdot T[ºC] - 16.9}{1 - 0.015862 \cdot T[ºC] + 5.88 \cdot 10^{-5} \cdot T[ºC]^2} \quad [7]$$

1.2. COUNTERFLOW GAS-LIQUID CONTACT EQUIPMENT

The design a counterflow gas-liquid contact equipment is based in the scheme of Figure 4. Gas-liquid contact operations are dehumidifiers and humidifiers like cooling towers, that is why the design is valid for all these kinds of operation, as long as these operations follow the scheme. The gas-liquid contactor operates in counterflow. The bottom of the tower is considered as zone 1 and the top of the tower is considered zone 2. All flows and properties of the top or bottom of the tower have the subindex of its zone. The gas (G) enters to the bottom at a temperature T_{G1} and with a humidity H_1 and leaves to the top at a temperature T_{G2} and with a humidity H_2. The liquid (L) enters to the top at a temperature T_{L2} and leaves to the bottom at a temperature T_{L1}. The section (S) of the tower is constant. The height (z) of the tower includes only the range where gas and liquid are in contact.

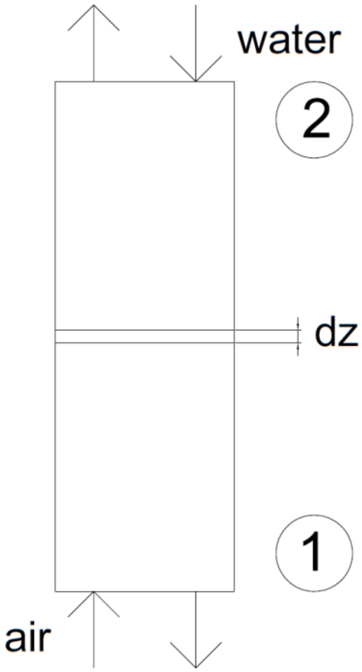


Figure 4. Scheme of a counterflow gas-liquid contact equipment.

1.3. OPERATION LINE

The mass balance for the scheme in Figure 4 is: $dL = G' \cdot dH$

The enthalpy balance for the scheme in Figure 4 is: $G' \cdot di_G = L \cdot \ c_L \cdot dT_L$

The operation line is obtained integrating the enthalpy and mass balance from the bottom of the tower to a certain point of the tower. The operation line is Equation 8. The flow of liquid (L) is considered constant despite some of the liquid evaporates or condensates in the process, because the variation is negligible. The gas flow without vapor (G') and the calorific capacity (c_L) are constants. Because these parameters are constant the result of the integral is a line. The operation line is the relation of the total enthalpy of the gas at any point of the tower (i_G) with the temperature of the liquid at any point of the tower (T_L), considering at the bottom of the tower that the total enthalpy of the gas is i_{G1} and the temperature of the liquid is T_{L1}.

$$i_G = i_{G1} + \frac{L \cdot c_L}{G'}(T_L - T_{L1}) \qquad [8]$$

The operation line goes from the zone 1 (bottom of the tower) to the zone 2 (top of the tower). Two different operation lines are represented in Figure 5. The operation line cannot cross the saturation curve because is not possible to pass from one side of the equilibrium to the other. All operation lines below the saturation curve are operations of cooling. We can see at this kind of operation that T_{L1} is lower than T_{L2}, that means that the temperature of liquid at the exit tower has decreased (zone 1 and the exit of liquid are in the bottom of the tower), while the enthalpy of the wet gas has increased, because it has been humidified.

Otherwise, all operation lines above the saturation curve are dehumidification operations. In that case, the temperature of the liquid has increased, and the enthalpy of the wet gas has decreased due the loss of humidity and temperature.

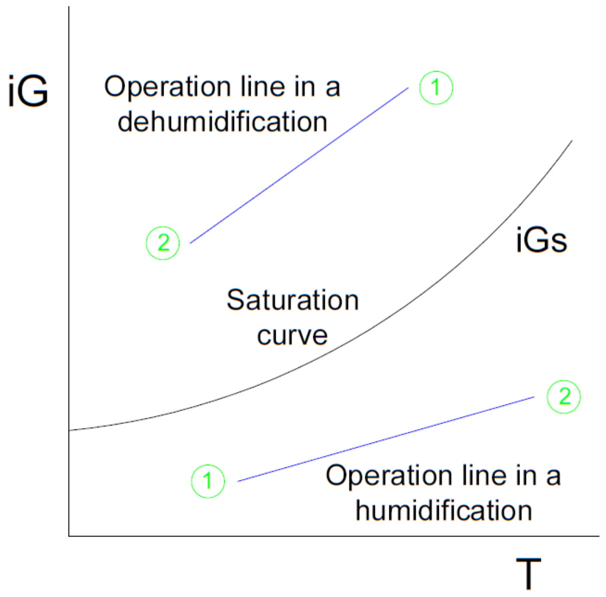

iG
Operation line in a dehumidification
1
2
iGs
Saturation curve
2
1
Operation line in a humidification
T

Figure 5. Operation lines of cooling (humidification) and dehumidification towers.

1.4. MASS AND HEAT TRANSFER THROUGH THE GAS OR LIQUID FILM

From mass balance, enthalpy balance and heat flows balance the following differential equations result.[4] In these equations the heat-transfer area per contact volume is represented as a. The subindex i means the parameter is in the interface. The interface is in the equilibrium of saturated gas.

Equation 9 relates variation of z with the variation of humidity. This relation is obtained by the mass transfer in the gas film, so it is related with the mass-transfer coefficient from gas to interface (k_Y).

$$dz = \frac{G'/S}{k_Y \cdot a} \frac{dH}{H_i - H} \qquad [9]$$

In this case, Equation 10 relates variation of z with the variation of liquid temperature. This relation is obtained by the heat transfer in the liquid film, so it is related with the heat-transfer coefficient from liquid to interphase (h_L).

$$dz = \frac{\left(L/_S\right) \cdot c_L}{h_L \cdot a} \frac{dT_L}{T_L - T_i} \qquad [10]$$

In Equation 11, variation of z and variation of gas temperature are related. The relation is obtained by the heat transfer in the gas film, so it is related with the heat-transfer coefficient from gas to interface (h_G).

$$dz = \frac{\left(G'/_S\right) \cdot c_s}{h_G \cdot a} \frac{dT_G}{T_i - T_G} \qquad [11]$$

Finally, Equation 12 relates z variation with total enthalpy of the gas variation. This equation is only valid for air-water systems because the relation $h_G = k_Y \cdot c_s$ has been used in the balances and it is a relation only valid for the air-water system.

$$dz = \frac{G'/_S}{k_Y \cdot a} \frac{di_G}{i_{Gi} - i_G} \qquad [12]$$

1.5. **TIE LINE**

The tie line is the relation between the conditions in the interface and the films. It comes from the union of Equations 10, 12 and the mass and enthalpy balance. Because Equation 12 is used, the equation is only valid for air-water systems. The resultant Equation 13 is a line with a slope of $-h_L/k_Y$.

$$\frac{i_G - i_{Gi}}{T_L - T_i} = \frac{-h_L \cdot a}{k_Y \cdot a} \qquad [13]$$

The tie line is shown in red at Figure 6. This line slope limits are minus infinite and zero. When the transfer in the gas film is the only controlling step, the h_L coefficient tends to infinity. That is the reason why slope is minus infinite, therefore the tie line is vertical as shown in the representation as case A. On the other hand, if the transfer in the liquid film is the only controlling step, the k_Y coefficient tends to infinity. In that case, the slope of the tie line is zero to give a horizontal line, as shown in case B. When the transfer control is in both films, the slope of the tie line is between these limits, as case C.

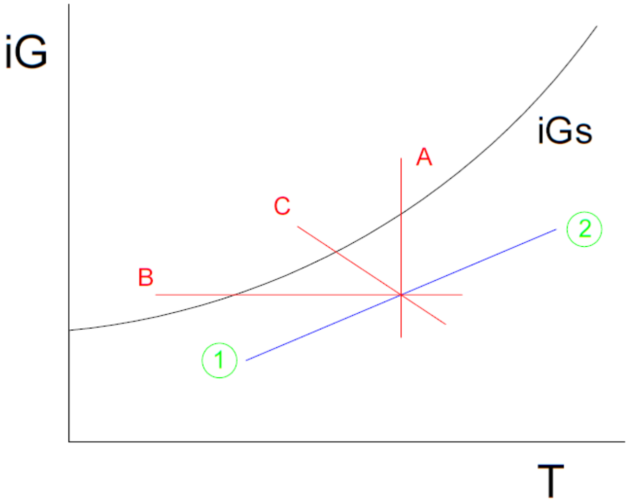


Figure 6. Representation of tie line.

1.6. CALCULATION OF THE TOWER HEIGHT

Usually, the objective to the design of a cooling tower is to determine the height of the tower if it is wanting to cool the water with a gas stream with a given temperature and humidity. The determination of that height (z) is made in different ways depending on the h_L and k_Y coefficients.

If the control is in the liquid film, z is determined by Equation 14. Equation 14 comes from the integration of Equation 10 from T_{L1} to T_{L2}. The tie line is $i_G = i_{Gi}$ because the slope is 0. The h_L coefficient is in the Equation 14 because is the controlling parameter in the transfer of heat in the liquid film.

$$z = \frac{\left({}^{L}/_{S}\right) \cdot c_L}{h_L \cdot a} \int_{T_{L1}}^{T_{L2}} \frac{dT_L}{T_L - T_i} \qquad [14]$$

In the other hand, if the control is in the gas film, z is determined by Equation 15. This equation comes from the integration of Equation 12 from i_{G1} to i_{G2}. In that case the tie line is vertical, that means that $T_L = T_i$. The k_Y coefficient is relevant because is the controlling parameter in the gas film.

$$z = \frac{G'/S}{k_Y \cdot a} \int_{i_{G1}}^{i_{G2}} \frac{di_G}{i_{Gi} - i_G} \qquad [15]$$

If the transfer control is in both films, the tie line is Equation 13. Equation 14 and 15 are both valid to determine z and will have the same solution. Equation 14 and 15 are only vali for air-water system.

