Analyzing the Opportunities for NIPAAm Dehumidification in Air Conditioning Systems

by

Jordan Daniel Kocher

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

When air is supplied to a conditioned space, the temperature and humidity of the air often contribute to the comfort and health of the occupants within the space. However, the vapor compression system, which is the standard air conditioning configuration, requires air to reach the dew point for dehumidification to occur, which can decrease system efficiency and longevity in low temperature applications.

To improve performance, some systems dehumidify the air before cooling. One common dehumidifier is the desiccant wheel, in which solid desiccant absorbs moisture out of the air while rotating through circular housing. This system improves performance, especially when the desiccant is regenerated with waste or solar heat; however, the heat of regeneration is very large, as the water absorbed during dehumidification must be evaporated. N-isopropylacrylamide (NIPAAm), a sorbent that oozes water when raised above a certain temperature, could potentially replace traditional desiccants in dehumidifiers. The heat of regeneration for NIPAAm consists of some sensible heat to bring the sorbent to the regeneration temperature, plus some latent heat to offset any liquid water that is evaporated as it is exuded from the NIPAAm. This means the NIPAAm regeneration heat has the potential to be much lower than that of a traditional desiccant.

Models were created for a standard vapor compression air conditioning system, two desiccant systems, and two theoretical NIPAAm systems. All components were modeled for simplified steady state operation. For a moderate percent of water evaporated during regeneration, it was found that the NIPAAm systems perform better than standard vapor compression. When compared to the desiccant systems, the NIPAAm systems performed better at almost all percent evaporation values. The regeneration heat was modeled as if supplied by an electric heater. If a cheaper heat source were utilized, the case for NIPAAm would be even stronger.

Future work on NIPAAm dehumidification should focus on lowering the percent evaporation from the 67% value found in literature. Additionally, the NIPAAm cannot exceed the lower critical solution temperature during dehumidification, indicating that a NIPAAm dehumidification system should be carefully designed such that the sorbent temperature is kept sufficiently low during dehumidification.

Dedicated to my family: Eric, Vickie, Alison, Molly, and Lucy.

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Robert Wang, who served as both my Master's and undergraduate thesis advisor. Since joining his research group, Dr. Wang has imparted much of his knowledge and experience to me. He has helped prepare me for graduate school and my future career, regardless of the path I choose. Over the years Dr. Wang has introduced me to many topics of research in which I am now greatly interested, and for that I am very thankful.

Additionally, I would like to acknowledge and thank Dr. Patrick Phelan and Dr. Kristen Parrish for serving on my thesis committee. Dr. Phelan also served on my undergraduate thesis committee and taught several of the courses I took as a Master's student; I am very grateful of the knowledge he has imparted to me over the years.

TABLE OF CONTENTS

LIST OF TABLES

TABLE OF FIGURES

Figure Page

LIST OF NOMENCLATURE

C	heat rate, i.e.: the product of mass flow rate and specific heat
ΔC	absorbed moisture per unit sorbent mass
COP	coefficient of performance
c_p	specific heat
$\dot{E}$	electrical power
h	specific enthalpy
m	mass
$\dot{m}$	mass flow rate
$\dot{Q}$	rate of heat transfer
RH	relative humidity
T	temperature
UA	heat transfer coefficient
V	volume
$\dot{W}$	compressor power
x	humidity ratio

Subscripts

a	dry air
des	desiccant
e	evaporator
f	final
HX	heat exchanger
i	inlet
$isen$	isentropic
o	outlet
$outside$	describes the properties of the outside air
$outside,p$	flow rate of outside air that is sent to the process air stream
$outside,s$	flow rate of outside air that is sent to the supply air stream
p	process
r	return
ref	refrigerant
$return$	denotes the total return air flow rate and properties of the return air
$return,p$	flow rate of return air that is sent to the process air stream
$return,s$	flow rate of return air that is sent to the supply air stream
s	supply
sat	saturated
tot	total air (dry air plus moisture)
v	water vapor
VC	vapor compression
w	liquid water

Greek letters

ε	heat exchanger effectiveness
η	efficiency
ω	rotational speed

1 INTRODUCTION

1.1 Background

Dehumidification is a process that is useful in many scenarios. Indoor air must often have a relative humidity within a certain range to produce comfortable conditions for occupants and inhibit mold growth. The EPA states that the relative humidity of indoor air should remain between 30 and 60% to inhibit the growth of mold, while ASHRAE states that relative humidity should remain below 65% for human comfort [1]. Certain types of buildings, like supermarkets and ice rinks, require precise and often relatively low humidity levels [2]. Thus, it is important that the heating, ventilation, and air conditioning (HVAC) system of a building properly controls the air humidity level, as well as temperature.

The humidity of indoor air can be controlled through several means. One of the most common methods, and often the simplest, is to use a traditional vapor compression refrigeration cycle air conditioner. At the moment, vapor compression air conditioning is the most prominent method of providing cool air to a space [3], and these systems require no additional components to dehumidify the air, as the cooling and dehumidification takes place simultaneously. However, in certain scenarios, these systems can be inefficient or ineffective, at which point a supplemental dehumidifying component becomes necessary to make the system cost effective or to reach the desired humidity. One such dehumidifier is a rotary desiccant wheel. This component, pictured below, absorbs moisture out of the air and into the desiccant material.