1.7. CONDITIONS FOR A FEASIBLE COOLING PROCESS

Not always a cooling (humidifier) or dehumidifier operation is possible. To be possible some condition must be met. First one is that must be thermodynamically possible. That means that for a cooling (humidification) the wet temperature of the air must be below the temperature of the liquid at the tower bottom. Another way to see it is that the total enthalpy of the gas in entering of the tower (i_{G1}) must be below the saturation curve.

In contrast, for dehumidification is the opposite. The wet temperature of the inlet air must be above the temperature of the liquid at the tower bottom, and the total enthalpy of the inlet air in (i_{G1}) must be above the saturation curve.

The second condition is that the relation L/G' must be below a maximum slope. The operation line slope is defined by $(L \cdot c_L)/G'$. The operation line cannot cross the saturation curve. That is why there is a maximum L/G' ratio that cannot be exceeded in order that the operation line does not cross the saturation line. That means that if a given water flow is fed in the tower, there is a minimum air flow (G'_{min}) that necessarily needs to be surpassed, in order the operation to be feasible. Or if the air flow is given, there is a maximum water flow (L_{max}) that cannot be surpassed in order the operation is possible.

In order to find this maximum slope for humidification, there are two possibilities due the convex form of the saturation line. The first one is the possibility in Figure 7 (situation a). Where

T_{L2} has the enthalpy of the equilibrium. But if using this method we cross the saturated curve, the maximum slope is the tangential between the operation line and the saturated curve shown in Figure 8 (situation b). One easy way to see if it is situation a or b is looking at the slopes. If the slope of the operation line is steeper than the slope of the saturation curve at zone 2, it is situation a. Otherwise, it is situation b.

For dehumidification there is only one possibility due the form of the saturation line, and this is situation shown in Figure 9.

If L_{max} or G'_{min} are used, the height of the tower is infinite, that's why the liquid flow should be below L_{max} or the air flow should be above G'_{min} (not equal).

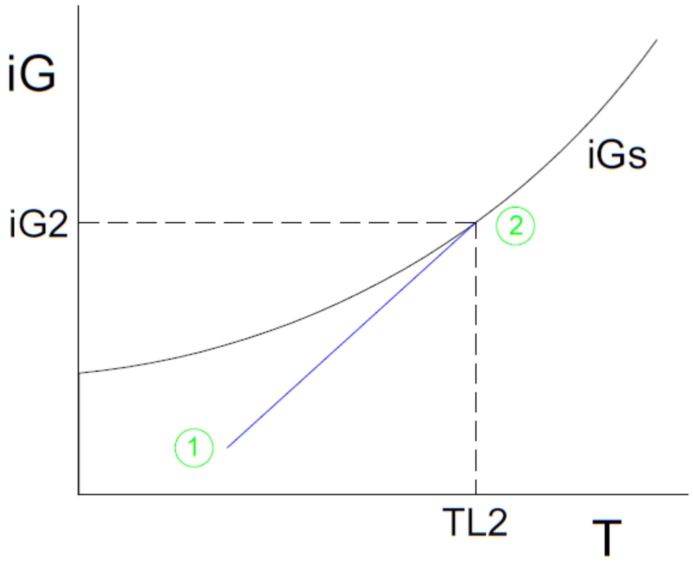


Figure 7. Representation of the maximum slope for a cooling tower in situation a.

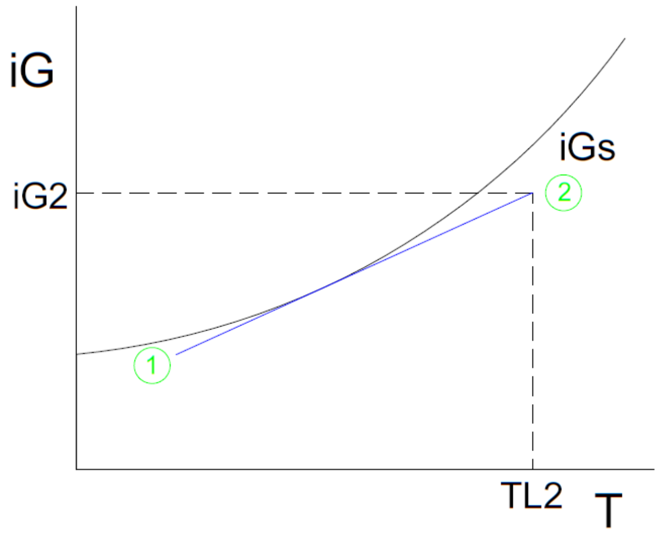


Figure 8. Representation of the maximum slope for a cooling tower in situation b.

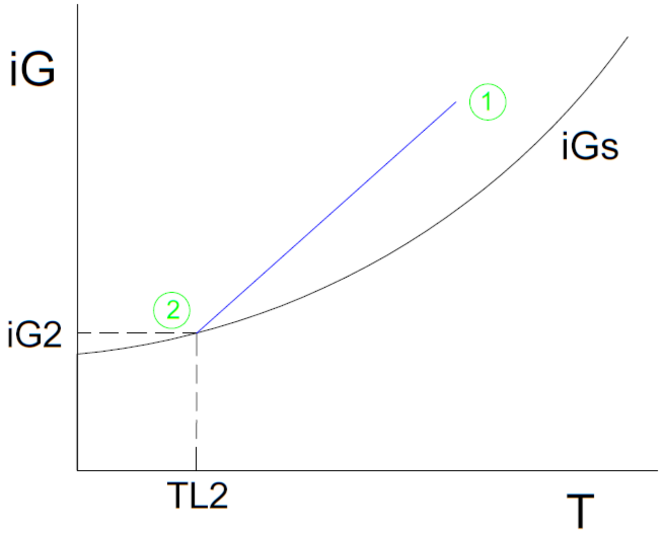


Figure 9. Representation of the maximum slope for a dehumidifier.

1.8. TOWER PROFILES

Tower profiles show the variation of some properties at different heights of the tower. Some interesting properties to determine are the liquid temperature, the gas temperature, and the gas humidity.

One way to determine the temperature of the liquid at any tower height is solving the differential Equation 10. If the tie line is horizontal, this equation cannot be integrated because T_i equals T_L, and, since the mass transfer coefficient in the gas film tends to infinity, originates an indeterminate form. Instead, Equation 12 should be solved to obtain the variation of the total enthalpy of the humid gas (i_G). The operation line relates i_G with the liquid temperature (T_L), with that the temperature of the liquid can be determined.

The temperature and humidity of the gas can be determined at any height of the tower by solving simultaneously Equations 9 and 11. If the transfer control is not in gas film (the tie line is horizontal), the temperature and the humidity of the gas become the values in the interface.

2. OBJECTIVES

The aim of the project is to create a python software that calculates the height of a counterflow cooling tower when some parameters are introduced. The software must generate and application easy and friendly to be used, where the parameters are introduced in a window by the user. In addition to the calculation of the height of the tower, the program should show other information, like the operation line, and water temperature, air temperature and humidity along the tower. It also should warn when the operation is not possible, and it should tell the user why it is not possible.

The software is thought to calculate the height (z) of a counterflow humidifier (cooling tower) or dehumidifier as the pattern shown in Figure 4. The parameters that must be introduced are the common parameters that are used in this unit operation design. These parameters are:

- Temperature of inlet water at the top of the tower (T_{L2}).
- Temperature of outlet water at the bottom of the tower (T_{L1}).
- Temperature of inlet air at the bottom of the tower (T_{G1}).
- Humidity of inlet air at the bottom of the tower (H_1).
- Water flow velocity (L/S).
- Air flow velocity in dry basis (G'/S).
- Volumetric mass-transfer coefficient in the gas film (k_Ya).
- Volumetric heat-transfer coefficient in the liquid film (h_La).

3. METHODOLOGY

In this part of the project is shown how the software works. Not all parts of the software are shown, there are only explained the basics of the code in order to understand how it operates. The full software is the Appendix 1.

3.1. USER INTERFACE

To make easier the use of the program, a graphical user interface has been created by means of the package tkinter, imported as tk. Initially a main window is created with the function tk.Tk. The parameters to be entered by the user are defined in the code as string variables. Then a table is created, where the left column (0) are labels of the parameter name and units, and the right column (1) is formed by text boxes where the user can introduce keyboard characters. The left column is created with the function tk.Label, and the right column is created with the function tk.Entry. After every tk.Label and tk.Entry function the suffix .grid is added to show the position in the table. Every row in the table correspond to a parameter.

Another tk.Entry is created in the next column to show the answer. In order the user cannot modify the box, this is in disabled state.

An image of the scheme of the cooling tower is added under the table with function Image. A white figure is also added to posteriorly display graphs.

Then two buttons are added with the function tk.Button. The first one is the “Clear all” button. This button realises the command clear. The command clear is function where every parameter is set to nothing. To do that, every variable in function clear has the suffix .set(“”). This command also clears the graph and the solution. The second is “Solve” button. This is situated next to the “Clear all” button. The “Solve” button has the command answer. The command answer is the one that do all the calculations.