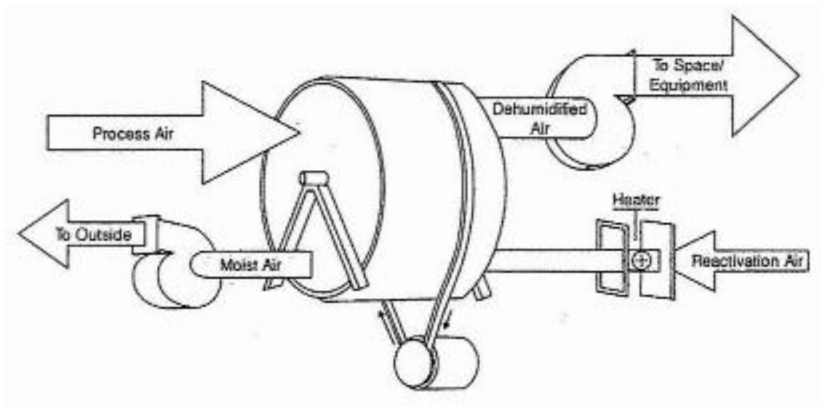


Fig. 1.1.1: Desiccant wheel with a dehumidification section (top) and regeneration section (bottom) [2]

To allow for continuous use, the water that is absorbed in the dehumidification portion of the wheel must be desorbed and evaporated off, which requires heat input. While desiccant dehumidification provides an improvement to performance and efficiency in many scenarios, the efficiency would be improved further if the dehumidifying wheel required less energy input for regeneration.

1.2 HVAC Review

The HVAC system within a building provides air that is fresh (through ventilation) and comfortable (through heating or cooling, depending on the outdoor conditions). As well as controlling temperature, the HVAC system controls the humidity of the air, as a combination of temperature and humidity determine the comfort of building occupants [1]. Most HVAC systems control humidity by either cooling the supply air below the dew point, at which point moisture in the air is forced to condense as the air continues to cool, or by absorbing moisture out of the air [2]. The dew point of air at a given humidity ratio is the temperature at which the air becomes saturated with moisture and cannot cool any further without giving up some moisture. Vapor compression HVAC systems, which are the dominant type of system used to cool buildings [3], utilize the dew point condensation method of dehumidification. Vapor compression refrigeration systems contain four main components: a compressor, condenser, expansion valve, and evaporator. These components create a refrigeration cycle; first, refrigerant vapor enters the compressor, at which point the temperature and pressure are significantly increased. Next, the superheated vapor enters the condenser, at which point a fan blows outside air over the condenser coils, which cools the refrigerant and causes it to condense. After leaving the condenser, the liquid refrigerant enters the expansion valve, at which point it cools and drops to a lower pressure. Finally, the refrigerant flows through the evaporator. When the cool refrigerant flows through the evaporator, which is placed within the building, a fan blows supply air over the evaporator coils. The supply air is cooled by the coils and sent to the conditioned space. During this cooling process, if the evaporator coils are significantly colder than the dew point of the supply air, some of the moisture is forced to condense out of the air, thus decreasing the humidity ratio of the air.

In certain locations, ventilation of fresh, outside air into buildings can significantly increase the latent load (moisture to be removed) of the air supplied to the conditioned space. Due to increased concern about the effects of indoor air quality on occupant health, building ventilation rates have increased over the years [2]. When the ventilation constraints in a humid location require a significant portion of outside air to be provided to the space, discomfort associated with the air can occur in one of two ways. It is possible that the vapor compression air conditioner will not be able to dehumidify the air to a comfortable humidity level, due to a lack of cooling power, or due to an inability to reach a sufficiently low evaporator temperature. It is also possible that the vapor compression system is able to dehumidify the air sufficiently, but the dew point of the dry air is so low that it becomes too cold to be comfortable for the occupants. In this scenario, the air may need to be reheated to reach comfortable levels, which would introduce inefficiency to the overall process [2].

In certain cases, condensation based dehumidification from vapor compression systems is the most desirable method for dehumidifying air, as vapor compression systems are common, and they often have relatively high coefficient of performance (COP) values. However, in many scenarios, the coupling of outlet temperature and humidity associated with condensation based dehumidification makes a separate dehumidification system desirable.

1.3 Desiccant Dehumidification Review

While vapor compression air conditioning systems are capable of dehumidifying air, there are many cases in which dehumidification by cooling the air beyond its dew point is ineffective or inefficient. When the dew point of the cooled supply air is below the freezing point of water, frost will accumulate on the evaporator coils as the condensed moisture begins to freeze, which can negatively affect performance. This scenario is common in buildings which require that the humidity or temperature of the space be kept at a significantly low value, such as supermarkets and ice rinks [2]. Aside from issues with frost, there are some scenarios where the required humidity of the supply air corresponds to a dew point that is too low for comfort, in which case the conditioned air leaving the evaporator coils must be reheated before it is supplied to the conditioned space [2]. For these scenarios, it would be more efficient to dehumidify the air to the desired humidity ratio first, after which the air could be cooled to, but not beyond, the dew point, thus preventing the buildup of frost on the evaporator coils. A desiccant dehumidifier is an example of a device that could be place in-line with a traditional cooling system to dehumidify the air first. While there are many types of desiccant dehumidifiers, the rotary solid desiccant wheel is a commonly used variation, as it allows for constant use, as pictured in Fig. 1.1.1. Air is passed over the desiccant in the wheel, and the difference in vapor pressure between the desiccant surface and the air causes the desiccant to absorb the moisture [4]. During the absorption process, the heat is released from the water vapor to the air and desiccant material. To pre-cool the supply air before sending it to the cooling system, a heat exchanger can be utilized to transfer heat between the supply air and a stream of process air, which is often a combination of outside air and return air from the conditioned space. This pre-cooling brings the temperature of the supply air closer to room temperature and allows the vapor compression cooling system to do less work for the same overall process. The pre-cooling of the supply air also serves to pre-heat the process air, which can then be used to regenerate the saturated desiccant in the wheel. Usually, the heat from the heat exchanger is not enough to heat the process air to the regeneration temperature necessary to dry out the desiccant, in which case a supplemental heat source is required. While an electric heater could be used as the heating device, some systems utilize natural gas as the heat source. In many of these scenarios, the difference between the price of natural gas and on-peak electricity is leveraged and the system is run during peak hours to reduce the electricity load associated with the vapor compression system [2]. Other systems utilize solar or waste heat to regenerate the desiccant [2]. These scenarios are desirable as they require no extra fuel cost for regeneration, but the heat source must be readily available and able to reach the required regeneration temperature.

Desiccant dehumidification can be an economic and efficient method of removing moisture when the latent load of the air is high, and the method of regeneration is cost effective. However, desiccant dehumidification also presents another interesting application in desiccant air conditioning. Desiccant air conditioning is simply a system that utilizes a desiccant dehumidifier in line with a heat exchanger and an evaporative cooler [4]. In this configuration, the supply air is dried and heated by the dehumidifier, pre-cooled by the heat exchanger, and humidified and cooled by the evaporative cooler. The pre-cooling from the heat exchanger allows for a net cooling and drying process to occur, as illustrated on the psychrometric chart in the figure below.

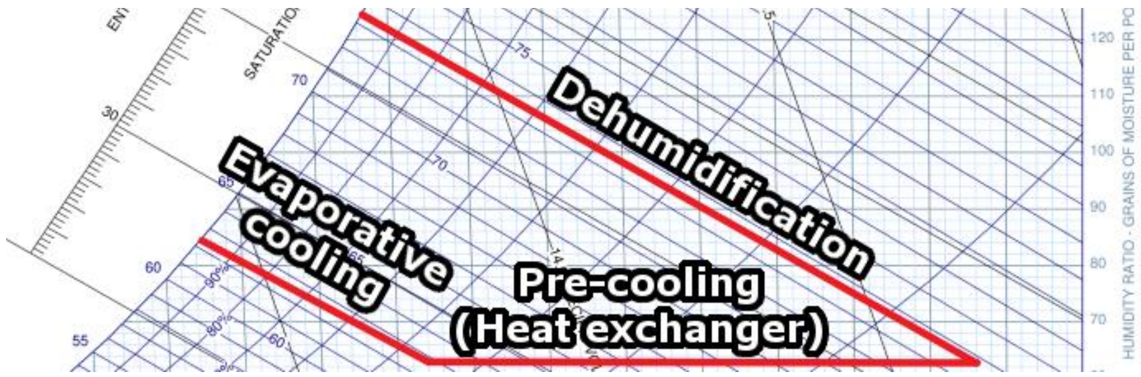


Fig. 1.3.1: Desiccant air conditioning process

This type of system can be desirable for two reasons. Desiccant air conditioning uses water to achieve the refrigeration effect, while vapor compression systems use hydrofluorocarbons, which have high global warming potentials [5]. Additionally, if the system is regenerated with waste heat, solar heat, or natural gas, desiccant air conditioning can be cheaper than a vapor compression system in locations where electricity is relatively expensive. If the consumption of water is favorable to the consumption of electricity, desiccant air conditioning can serve as a viable replacement to vapor compression air conditioning. However, depending on the operating conditions and system design, the required regeneration temperature can exceed 100 °C [4]. A desiccant material with a lower regeneration temperature could significantly improve the efficiency and cost effectiveness of desiccant dehumidifiers and desiccant air conditioning systems.

1.4 NIPAAm Review

Traditional desiccants are regenerated by causing the absorbed water to desorb, which requires enough heat to offset the energy of bonding between the water molecules and the sorbent, and enough heat to vaporize the water molecules [6]. However, there are certain sorbents that, in response to a slight temperature change, give off the absorbed water as a liquid. If one of these materials were leveraged properly in a desiccant dehumidifier, it could be regenerated with far less heat than a traditional desiccant.