Finally, the widget .mainloop is responsible to handle all events received by the application. There is a fragment of the code for a better understanding of the past explanation:

```
window = tk.Tk()
window.title("Water cooling or air dehumidification tower")
kYa_w = tk.StringVar()
[…]
tk.Label(window, text = "kYa [kg/(s·m^3)]", width=15).grid(row
= 7, column = 0)
[…]
tk.Entry(window, textvariable = kYa_w, justify="center",
width=15).grid(row = 7, column = 1)
[…]
z_w = tk.Entry(window, justify="center", width=15)
z_w.grid(row= 1 , column = 4)
z_w.configure(state="readonly")
png0 = PIL.Image.open("app_image.png")
pngr = png0.resize((500, 450), resample=4)
pngf = ImageTk.PhotoImage(pngr)
labelphoto = tk.Label(window, image = pngf).grid(row = 9,
column = 0, rowspan = 2, columnspan = 5)
fig1 = plt.figure(figsize=(7.5, 6.2))
canvas = FigureCanvasTkAgg(fig1, master=window)
canvas.draw()
canvas.get_tk_widget().grid(row=1, column=5, rowspan=9)
clear_but = tk.Button(window, text = "Clear all", font=(15), fg
= "white", bg = "red", padx = 35, pady = 5, command =
clear).grid(row = 3, column = 2, rowspan = 3, columnspan = 3)
solve_but = tk.Button(window, text = "Solve", font=(15), fg =
"white", bg = "green", padx = 46, pady = 5, command =
answer).grid(row = 6, column = 2, rowspan = 3, columnspan = 3)

window.mainloop()
```

3.2. PARAMETER INTRODUCTION AND CHECKING

When the “Solve” button is pressed, the software does all the calculation. But the parameters introduced by the user need to be introduced and check in the function answer. The function answer starts proving if the text introduced can be transformed into a float. To do that, a function called is_valid_fload has been made. This function tries to transform a string into a float, if it can return a True, but if it cannot return a False.

If a parameter cannot pass that function as a True, the software generates a warning message box. Otherwise the parameter is introduced with the suffix .get().

Some other checks are required. For instance: the air water cannot be supersaturated; if it is a humidification i_{G1} must be below the saturation point; etc. These are on consecutive sequence of “if” and “elif”. Each of it has his own warning message box. In the else the software continues, that means that if any of these checks happened, a message box informs on which parameter is wrong and stops.

3.3. OPERATION LINE INTERSECT THE SATURATION CURVE

If the operation line crosses the saturation curve, the process is impossible. To see if it happens, the function fsolve from scipy.optimize is used to solve an equation. What is wanted to know is which temperature the saturation curve (Equations 5, 6 and 7) is equal to the operation line (Equation 8). This solver is the next code:

```
TLintersection, infodict, ier, msg = fsolve(lambda TL: iGi(TL)
- iG(TL, param1), x0=TL1, full_output=True)
```

$T_{Lintersection}$ is the temperature where both functions cross. The parameter ier is 1 when it found a solution, and it is another number when there is no solution. So, to know if there is an intersection ier must be 1 and $T_{Lintersection}$ must be between T_{L1} and T_{L2}.

If there is an intersection, the operation is impossible. However, the maximum water flow (L_{max}) and the minimum air flow (G'_{min}) can be calculated to help the user to find a possible operation.

In a cooling tower, T_{L2} is higher than T_{L1}. In a humidification there are two possibilities. In situation a (Figure 7) the maximum slope is found with the enthalpy of the saturated curve at T_{L2}. But if the slope of the saturation curve at T_{L2} is higher than the slope of the operation line in situation a, occurs situation b (Figure 8). In situation b, the maximum slope is found when the saturation curve slope is equal to the slope of the operation line when its contact. So, a fsolve need to be realised with that condition. The function pendiGi(TL_) calculates the slope of the saturation line at any TL_.

For a dehumidification T_{L2} is lower than T_{L1}. In that situation (Figure 9) the maximum slope is the same as situation a of the previous case. Finally, knowing that the maximum slope is $L/(G'\cdot c_L)$, L_{max} and G'_{min} can be determine. There is a fragment of the code to for a better understanding:

```
if ier == 1 and (TL1 - TLintersection)*(TL2 - TLintersection) < 0:
      if TL2 > TL1:
            pendromax = (iGi(TL2) - iG1) / (TL2 - TL1)
      if pendromax < pendiGi(TL2):
            TLpendiguals = fsolve(lambda TL: pendiGi(TL) - (iGi(TL) - iG1)/(TL - TL1), TL2)[0]
            pendromax = (iGi(TLpendiguals) - iG1) / (TLpendiguals - TL1)
      else:
            pendromax = (iG1 - iGi(TL2)) / (TL1 - TL2)
```

After that, a message box generates a warning that tell the user G'_{min} and L_{max}. The code generates a graphic that shows how the operation line cross the saturation curve. To generate the graphic matplotlib.pyplot is used.

3.4. TOWER HEIGHT CALCULATION

If there is none of the previous problems, the tower height (z) can finally be calculated. To calculate this, there are 3 situations (Figure 6). For cases A and C, the equation to solve is Equation 15. On the other hand, the Equation 14 is solved for case B. Both equations got an integral. To solve these integrals function quad from scipy.integrate is used.

The way to solve these equations for the different cases is similar. This is the code for case C:

```
def subintd(iG_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    pend = -hLa / kYa
    TLsub = TL(iG_, param)
    Ti = fsolve(lambda T: (iG_ - iGi(T)) - pend*(TLsub - T),
    TLsub)[0]
    return 1 / (iGi(Ti) - iG_)
def z_d(param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    iG2 = iG(TL2, param)
    NUT = quad(subintd, iG1, iG2, args = param)[0]
    return (G / kYa)*NUT
param1 = [kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal]
z = z_d(param1)
```

The tie line in case C is described by equation 13. For case A the tie line is vertical, in that case T_i is equal to T_L. For case B the tie line is horizontal, so i_{Gi} is equal to i_G.

3.5. OPERATION LINE GRAPH

The operation line graph plots the saturation curve yeq vs xeq, the operation line yro vs xro and 5 tie lines yru1 vs xru1, yru2 vs xru2, etc. The saturation curve is obtained by Equations 5, 6 and 7; the operation line by Equation 8; and the tie lines by Equation 13.

The code of this graphic is shown because is the most complex. Other graphs have been done in a similar way.

```
fig1, ax = plt.subplots(2, 2, figsize = (7.5, 6), sharey =
False)
if pend1 == -float("inf"):
     xru1 = TL1, TL1
     yru1 = iG(TL1, param1), iGi(TL1)
[…]
elif pend1 == 0:
     Ti1 = fsolve(lambda T: iG(TL1, param1) - iGi(T), TL1)[0]
     xru1 = TL1, Ti1
     yru1 = iG(TL1, param1), iGi(Ti1)
[…]
else:
     Ti1 = fsolve(lambda T: (iG1 - iGi(T)) - pend1*(TL1 - T),
     TL1)[0]
     xru1 = TL1, Ti1
     yru1 = iG(TL1, param1), iGi(Ti1)
[…]
if pend1 == -float("inf"):
     xeq = np.linspace(TL1, TL2, 1000)
else:
     xeq = np.linspace(Ti1, Ti5, 1000)
yeq = iGi(xeq)
xro = np.linspace(TL1, TL2, 1000)
yro = iG(xro, param1)
ax[0, 0].plot(xeq, yeq, color = "black", label = "Saturation
curve")
ax[0, 0].plot(xro, yro, color = "blue", label = "Operation
line")
ax[0, 0].plot(xru1, yru1, color = "red", label = "Tie line")
[…]
leg = ax[0, 0].legend(loc="best", shadow=False, ncol=1,
fontsize = "small")
ax[0, 0].set_title("Operation line", fontsize = "large")
ax[0, 0].set_xlabel("Temperature [ºC]", fontsize = "medium")
ax[0, 0].set_ylabel("iG [kJ/kg]", fontsize = "medium")
[…]
plt.tight_layout()
canvas = FigureCanvasTkAgg(fig1, master=window)
canvas.draw()
canvas.get_tk_widget().grid(row=1, column=5, rowspan=9)
```

In that fragment of the code ax[0, 0] determine the row and column position of the graph, because 4 graphs are displayed in the same figure. The final part displays the 4 graphs in the tkinter main window.

3.6. LIQUID TEMPERATURE PROFILE

The water temperature along the tower is calculated in a similar way of z calculation. Equation 15 is solved for case A and C (Figure 6). However, this time integrates from i_{G1} to i_{Gx}, where i_{Gx} are a thousand values equally spaced from i_{G1} to i_{G2}. Function np.linspace creates the thousand equally spaced values. Function z_v_g calculates the integral from i_{G1} to i_{Gx} for case A, and z_d_g calculates for case C. Finally, the water temperature is calculated with the operation line (Equation 8).

```
y_iG = np.linspace(iG1, iG(TL2, param1), 1000)
x_z = z_v_g(y_iG, param1)
y_TL = TL(y_iG, param1)
```

Equation 14 is solved for case C in the same way as cases A and C, with the difference that the function z_h_g calculates the integral from T_{L1} to T_{Lx}.

The plot y_TL vs x_z is represented as the water temperature profile. The graphs is represented in ax[1, 0].

3.7. HUMIDITY AND GAS TEMPERATURE PROFILES

The humidity and gas temperature along the tower is obtained by the simultaneously resolution of differential Equations 9 and 11. To solve these differential equations simultaneously odeint function is used.[6]

First the temperature in the interface is obtained by the tie line (Equation 13). Then Equations 9 and 11 are solved by odeint. Equations 9 and 11 cannot be solved for case B (Figure 6), because the transfer in the liquid film is the only controlling step and k_Ya coefficient tends to infinity.