Poly(N-isopropylacrylamide) (PNIPAAm) is a hydrogel that responds to a change in temperature. When the polymer is raised above its lower critical solution temperature (LCST), it changes from hydrophilic to hydrophobic. This means the PNIPAAm tends to absorb moisture out of air when it is below the LCST, but it will ooze some of the absorbed moisture when it is raised above the LCST. The LCST of PNIPAAm is approximately 32 °C [7]. Sodium alginate (Alg) is a highly hydrophilic material, and researchers created an interpenetrating polymer network (IPN) gel that consisted of PNIPAAm chains, to ooze water upon temperature response, and Alg chains to improve the absorption capacity of the gel [7]. While the researchers in this group also created several other materials based on N-isopropylacrylamide (NIPAAm), the main focus of their research was on the IPN gel. The researchers were able to produce a sample that absorbed moisture out of humid air below the LCST and regenerated by giving off a mix of liquid water and water vapor when raised above the LCST [7]. To see the maximum potential of a thermo-responsive NIPAAm based hydrogel, the NIPAAm should be synthesized and the system should be constructed such that the amount of water that is evaporated during regeneration is minimized, as this is the scenario in which the least amount of heat is necessary to drive regeneration. Additionally, the NIPAAm sorbent must stay below the LCST during dehumidification, otherwise the NIPAAM will be unable to sustain the dehumidification process. This presents a potential challenge, as the supply air heats up during dehumidification, and it is unclear how much heat from the dehumidification process will be transferred to the NIPAAm.

1.5 Objectives

The objective of this thesis is to present an analysis of a dehumidifier with NIPAAm as the sorbent. A simple steady state model was created for the NIPAAm dehumidifier, and system-level models were created for a NIPAAm dehumidifier in series with a vapor compression cooling system, as well as a NIPAAm dehumidifier in series with an evaporative cooler. To assess the performance of the proposed NIPAAm dehumidification process, models were created for a standard vapor compression system, a desiccant dehumidifier in series with a vapor compression cooling system, and a traditional desiccant air conditioning system. Cases were run for all five models, and the performance of each system configuration was compared.

While the goal of this report is to highlight the general potential of any thermo-responsive NIPAAm hydrogel, the PNIPAAm/Alg IPN was the specific material that was modeled. For the sake of simplicity, the term "NIPAAm" is hereafter used to refer to the PNIPAAm/Alg IPN.

Because the models created for this report are system-level models that do not capture certain details, such as the dynamic temperature response of the NIPAAm during dehumidification, it is simply assumed that the NIPAAm temperature does not reach the LCST during dehumidification (as this would stop the dehumidification process). This, however, is a non-trivial qualification, and it is unclear under what circumstances this is a valid assumption. A brief discussion is presented at the end of the report regarding the mitigation of temperature rise within the NIPAAm during dehumidification, and a design for an isothermal NIPAAm dehumidifier is presented.

After running the models for various cases, it was found that there is significant potential for a NIPAAm dehumidifier that could be designed to generally behave as it was modeled in this report. For high latent loads, the NIPAAm dehumidification and vapor compression cooling method was found to perform better than standard vapor compression, even when the percent of water evaporated during NIPAAm regeneration was high. For cases with a lower latent load, the NIPAAm dehumidification and vapor compression cooling outperformed standard vapor compression cooling for low percent evaporation values. The NIPAAm dehumidification and vapor compression cooling model was found to outperform the desiccant dehumidification and vapor compression cooling model in every case. The NIPAAm dehumidification and evaporative cooling was found to be much more efficient than desiccant dehumidification and evaporative cooling when at low percent evaporation values, an as the percent evaporation increased, the NIPAAm efficiency was found to approach the efficiency of the desiccant system. When waste heat source is used, the NIPAAm desiccant systems require the same amount of electricity input, but the NIPAAm requires less heat than the desiccant to regenerate, and the heat can be supplied at a lower temperature for the NIPAAm.

2 METHODS

2.1 Overview

To assess the potential of NIPAAm as a dehumidifying agent in HVAC applications, system models were created in Python for several different air conditioning configurations. One model was created for a standard vapor compression system, models were created for two systems containing desiccant dehumidification, and models were created for two systems containing NIPAAm dehumidification. The system models take inputs for the temperature and humidity of the air at the system inlet, as well as the desired temperature and humidity at the system outlet. The components for each system were modeled for steady state operation, and, upon input, the models determine the characteristics for each component, such as sorbent mass or electrical power that is drawn, that are necessary to bring the inlet air to the required outlet conditions. The component models are combined, by passing the outputs of an upstream component as inputs to the component immediately downstream, to form the overall system models. After the characteristics of each component are determined, the system model sums the required power input to all of the components and determines the COP by dividing the rate of cooling by the total power input. Aside from required power and COP, the models also output psychrometric charts of the processes. One of the desiccant systems and one of the NIPAAm systems utilize evaporative cooling; for these systems, the models also output the amount of water consumed by the evaporative cooler that is necessary to achieve the desired cooling process. The two NIPAAm configurations also output the amount of liquid water that is reclaimed during regeneration. A graphical interface was created in Python to display the outputs of each system model.

The function of each system model is to determine the energy that must be supplied to the system in order to achieve a desired cooling process. The user inputs the desired cooling provided by the system in the form of air temperature and humidity at the system inlet and outlet. The following is a list of the model inputs:

- Thermostat set temperature
- Indoor air humidity ratio
- Outdoor air temperature
- Outdoor air humidity ratio
- Desired temperature of the supply air
- Desired humidity ratio of the supply air

Once the values are set by the user, the initial temperature of the indoor air is set to 5/9 °C (1 °F) higher than the thermostat set temperature input by the user, as it is assumed that the air conditioning would switch on once the indoor temperature is 1 °F greater than the thermostat setting.

In the configurations modeled in this paper, the supply air is a mix of indoor return air and outdoor air. The air that returns from the conditioned space is often cooler and drier than the outdoor air, and thus increases system efficiency, while the outdoor air is fresh and improves ventilation. Before any dehumidification or cooling occurs, some mass flow rate of return air, $\dot{m}_{return,s}$, is combined with some mass flow rate of outside air, $\dot{m}_{outside,s}$, to form the supply air flow rate, $\dot{m}_{supply}$. Additionally, some of the configurations that were modeled utilize a “process” air stream to regenerate the dehumidifier. The process air stream is also a combination of return and outside air. The separate air streams are illustrated in the graphic below, where the black lines with arrows are air streams; the return air stream splits in two, with some portion being sent to the supply air stream, and the remainder being sent to the process air stream.

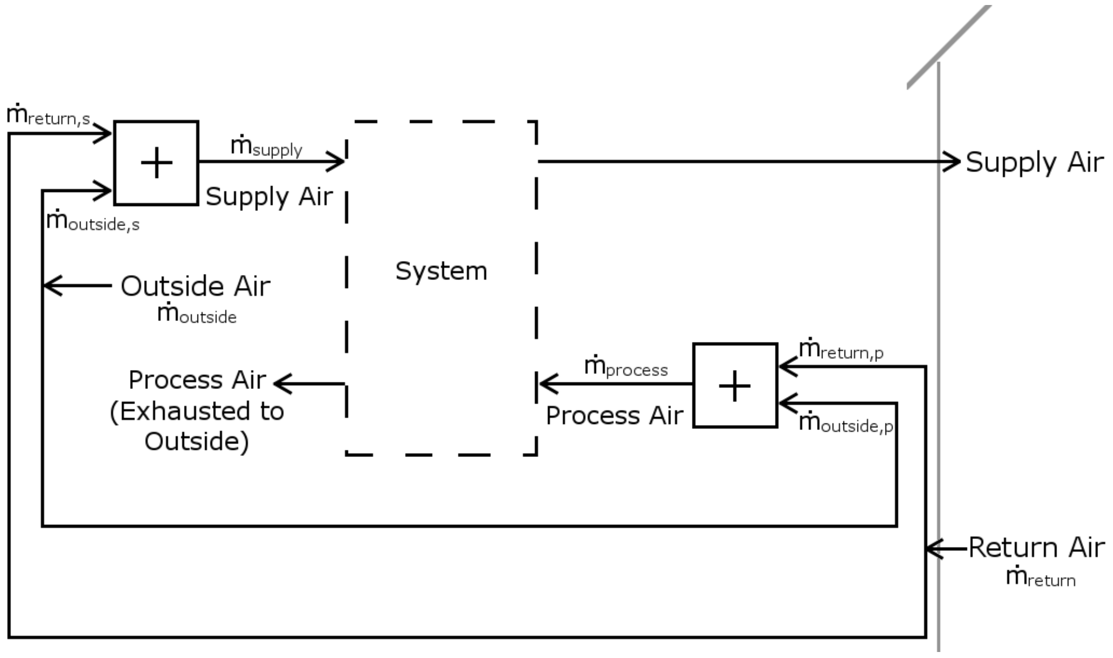


Fig. 2.1.1: Diagram of system airflows

Within the model, it is assumed that the supply and return air flow rates are the only airflows in or out of the house (i.e.: there is no infiltration or passive ventilation). Thus, to maintain constant pressure within the house, the supply air mass flow rate must be equal to the return air mass flow rate. Additionally, it is often advantageous for the process air flow rate to be equal to the supply air flow rate, as explained later in Section 2.2.2. Thus, the following equation holds true.

$$\dot{m}_{supply} = \dot{m}_{return} = \dot{m}_{process} \tag{2.1}$$

As seen in Fig. 2.1.1 above, some of the return air is provided to the supply air ($\dot{m}_{return,s}$), while the remainder is provided to the process air ($\dot{m}_{return,p}$), as illustrated in the equation below.

$$\dot{m}_{return} = \dot{m}_{return,s} + \dot{m}_{return,p} \tag{2.2}$$

When combining the previous equation with Eq. 2.3 and 2.10, it can be seen that the magnitude of the mass flow rate of the return air that is sent to the process air stream is the same as the magnitude of the mass flow rate of outdoor air that is sent to the supply air. Additionally, it can be seen that the magnitude of mass flow rate of the outdoor component of process air is the same as the magnitude of mass flow rate of the return air component of supply air.