However, if the transfer in the liquid film is the only controlling step, the gas has no resistance, and it pass directly to the condition in the interface. That is why for a tie line slope of 0, the air temperature and humidity are from the air-water interface.

```
pend1 = -hLa / kYa
H_TG0 = [H1, TG1]
```

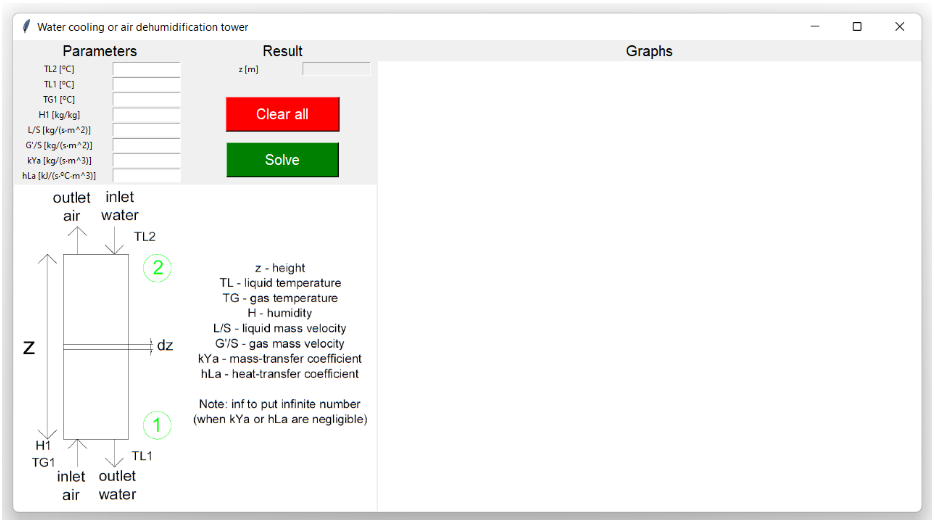
```
if pend1 == -float("inf"):
     Ti = y_TL
elif pend1 == 0:
     Ti = [fsolve(lambda T: iGi(T) - a, TL1)[0] for a in y_iG]
else:
     Ti = [fsolve(lambda T: (a - iGi(T)) - pend1*(b - T),
     TG1)[0] for a, b in zip(y_iG, y_TL)]
Hi = [fsolve(lambda H: iGi(a) - iG_H_TG(H, a), H1)[0] for a in
Ti]
if pend1 == 0:
     H = Hi
     TG = Ti
     x_z0 = 0, 0
     y_H1 = H1, H[0]
     y_TG1 = TG1, TG[0]
else:
     H = np.empty_like(x_z)
     TG = np.empty_like(x_z)
     H[0] = H_TG0[0]
     TG[0] = H_TG0[1]
     for i in range(1,1000):
          zspan = [x_z[i-1],x_z[i]]
          H_TG = odeint(perfils_H_TG, H_TG0, zspan,
          args=(Hi[i], Ti[i], param1))
          H[i] = H_TG[1][0]
          TG[i] = H_TG[1][1]
          H_TG0 = H_TG[1]
```

The plot of humidity and gas temperature are represented respectively in position ax[0, 1] and ax[1, 1].

4. RESULTS

The objective of this part is to see the results of the software. The python code has been converted into an executable application by pyinstaller. When it is executed, the main window appears (Figure 10). This window let the user click and write in any of the parameter textbox. The application got an image of the cooling tower scheme. This image is shown in Figure 11. The window also got a textbox to show the result, and a space to display the graphs.

When the Clear all button is pressed it delete all information in all textboxes and clear the space of the graphs. When the Solve button is pressed, depending on the data introduced, different things happen. This program is going to be tested in different situations.



z - height
TL - liquid temperature
TG - gas temperature
H - humidity
L/S - liquid mass velocity
G'/S - gas mass velocity
kYa - mass-transfer coefficient
hLa - heat-transfer coefficient

Note: inf to put infinite number
(when kYa or hLa are negligible)

Figure 10. Main window.

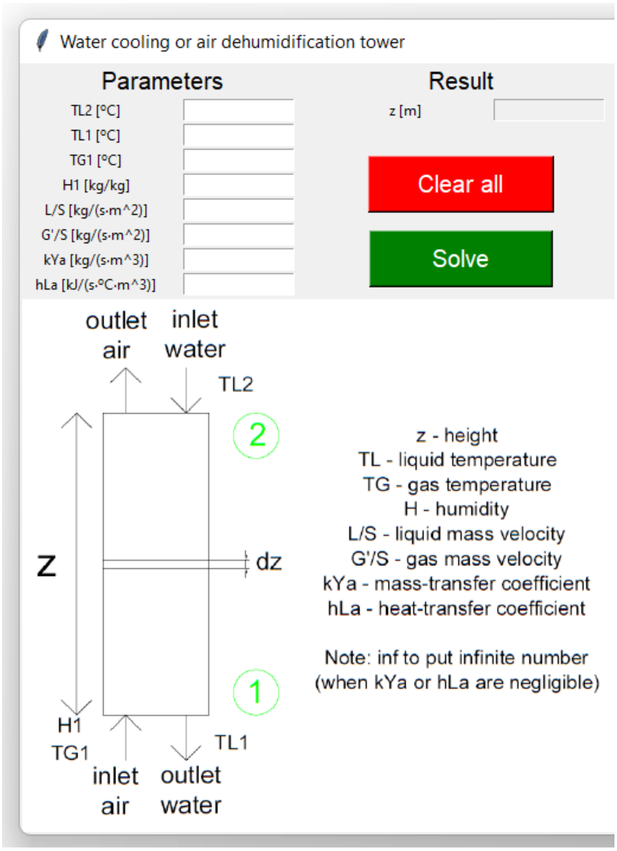


Figure 11. Enlargement of the left part of the principal window.

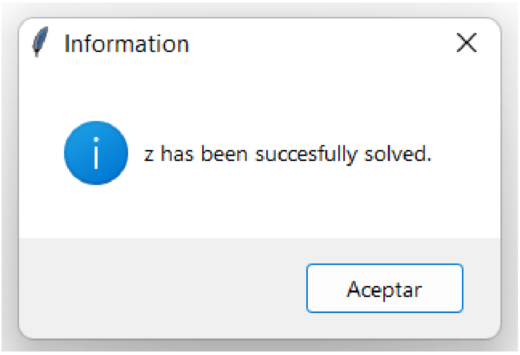
4.1. COOLING TOWER

The first situation to test is a water cooling (or air humidification). The parameters of the problem have been obtained from a list of problems[4]. The solution of that problem is a height of 9.35 m, and the parameters are these:

- T_{L2} = 55 ºC
- T_{L1} = 30 ºC
- T_{G1} = 30 ºC

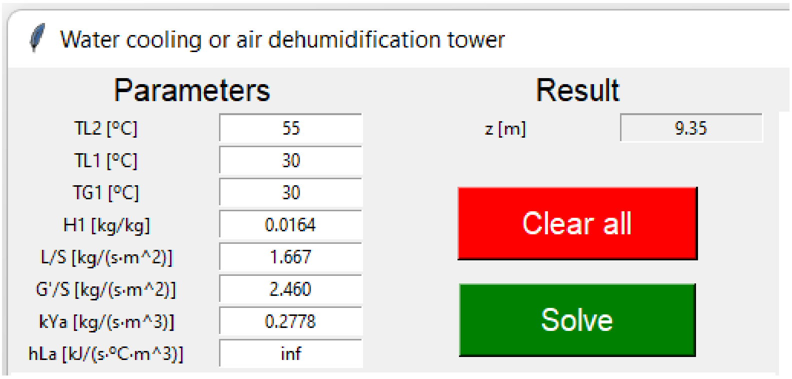
- H_1 = 0.0164 kg/kg
- L/S = 1.667 kg/(s·m2)
- G'/S = 2.460 kg/(s·m2)
- k_Ya = 0.2778 kg/(s·m3)
- h_La = inf (because the transference of heat in the liquid film is negligible)

After introducing the parameters and pressing solve the software generate window with a messagebox shown in Figure 12. In the main window at the textbox of z appears the result of 9.35, this is shown in Figure 13.


Information

z has been succesfully solved.

Figure 12. Messagebox when z has been solved.


Water cooling or air dehumidification tower

Parameters		Result	
TL2 [ºC]	55	z [m]	9.35
TL1 [ºC]	30		
TG1 [ºC]	30		
H1 [kg/kg]	0.0164		
L/S [kg/(s·m^2)]	1.667		
G'/S [kg/(s·m^2)]	2.460		
kYa [kg/(s·m^3)]	0.2778		
hLa [kJ/(s·ºC·m^3)]	inf		

Figure 13. Parameters and result for a cooling.

When the messagebox is closed some graphics appear in the graph space. These graphics are the operation line graph, and the humidity, gas temperature and liquid temperature profiles along the height of the tower. They are shown in Figure 14. The operation line is below the saturation curve because it is a cooling tower. The tie lines are vertical like case A from Figure 6.

These graphs show that the variation of humidity, air temperature and water temperature at the bottom of the tower is lower than in the top. This is because the operation line is closer to the saturation curve in the bottom. The tie line shows the separation from the operation to the saturation (equilibrium). So, if the tie line is longer, the driving force is bigger, and the variation of humidity and temperature increases.

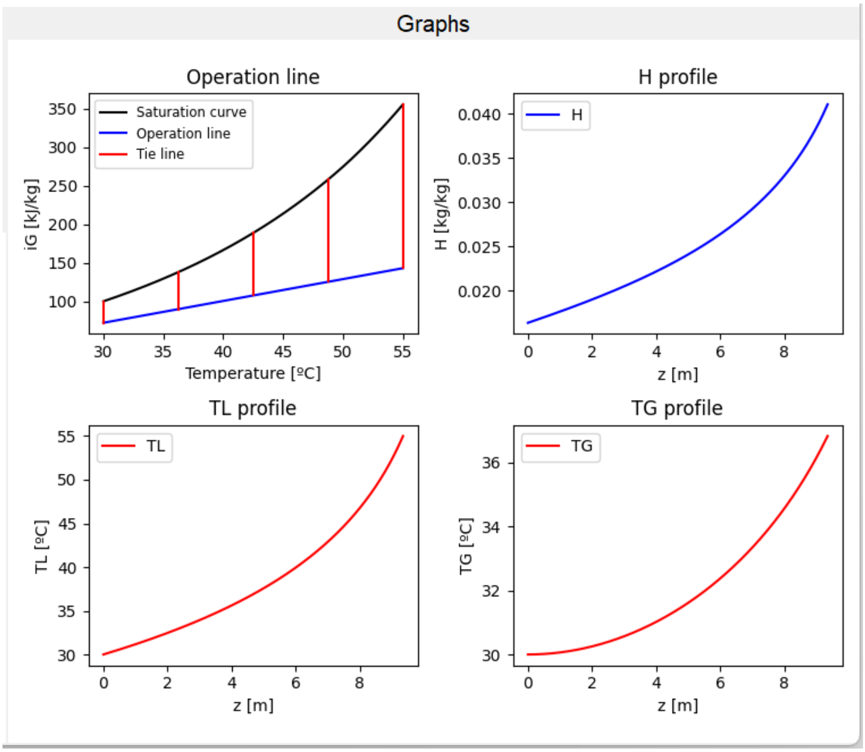


Figure 14. Graphs for a cooling.

For this case the program has calculated the height of the tower and shows some graphs. If some parameter is wanted to be changed, the user can press that parameter and change it, without having to introduce all parameters again. The result textbox is not changeable for the user. The user can press solve to calculate the new height and generate the new graphs. It also let the user delete all information with the button Clear all.

4.2. DEHUMIDIFICATION TOWER

In this case, a dehumidification is tested. The parameters are from a list of problems.[4] The problem wants to find the temperature of the outlet water for a tower of 3 meters with the next parameters:

- T_{L2} = 15 ºC
- T_{L1} = ?
- T_{G1} = 50 ºC
- H_1 = 0.037 kg/kg
- L/S = 1.323 kg/(s·m2)
- G'/S = 1.148 kg/(h·m2)
- k_Ya = 0.889 kg/(h·m3)
- h_La = 3 kJ/(s·ºC·m^3) (this parameter has been changed from the original problem in order to show the software works for tie lines from case C in figure 6).

To solve this problem, the user has to check different numbers in parameter T_{L1} until the result is a height of 3 m. After some attempts, T_{L1} value is 26.96 ºC, and with this T_{L1} the height of the dehumidification tower is 3.00 m. The graphs obtained are shown in Figure 15. Since it is a dehumidification tower, the operation line is above de saturation curve.

The resolution of this kind of problems shows that this software can also be used to determine other parameters for a certain height. To do it the user has to iterate different values. The fact that the software quickly calculates the height, and the easy way to change parameters, shows the utility of the software.

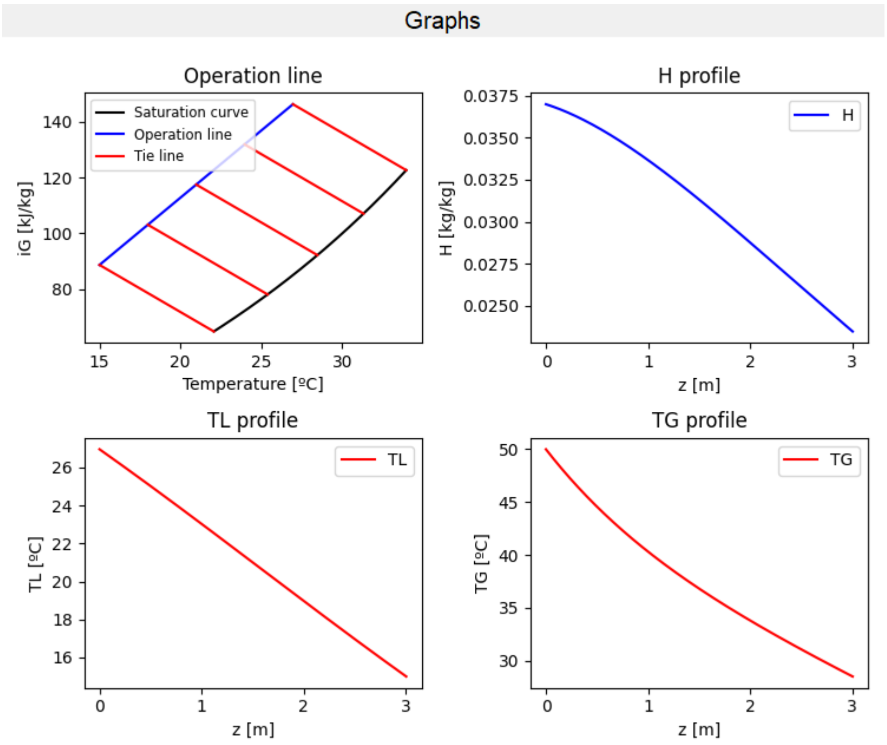


Figure 15. Graphs for a dehumidification.

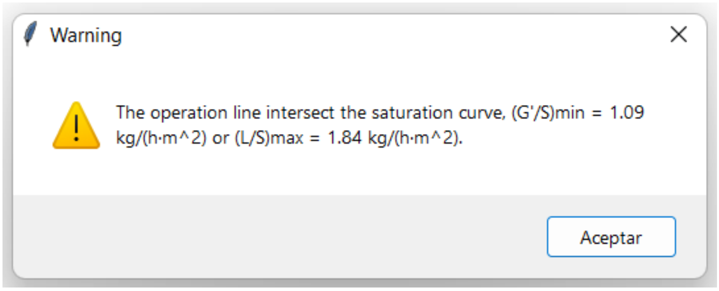
4.3. OPERATION LINE INTERSECTS THE SATURATION CURVE

When the operation line intersects the saturation curve, the operation is not possible. For the next parameters that happens:

- T_{L2} = 50 ºC
- T_{L1} = 25 ºC
- T_{G1} = 25 ºC
- H_1 = 0.01 kg/kg
- L/S = 2 kg/(s·m2)
- G'/S = 1 kg/(s·m2)

- k_Ya = inf (because the mass transfer resistance in the gas film is negligible)
- h_La = 1.5 kJ/(s·ºC·m^3)

After pressing the solver button, a warning messagebox appears (Figure 16). This messagebox contain the minimum air flow (G'_{min}) and the maximum water flow (L_{max}) for the operation to be possible. A graph of the intersection also appears (Figure 17). Textbox of z result is blank because the operation is not possible.



Warning

The operation line intersect the saturation curve, (G'/S)min = 1.09 kg/(h·m^2) or (L/S)max = 1.84 kg/(h·m^2).

Figure 16. Messagebox with $(G'/S)_{min}$ and $(L/S)_{max}$

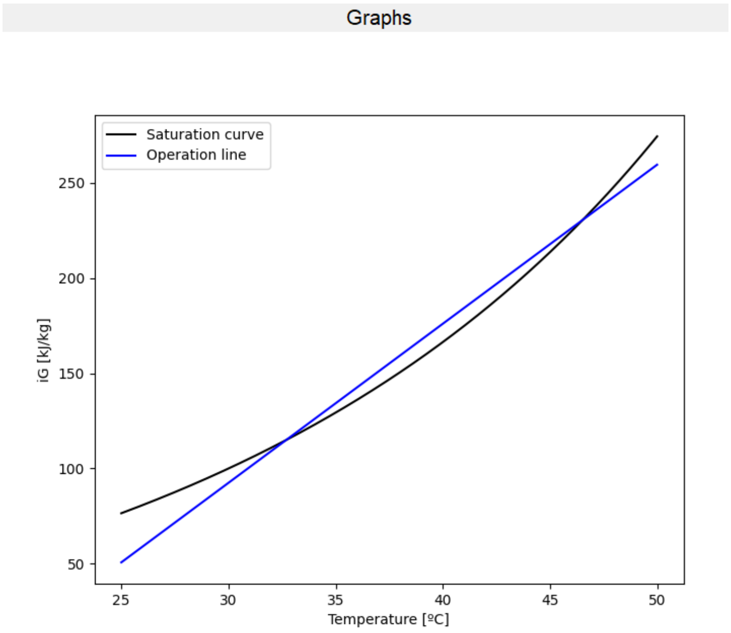


Figure 17. Graph of the intersection.

If the user change G'/S for $(G'/S)_{min}$ or L/S for $(L/S)_{max}$, the result for this case is the limit case shown in Figure 8, where the tower height tends to infinity. In that case, if it is replaced G'/S for 1.09 kg/h·m^2, the result obtained is a tower of 764 m, it is an extremely large number for a tower (is not infinite because the $(G'/S)_{min}$ is rounded). Otherwise, if a G'/S is quite bigger, for example 1.6 kg/(s·m^2), the result is 25 meters. It is still a big number because the operation is difficult, but it is much better. The graphs obtained for this operation are in Figure 18.

The operation is difficult because the operation line is close to the saturation curve. It can be seen in Figure 18 that the tie lines are short in comparison to the other operations.