$$\dot{m}_{return,p} = \dot{m}_{outside,s} \tag{2.3}$$

$$\dot{m}_{outside,p} = \dot{m}_{return,s} \tag{2.4}$$

The first step of each system model is to calculate the temperature and humidity ratio of the supply air after the return and outdoor air are mixed. This process is modeled using the equations below, where *x* is the humidity ratio of the air, which is the ratio of water vapor to dry air. The specific heats are evaluated at the average of the return and outside temperatures.

$$x_{s,i} = \frac{x_{return} * \dot{m}_{return,s} + x_{outside}\dot{m}_{outside,s}}{\dot{m}_{supply}} \tag{2.5}$$

$$T_{s,i} = \frac{T_{return}\dot{m}_{return,s}\left(c_{p,a} + x_{return}c_{p,v}\right)}{\dot{m}_{supply}\left(c_{p,a} + x_{s,i}c_{p,v}\right)} + \frac{T_{outside}\dot{m}_{outside,s}\left(c_{p,a} + x_{outside}c_{p,v}\right)}{\dot{m}_{supply}\left(c_{p,a} + x_{s,i}c_{p,v}\right)} \tag{2.6}$$

$$\dot{m}_{supply} = \dot{m}_{return,s} + \dot{m}_{outside,s} \tag{2.7}$$

Each of the aforementioned mass flow rates is a mass flow rate of dry air. The total mass flow rate of a given air stream is the sum of the dry air mass flow rate and the mass flow rate of the water vapor in the air, as shown in the equation below.

$$\dot{m}_{tot} = \dot{m}_a(1 + x) \tag{2.8}$$

The specific heat or specific enthalpy of an air stream can be found by adding the dry air property with the product of the water vapor property and the humidity ratio, as shown in the equations below.

$$h_{tot} = h_a + x * h_v \tag{2.9}$$

$$c_{p,tot} = c_{p,a} + x * c_{p,v} \tag{2.10}$$

All of the Python models described in this paper use a wrapper called CoolProp to access various air and water properties, like specific heat, specific enthalpy, and temperature.

The process air stream is used for the system configurations that utilize a separate dehumidifier. The standard vapor compression configuration, which does not include a separate dehumidifier, does not utilize any process air. For the other configurations, the process air is a mix of return and outside air, which, just like the supply air, must be mixed. After the supply air mixing is modeled, the process air mixing is modeled as well, using equations very similar to Eq. 2.5 and 2.6. The only difference is that the supply air mass flow rates, $\dot{m}_{supply}$, $\dot{m}_{return,s}$, and $\dot{m}_{outside,s}$, are replaced with the process air mass flow rates, $\dot{m}_{process}$, $\dot{m}_{return,p}$, and $\dot{m}_{outside,p}$, respectively.

After the air mixing is modeled, the program calculates the processes necessary to bring the supply air stream to the desired output temperature and humidity ratio specified by the user. In determining the necessary processes, the program also calculates the performance of the components that induce these processes, such as the dehumidifier, process air heater, and cooling unit. From this information, the power required for each component is determined.

2.2 System Model Configurations

2.2.1 Vapor Compression Only

To analyze the performance of a NIPAAm dehumidification system, five general system configurations were considered and modeled through the use of several Python scripts. The configuration of the first model consists of a traditional vapor compression cycle air conditioning system. In this configuration, dehumidification occurs as the AC evaporator cools the supply air past its dew point, forcing water to condense on the evaporator coils. Because this configuration consists solely of a vapor compression air conditioning system, this configuration is hereafter referred to as “vapor compression only” or “standard vapor compression”. For this configuration, as well as the remaining ones, the air supplied to the conditioned space is a mix of return air and outside air. The schematic for this configuration is shown in the figure below.

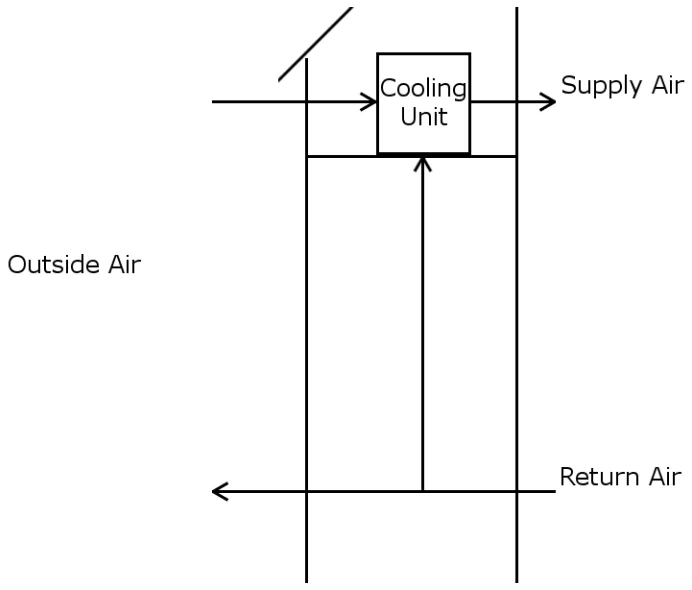


Fig. 2.2.1.1: Vapor compression only schematic

As mentioned previously, the return air flow rate is set equal to the supply air flow rate to keep constant pressure within the house. In the other configurations, some of the return air is used for the regeneration process; however, in this configuration, no process air is need as there is no dehumidifying wheel to regenerate, so some of the return air is exhausted to the outside. This has the same effect as building leakage and purposeful ventilation.