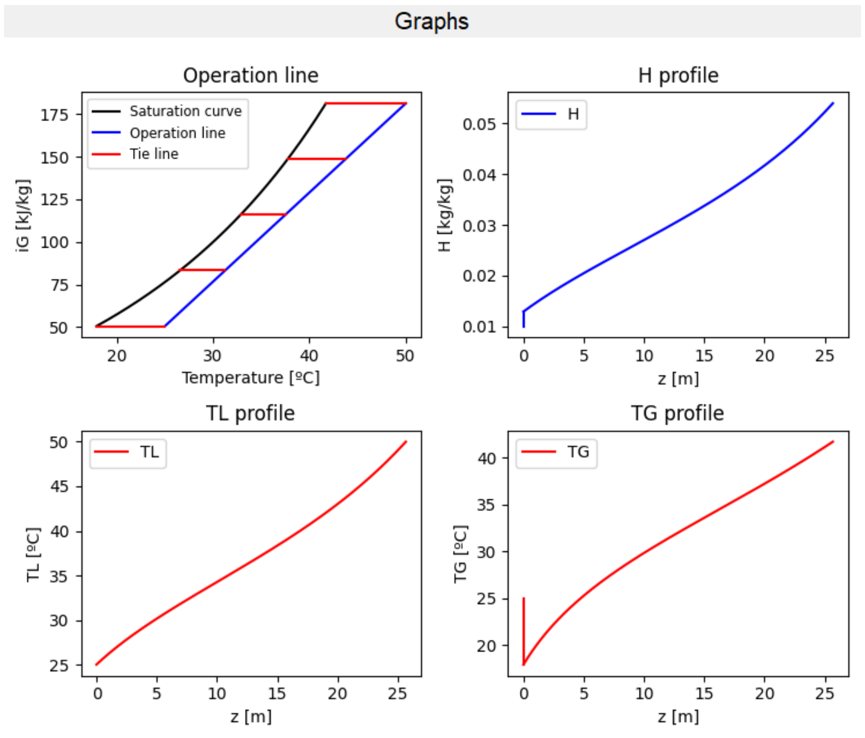


Figure 18. Graphs for a cooling.

These graphs of figure 18 are from a cooling operation where the tie lines are horizontal (case B of Figure 6). The jump of the gas temperature and humidity to the interface conditions can be appreciated.

5. CONCLUSIONS

The software has created an executable application for the design of cooling towers and dehumidification towers. The design is though for a tower that operates with the system air-water in countercurrent.

The software is useful because it calculates the height of the tower from most common parameters of design. It also generates additional information like the operation line and, water temperature, gas temperature and humidity profiles along the tower. It also let the user find other parameters when the height is known by iteration (trying values until the height is the one sought). It also helps the user by showing warnings, graphs and limit values, when the operation is impossible or there is something wrong.

The software is much faster and more accurate than the resolution made by ordinary handmade.

The application is easy and friendly to be used, because of its simplicity. It only has two buttons, one to solve, the other to clear all. The application has a little scheme to explain which are the parameters. It let the user change parameters with facility without having to introduce all the parameters again. It is also well visually pleasing and organized because every zone has his own purpose.

The software is compatible because .exe files can be used in many different devices without the requirement of the user to have installed python in his device.

The software does all its objectives, but it can continue developing with the addition of other utilities.

To conclude, python is an intuitive and powerful programming language, that can do innumerable things.

REFERENCES AND NOTES

1. Warren L. McCabe; Julian C. Smith; Peter Harriott. *Unit Operation of Chemical Engineering*. Fifth Edition. 1993. New York, McGraw-Hill
2. Robert H. Perry; Don W. Green. *Perry's Chemical Engineers' Handbook*. Seventh Edition. 2008. New York, McGraw-Hill.
3. John C. Hensley. *Cooling Tower Fundamentals*. Second Edition. 2009. Kansas, SPX Cooling Technologies, Inc.
4. C. Fité P. Psicometria I Operacions d'Humidificació. Class notes of Ampliació Operacions de Separació UB. 2021
5. SciPy v1.8.1 Manual --- scipy.optimize.fsolve https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html (accessed 12/06/2022)
6. Solve Differential Equations with ODEINT https://apmonitor.com/pdc/index.php/Main/SolveDifferentialEquations (accessed 12/06/2022)
7. Python python.org (accessed 12/06/2022)

ACRONYMS

T_{L2} - Temperature of inlet water at the top of the tower, ºC

T_{L1} - Temperature of outlet water at the bottom of the tower, ºC

T_{G1} - Temperature of inlet air at the bottom of the tower, ºC

H_1 - Humidity of inlet air at the bottom of the tower, kg/kg

L/S - Water flow velocity, $kg/(s \cdot m^2)$

G'/S - Air flow velocity in dry basis, $kg/(s \cdot m^2)$

$k_Y a$ - Volumetric mass-transfer coefficient in the gas film, $kg/(s \cdot m^3)$

$h_L a$ - Volumetric heat-transfer coefficient in the liquid film, $kJ/(s \cdot {}^{o}C \cdot m^3)$