An example case illustrating the general process of the supply air in a vapor compression only configuration is shown on the psychrometric chart in the figure below.

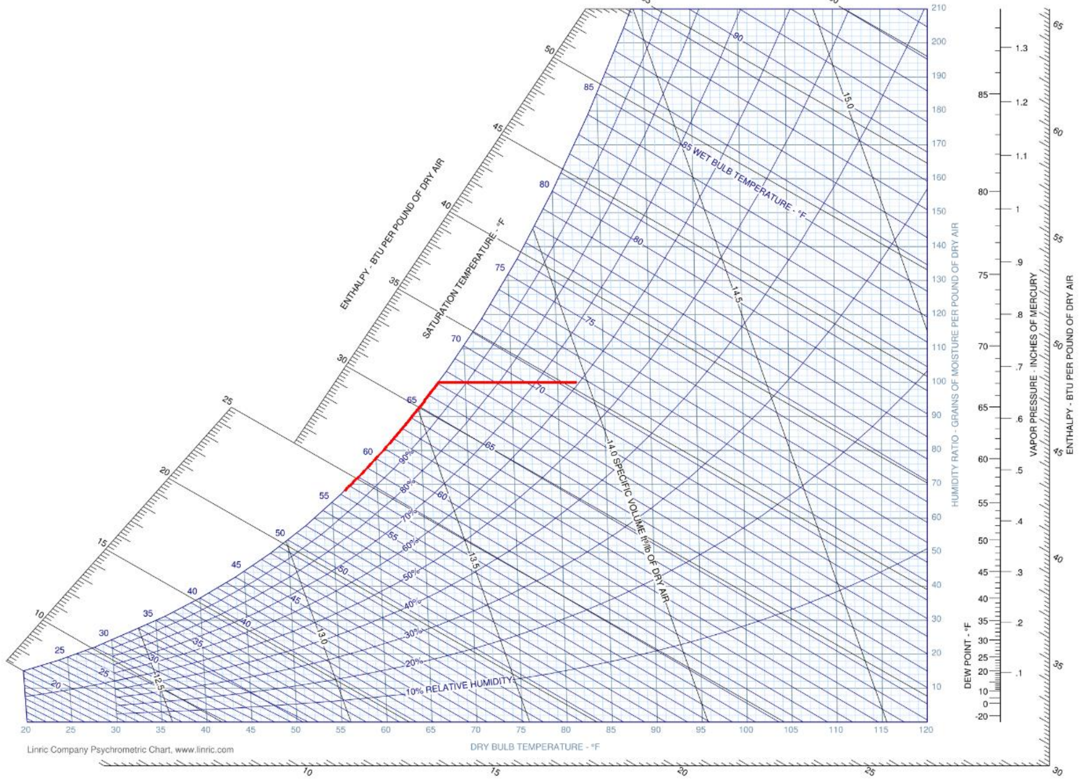


Fig. 2.2.1.2: Psychrometric chart of supply air cooled by a vapor compression only air conditioning system

To describe the performance of the system, the model calculates a COP, which depends on the useful cooling and the power consumed by the AC compressor, as described in the following equations, where $h_{house,i}$ is the specific enthalpy of the return air leaving the space, $h_{s,o}$ is the specific enthalpy of the cooled supply air, $\dot{m}_{ref}$ is the mass flow rate of the AC refrigerant, $h_{ref,2}$ is the specific enthalpy of the refrigerant as it leaves the evaporator and enters the compressor, and $h_{ref,3}$ is the specific enthalpy of the refrigerant as it leaves the compressor and enters the condenser.

$$\dot{Q}_{cool} = \dot{m}_{return} h_{house,i} - \dot{m}_{supply} h_{s,o} \qquad (2.11)$$

$$\dot{W}_{VC} = \dot{m}_{ref}\left(h_{ref,3} - h_{ref,2}\right) \tag{2.12}$$

$$COP = \frac{\dot{Q}_{cool}}{\dot{W}_{VC}} \tag{2.13}$$

It should be noted that any pumps, fans, or blowers in this configuration or any of the following configurations were not modeled, and thus the power required to run these elements was not considered in any COP calculations. The compressor is assumed to be adiabatic, but not isentropic. An isentropic efficiency was included in the calculation of the change in specific enthalpy across the compressor; the value selected for isentropic efficiency is discussed in Section 2.6.

2.2.2 Desiccant Dehumidification and Vapor Compression Cooling

The configuration for the second model consists of a desiccant dehumidifier in conjunction with a vapor compression air conditioner, and this configuration is called "desiccant dehumidification and vapor compression cooling." The model for this system was created such that it could be applied to any traditional desiccant, such as silica gel or a zeolite. To model this system, a rotary desiccant wheel was selected as the dehumidification component. The figure below shows a schematic for this configuration.