i_G - Humid enthalpy, kJ/kg

z – height of the tower, m

APPENDICES

APPENDIX 1: FULL PYTHON CODE

```
import tkinter as tk
import PIL.Image
from PIL import ImageTk
from tkinter import messagebox
from scipy.integrate import quad
from scipy.integrate import odeint
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np

def is_valid_float(element: str) -> bool:
    try:
        float(element)
        return True
    except ValueError:
        return False

def cs(H_):
    return 1.005 + 1.884*H_

def iG_H_TG(H_, TG_):
    return cs(H_)*TG_ + 2502*H_

def iG(TL_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    return iG1 + (L*cL / G)*(TL_-TL1)

def TL(iG_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    return TL1 + ((iG_ - iG1) * G) / (L*cL)

def iGi_aux(Ti_):
    if Ti_ < 7.6562545:
        iGi = (9.36 + 1.613*Ti_) / (1 - 0.01265*Ti_ + 6.0e-
        5*Ti_**2)
    elif Ti_ > 50.390373:
        iGi = (2.28*Ti_ - 16.9) / (1 - 0.015862*Ti_ + 5.88e-
        5*Ti_**2)
    else:
        iGi = (10.42 + 1.37*Ti_) / (1 - 0.019*Ti_ + 9.5e-
        5*Ti_**2)
    return iGi

def iGi(Ti_):
    if type(Ti_) is list:
        iGi_ = [iGi_aux(a) for a in Ti_]
    elif type(Ti_) is np.ndarray:
        iGi_ = np.array([iGi_aux(a) for a in Ti_])
    else:
        iGi_ = iGi_aux(Ti_)
    return(iGi_)

def pendiGi(Ti_):
    h = 1e-6
    return (iGi(Ti_ + h) - iGi(Ti_ - h)) / (2*h)

def subintv(iG_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    Tsub = TL(iG_, param)
    return 1 / (iGi(Tsub) - iG_)

def subinth(TL_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    iGsub = iG(TL_, param)
    Ti = fsolve(lambda T: iGsub - iGi(T), TL_)[0]
    return 1 / (TL_ - Ti)

def subintd(iG_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    pend = -hLa / kYa
    TLsub = TL(iG_, param)
    Ti = fsolve(lambda T: (iG_ - iGi(T)) - pend*(TLsub - T),
    TLsub)[0]
    return 1 / (iGi(Ti) - iG_)

def z_v(param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    iG2 = iG(TL2, param)
    NUT = quad(subintv, iG1, iG2, args = param)[0]
    return (G / kYa)*NUT

def z_v_g_aux(iG_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    NUT = quad(subintv, iG1, iG_, args = param)[0]
    return (G / kYa)*NUT

def z_v_g(iG_, param):
    if type(iG_) is list:
        z_ = [z_v_g_aux(a, param) for a in iG_]
    elif type(iG_) is np.ndarray:
        z_ = np.array([z_v_g_aux(a, param) for a in iG_])
    else:
        z_ = z_v_g_aux(iG_, param)
    return(z_)

def z_h(param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    NUT = quad(subinth, TL1, TL2, args = param)[0]
    return (L*cL / hLa)*NUT
```

```
def z_h_g_aux(TL_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    NUT = quad(subinth, TL1, TL_, args = param)[0]
    return (L*cL / hLa)*NUT

def z_h_g(TL_, param):
    if type(TL_) is list:
        z_ = [z_h_g_aux(a, param) for a in TL_]
    elif type(TL_) is np.ndarray:
        z_ = np.array([z_h_g_aux(a, param) for a in TL_])
    else:
        z_ = z_h_g_aux(TL_, param)
    return(z_)

def z_d(param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    iG2 = iG(TL2, param)
    NUT = quad(subintd, iG1, iG2, args = param)[0]
    return (G / kYa)*NUT

def z_d_g_aux(iG_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    NUT = quad(subintd, iG1, iG_, args = param)[0]
    return (G / kYa)*NUT

def z_d_g(iG_, param):
    if type(iG_) is list:
        z_ = [z_d_g_aux(a, param) for a in iG_]
    elif type(iG_) is np.ndarray:
        z_ = np.array([z_d_g_aux(a, param) for a in iG_])
    else:
        z_ = z_d_g_aux(iG_, param)
    return(z_)

def perfils_H_TG(H_TG_, z_, Hi_, Ti_, param):
    kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal = param
    dHdz = (Hi_ - H_TG_[0])/(G/kYa)
    dTGdz = (Ti_- H_TG_[1])/(G/kYa)
    dHTGdz = [dHdz, dTGdz]
    return dHTGdz

def answer():
    z_w.configure(state="normal")
    z_w.delete(0, tk.END)
    z_w.configure(state="readonly")
    fig1 = plt.figure(figsize=(7.5, 6))
    canvas = FigureCanvasTkAgg(fig1, master=window)
    canvas.draw()
    canvas.get_tk_widget().grid(row=1, column=5, rowspan=9)
    plt.close()

    if is_valid_float(TL2_w.get()) == False:
        messagebox.showwarning(message="TL2  is  not  a  number",
        title="Warning")
    elif is_valid_float(TL1_w.get()) == False:
        messagebox.showwarning(message="TL1  is  not  a  number",
        title="Warning")
    elif is_valid_float(TG1_w.get()) == False:
        messagebox.showwarning(message="TG1  is  not  a  number",
        title="Warning")
    elif is_valid_float(H1_w.get()) == False:
        messagebox.showwarning(message="H1  is  not  a  number",
        title="Warning")
    elif is_valid_float(L_w.get()) == False:
        messagebox.showwarning(message="L/S  is  not  a  number",
        title="Warning")
    elif is_valid_float(G_w.get()) == False:
        messagebox.showwarning(message="G'/S  is  not  a  number",
        title="Warning")
    elif is_valid_float(kYa_w.get()) == False:
        messagebox.showwarning(message="kYa  is  not  a  number",
        title="Warning")
    elif is_valid_float(hLa_w.get()) == False:
        messagebox.showwarning(message="hLa  is  not  a  number",
        title="Warning")
    else:
        kYa = float(kYa_w.get())
        hLa = float(hLa_w.get())
        L = float(L_w.get())
        G = float(G_w.get())
        TL1 = float(TL1_w.get())
        TL2 = float(TL2_w.get())
        H1 = float(H1_w.get())
        TG1 = float(TG1_w.get())
        zreal = None

        if kYa/hLa > 100000:
            kYa = float("inf")
        if hLa/kYa > 100000:
            hLa = float("inf")
        cL = 4.18
        iG1 = iG_H_TG(H1, TG1)
        param1 = [kYa, hLa, cL, L, G, TL1, TL2, iG1, zreal]
        pend1 = -hLa / kYa

        if TL1 >= 70 or TL2 >= 70:
            messagebox.showwarning(message="Water   temperature
            must be below 70ºC.", title="Warning")
        elif TL1 <= 5 or TL2 <= 5:
            messagebox.showwarning(message="Water   temperature
            must be above 5ºC.", title="Warning")
        elif TL2 == TL1:
            messagebox.showwarning(message="TL1  and  TL2  can't
            be the same.", title="Warning")
        elif TG1 < -15:
            messagebox.showwarning(message="The  temperature  of
            air shouldn't be that low.", title="Warning")
        elif TG1 > 90:
            messagebox.showwarning(message="The temperature of
            air shouldn't be that high.", title="Warning")
        elif H1 < 0:
            messagebox.showwarning(message="Humidity can't be
            negative.", title="Warning")
        elif H1 > 1.42:
            messagebox.showwarning(message="Humidity must be
            lower.", title="Warning")
        elif L > 10:
            messagebox.showwarning(message="L/S must be
            lower.", title="Warning")
        elif L < 0.05:
            messagebox.showwarning(message="L/S must be
            higher.", title="Warning")
        elif G > 10:
            messagebox.showwarning(message="G'/S must be
            lower.", title="Warning")
        elif G < 0.05:
            messagebox.showwarning(message="G'/S must be
            higher.", title="Warning")
        elif kYa < 0.05:
            messagebox.showwarning(message="kYa must be
            higher.", title="Warning")
        elif hLa < 0.05:
            messagebox.showwarning(message="hLa must be
            higher.", title="Warning")
        elif iG1 > iGi(TG1):
            messagebox.showwarning(message="iG1 can't be above
            the saturation point.", title="Warning")
        elif TL2 > TL1 and iG1 >= iGi(TL1):
            warntext = "For this humidification iG1 is to high,
            iG1max = " + str(round(iGi(TL1), 2)) + " kJ/kg."
            messagebox.showwarning(message=warntext,
            title="Warning")
        elif TL2 < TL1 and iG1 <= iGi(TL1):
            warntext = "For this dehumidification iG1 is to
            low, iG1min = " + str(round(iGi(TL1), 2)) + "
            kJ/kg."
            messagebox.showwarning(message = warntext, title =
            "Warning")
        elif hLa == float("inf") and kYa == float("inf"):
            messagebox.showwarning(message="hLa  and  kYa  can't
            be both infinite.", title="Warning")
        else:
            TLintersection, infodict, ier, msg = fsolve(lambda
            TL:    iGi(TL)    -    iG(TL,    param1),    x0=TL1,
            full_output=True)

            if  ier  ==  1  and  (TL1  -  TLintersection)*(TL2  -
            TLintersection) < 0:

                if TL2 > TL1:
                    pendromax = (iGi(TL2) - iG1) / (TL2 - TL1)

                    if pendromax < pendiGi(TL2):
                        TLpendiguals    =    fsolve(lambda    TL:
                        pendiGi(TL)  -  (iGi(TL)  -  iG1)/(TL  -
                        TL1), TL2)[0]
                        pendromax = (iGi(TLpendiguals) - iG1) /
                        (TLpendiguals - TL1)

                else:
                    pendromax = (iG1 - iGi(TL2)) / (TL1 - TL2)

                Gmin = L*cL / pendromax
                Lmax = pendromax*G/cL
                warningtext = "The operation line intersect the
                saturation    curve,    (G'/s)min    =    "    +
                str(round(Gmin, 2)) + " kg/(h·m^2) or (L/S)max
                = " + str(round(Lmax, 2)) + " kg/(h·m^2)."
                messagebox.showwarning(message=warningtext,
                title="Warning")

                fig1, ax = plt.subplots(figsize = (7.5, 6))
                x = np.linspace(TL1, TL2, 1000)
                yeq = iGi(x)
                yro = iG(x, param1)
                ax.plot(x,    yeq,    color    =    "black",
                label="Saturation curve")
                ax.plot(x,  yro,  color  =  "blue",  label  =
                "Operation line")
                leg   =   ax.legend(loc="best",   shadow=False,
                ncol=1)
                plt.xlabel("Temperature [ºC]")
                plt.ylabel("iG [kJ/kg]")
                canvas = FigureCanvasTkAgg(fig1, master=window)
                canvas.draw()
                canvas.get_tk_widget().grid(row=1,    column=5,
                rowspan=9)
                plt.close()

            else:
                if pend1 == -float("inf"):
                    z = z_v(param1)

                elif pend1 == 0:
                    z = z_h(param1)

                else:
                    z = z_d(param1)

                z_w.configure(state="normal")
                z_w.insert(tk.END, round(z, 2))
                z_w.configure(state="readonly")
                messagebox.showinfo(message="z     has     been
                succesfully solved.", title="Information")

                fig1, ax = plt.subplots(2, 2, figsize = (7.5,
                6), sharey = False)

                if pend1 == -float("inf"):
                    xru1 = TL1, TL1
                    yru1 = iG(TL1, param1), iGi(TL1)
                    xru2 = TL1+(TL2-TL1)/4, TL1+(TL2-TL1)/4
                    yru2    =    iG(TL1+(TL2-TL1)/4,    param1),
                    iGi(TL1+(TL2-TL1)/4)
                    xru3 = TL1+2*(TL2-TL1)/4, TL1+2*(TL2-TL1)/4
                    yru3   =   iG(TL1+2*(TL2-TL1)/4,   param1),
                    iGi(TL1+2*(TL2-TL1)/4)
                    xru4 = TL1+3*(TL2-TL1)/4, TL1+3*(TL2-TL1)/4
                    yru4   =   iG(TL1+3*(TL2-TL1)/4,   param1),
                    iGi(TL1+3*(TL2-TL1)/4)
                    xru5 = TL2, TL2
                    yru5 = iG(TL2, param1), iGi(TL2)

                elif pend1 == 0:
                    Ti1  =  fsolve(lambda  T:  iG(TL1,  param1)  -
                    iGi(T), TL1)[0]
                    xru1 = TL1, Ti1
                    yru1 = iG(TL1, param1), iGi(Ti1)
                    Ti2  =  fsolve(lambda  T:  iG(TL1+(TL2-TL1)/4,
                    param1) - iGi(T), TL1+(TL2-TL1)/4)[0]
                    xru2 = TL1+(TL2-TL1)/4, Ti2
                    yru2    =    iG(TL1+(TL2-TL1)/4,    param1),
                    iGi(Ti2)
                    Ti3   =   fsolve(lambda   T:   iG(TL1+2*(TL2-
                    TL1)/4,   param1)   -   iGi(T),   TL1+2*(TL2-
                    TL1)/4)[0]
                    xru3 = TL1+2*(TL2-TL1)/4, Ti3
                    yru3   =   iG(TL1+2*(TL2-TL1)/4,   param1),
                    iGi(Ti3)
                    Ti4   =   fsolve(lambda   T:   iG(TL1+3*(TL2-
                    TL1)/4,   param1)   -   iGi(T),   TL1+3*(TL2-
                    TL1)/4)[0]
                    xru4 = TL1+3*(TL2-TL1)/4, Ti4
                    yru4   =   iG(TL1+3*(TL2-TL1)/4,   param1),
                    iGi(Ti4)
                    Ti5  =  fsolve(lambda  T:  iG(TL2,  param1)  -
                    iGi(T), TL2)[0]
                    xru5 = TL2, Ti5
                    yru5 = iG(TL2, param1), iGi(Ti5)

                else:
                    Ti1 = fsolve(lambda T: (iG1 - iGi(T)) -
                    pend1*(TL1 - T), TL1)[0]
                    xru1 = TL1, Ti1
                    yru1 = iG(TL1, param1), iGi(Ti1)
                    Ti2 = fsolve(lambda T: (iG(TL1+(TL2-TL1)/4,
                    param1) - iGi(T)) - pend1*(TL1+(TL2-TL1)/4
                    - T), TL1+(TL2-TL1)/4)[0]
                    xru2 = TL1+(TL2-TL1)/4, Ti2
                    yru2 = iG(TL1+(TL2-TL1)/4, param1),
                    iGi(Ti2)
                    Ti3 = fsolve(lambda T: (iG(TL1+2*(TL2-
                    TL1)/4, param1) - iGi(T)) -
                    pend1*(TL1+2*(TL2-TL1)/4 - T), TL1+2*(TL2-
                    TL1)/4)[0]
                    xru3 = TL1+2*(TL2-TL1)/4, Ti3
                    yru3 = iG(TL1+2*(TL2-TL1)/4, param1),
                    iGi(Ti3)
                    Ti4 = fsolve(lambda T: (iG(TL1+3*(TL2-
                    TL1)/4, param1) - iGi(T)) -
                    pend1*(TL1+3*(TL2-TL1)/4 - T), TL1+3*(TL2-
                    TL1)/4)[0]
                    xru4 = TL1+3*(TL2-TL1)/4, Ti4
                    yru4 = iG(TL1+3*(TL2-TL1)/4, param1),
                    iGi(Ti4)
                    Ti5 = fsolve(lambda T: (iG(TL2, param1) -
                    iGi(T)) - pend1*(TL2 - T), TL2)[0]
                    xru5 = TL2, Ti5
                    yru5 = iG(TL2, param1), iGi(Ti5)

                if pend1 == -float("inf"):
                    xeq = np.linspace(TL1, TL2, 1000)
                else:
                    xeq = np.linspace(Ti1, Ti5, 1000)
                yeq = iGi(xeq)
```

```
xro = np.linspace(TL1, TL2, 1000)
yro = iG(xro, param1)

ax[0, 0].plot(xeq, yeq, color = "black", 
label="Saturation curve")
ax[0, 0].plot(xro, yro, color = "blue", label = 
"Operation line")
ax[0, 0].plot(xru1, yru1, color = "red", label 
= "Tie line")
ax[0, 0].plot(xru2, yru2, color = "red")
ax[0, 0].plot(xru3, yru3, color = "red")
ax[0, 0].plot(xru4, yru4, color = "red")
ax[0, 0].plot(xru5, yru5, color = "red")
leg = ax[0, 0].legend(loc="best", shadow=False, 
ncol=1, fontsize = "small")
ax[0, 0].set_title("Operation line", fontsize = 
"large")
ax[0, 0].set_xlabel("Temperature [ºC]", 
fontsize = "medium")
ax[0, 0].set_ylabel("iG [kJ/kg]",fontsize = 
"medium")

if pend1 == -float("inf"):
    y_iG = np.linspace(iG1, iG(TL2, param1), 
    1000)
    x_z = z_v_g(y_iG, param1)
    y_TL = TL(y_iG, param1)
elif pend1 == 0:
    y_TL = np.linspace(TL1, TL2, 1000)
    x_z = z_h_g(y_TL, param1)
    y_iG = iG(y_TL, param1)
else:
    y_iG = np.linspace(iG1, iG(TL2, param1), 
    1000)
    x_z = z_d_g(y_iG, param1)
    y_TL = TL(y_iG, param1)
```

```
ax[1, 0].plot(x_z, y_TL, color = "red",
label="TL")
leg = ax[1, 0].legend(loc="best", shadow=False,
ncol=1)
ax[1, 0].set_title("TL profile")
ax[1, 0].set_xlabel("z [m]")
ax[1, 0].set_ylabel("TL [ºC]")


H_TG0 = [H1, TG1]
if pend1 == -float("inf"):
    Ti = y_TL
elif pend1 == 0:
    Ti = [fsolve(lambda T: iGi(T) - a, TL1)[0]
    for a in y_iG]
else:
    Ti = [fsolve(lambda T: (a - iGi(T)) -
    pend1*(b - T), TG1)[0] for a, b in
    zip(y_iG, y_TL)]
Hi = [fsolve(lambda H: iGi(a) - iG_H_TG(H, a),
H1)[0] for a in Ti]
if pend1 == 0:
    H = Hi
    TG = Ti
    x_z0 = 0, 0
    y_H1 = H1, H[0]
    y_TG1 = TG1, TG[0]
else:
    H = np.empty_like(x_z)
    TG = np.empty_like(x_z)
    H[0] = H_TG0[0]
    TG[0] = H_TG0[1]
    for i in range(1,1000):
        zspan = [x_z[i-1],x_z[i]]
        H_TG = odeint(perfils_H_TG, H_TG0,
        zspan, args=(Hi[i], Ti[i], param1))
        H[i] = H_TG[1][0]
```

```
                    TG[i] = H_TG[1][1]
                    H_TG0 = H_TG[1]

                ax[0,   1].plot(x_z,   H,   color   =   "blue",
                label="H")
                if pend1 == 0:
                    ax[0,1].plot(x_z0, y_H1, color = "blue")
                leg = ax[0, 1].legend(loc="best", shadow=False,
                ncol=1)
                ax[0, 1].set_title("H profile")
                ax[0, 1].set_xlabel("z [m]")
                ax[0, 1].set_ylabel("H [kg/kg]")

                ax[1,   1].plot(x_z,   TG,   color   =   "red",
                label="TG")
                if pend1 == 0:
                    ax[1,1].plot(x_z0, y_TG1, color = "red")
                leg = ax[1, 1].legend(loc="best", shadow=False,
                ncol=1)
                ax[1, 1].set_title("TG profile")
                ax[1, 1].set_xlabel("z [m]")
                ax[1, 1].set_ylabel("TG [ºC]")

                plt.tight_layout()
                canvas = FigureCanvasTkAgg(fig1, master=window)
                canvas.draw()
                canvas.get_tk_widget().grid(row=1,    column=5,
                rowspan=9)
                plt.close()

def clear():
    kYa_w.set("")
    hLa_w.set("")
    L_w.set("")
    G_w.set("")
    TL1_w.set("")
    TL2_w.set("")
    H1_w.set("")
    TG1_w.set("")
    z_w.configure(state="normal")
    z_w.delete(0, tk.END)
    z_w.configure(state="readonly")
    fig1 = plt.figure(figsize=(7.5, 6))
    canvas = FigureCanvasTkAgg(fig1, master=window)
    canvas.draw()
    canvas.get_tk_widget().grid(row=1, column=5, rowspan=9)
    plt.close()

window = tk.Tk()
window.title("Water cooling or air dehumidification tower")

kYa_w = tk.StringVar()
hLa_w = tk.StringVar()
L_w = tk.StringVar()
G_w = tk.StringVar()
TL1_w = tk.StringVar()
TL2_w = tk.StringVar()
H1_w = tk.StringVar()
TG1_w = tk.StringVar()

tk.Label(window, text = "Parameters", font=(20)).grid(row = 0,
column = 0, columnspan = 2)
tk.Label(window, text = "Result", font=(20)).grid(row = 0,
column = 2, columnspan = 3)
tk.Label(window, text = "Graphs", font=(20)).grid(row = 0,
column = 5)
tk.Label(window, text = "kYa [kg/(s·m^3)]", width=15).grid(row
= 7, column = 0)
tk.Label(window, text = "hLa [kJ/(s·ºC·m^3)]").grid(row = 8,
column = 0)
tk.Label(window, text = "L/S [kg/(s·m^2)]").grid(row = 5,
column = 0)
```

```
tk.Label(window, text = "G'/S [kg/(s·m^2)]").grid(row = 6, column = 0)
tk.Label(window, text = "TL1 [ºC]").grid(row = 2, column = 0)
tk.Label(window, text = "TL2 [ºC]").grid(row = 1, column = 0)
tk.Label(window, text = "H1 [kg/kg]").grid(row = 4, column = 0)
tk.Label(window, text = "TG1 [ºC]").grid(row = 3, column = 0)
tk.Label(window, text = "z [m]", width=15).grid(row = 1, column = 3)

tk.Entry(window, textvariable = kYa_w, justify="center", width=15).grid(row = 7, column = 1)
tk.Entry(window, textvariable = hLa_w, justify="center", width=15).grid(row = 8, column = 1)
tk.Entry(window, textvariable = L_w, justify="center", width=15).grid(row = 5, column = 1)
tk.Entry(window, textvariable = G_w, justify="center", width=15).grid(row = 6, column = 1)
tk.Entry(window, textvariable = TL1_w, justify="center", width=15).grid(row = 2, column = 1)
tk.Entry(window, textvariable = TL2_w, justify="center", width=15).grid(row = 1, column = 1)
tk.Entry(window, textvariable = H1_w, justify="center", width=15).grid(row = 4, column = 1)
tk.Entry(window, textvariable = TG1_w, justify="center", width=15).grid(row= 3 , column = 1)
z_w = tk.Entry(window, justify="center", width=15)
z_w.grid(row= 1 , column = 4)
z_w.configure(state="readonly")
png0 = PIL.Image.open("app_image.png")
pngr = png0.resize((500, 450), resample=4)
pngf = ImageTk.PhotoImage(pngr)
labelphoto = tk.Label(window, image = pngf).grid(row = 9, column = 0, rowspan = 2, columnspan = 5)

fig1 = plt.figure(figsize=(7.5, 6.2))
canvas = FigureCanvasTkAgg(fig1, master=window)
canvas.draw()
canvas.get_tk_widget().grid(row=1, column=5, rowspan=9)
plt.close()

clear_but = tk.Button(window, text = "Clear all", font=(15), fg
= "white", bg = "red", padx = 35, pady = 5, command =
clear).grid(row = 3, column = 2, rowspan = 3, columnspan = 3)
solve_but = tk.Button(window, text = "Solve", font=(15), fg =
"white", bg = "green", padx = 46, pady = 5, command =
answer).grid(row = 6, column = 2, rowspan = 3, columnspan = 3)

window.mainloop()
```