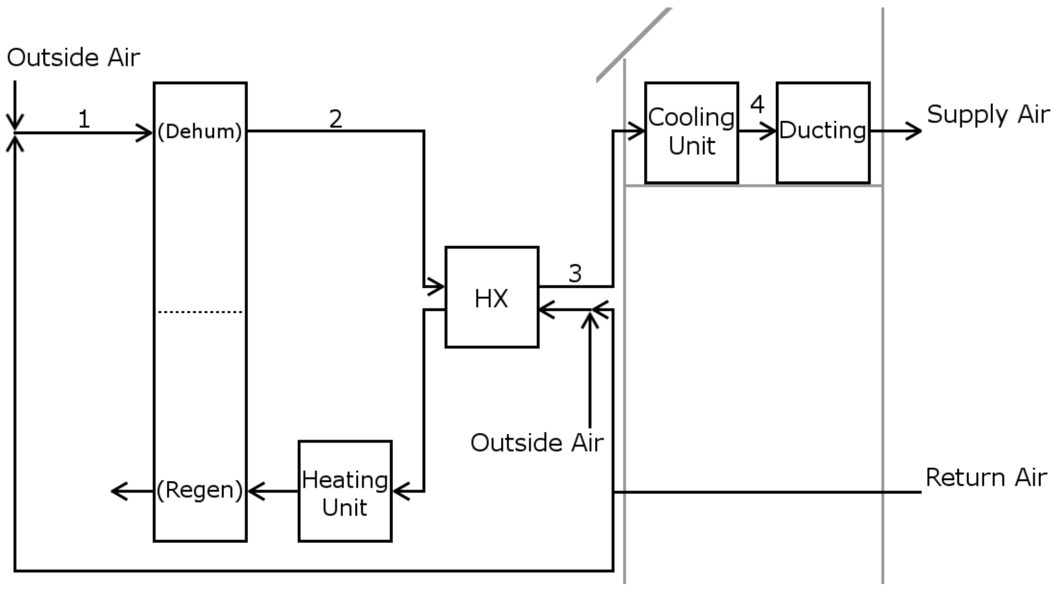


Fig. 2.2.2.1: Desiccant dehumidification and vapor compression cooling schematic

The general process of the supply air for this configuration is shown in the figure below.

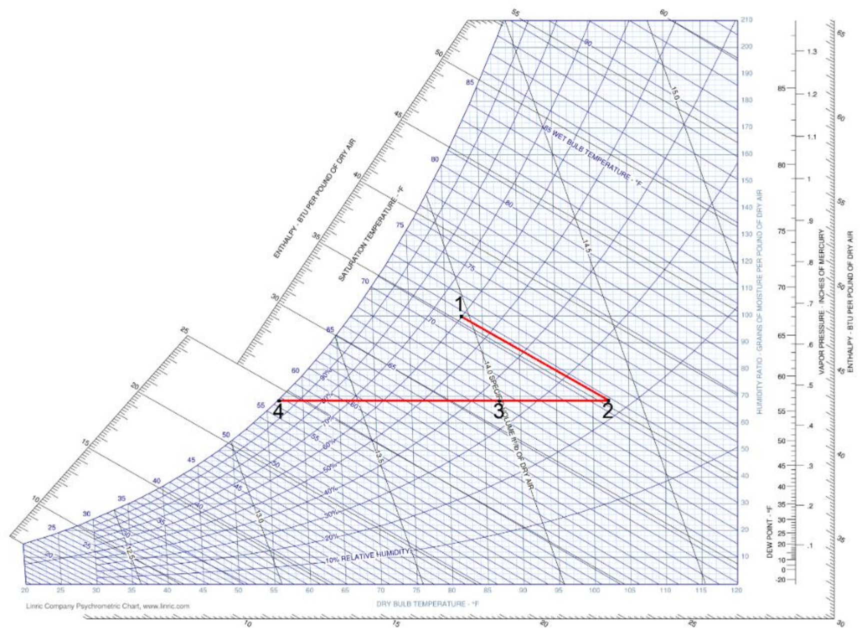


Fig. 2.2.2.2: Psychrometric chart for supply air in a desiccant dehumidification and vapor compression cooling system

At the beginning of this cycle, the supply air is sent to a portion of the desiccant wheel, at which point the humidity ratio of the air drops while the temperature of the air rises, as the latent heat of condensation is being converted to sensible heat within the air. When creating the model for this configuration, it was assumed that the air leaving the desiccant wheel has the same humidity ratio as the air leaving the evaporator coils in the previous configuration, as this allows for a direct comparison of the two configurations. After dehumidification, the air is then sent through a heat exchanger and pre-cooled, with the air on the other side of the heat exchanger, the process air, being a mix of return and outside air as well. As previously mentioned, the process air mass flow rate was set equal to the supply air mass flow rate. This was done to ensure the temperature increase of the process air across the heat exchanger is the same as the temperature drop of the supply air across the heat exchanger. After pre-cooling in the heat exchanger, the supply air, now dried to the desired humidity ratio but still hotter than desired, is sent to the vapor compression system, where the evaporator coils can cool the air to the desired temperature.

In this configuration, less cooling is required from the AC system, as no condensation needs to occur, which means the AC system compressor does not do as much work. At this point, the same net cooling effect is achieved with less input electricity to the AC system, indicating that the COP should increase. However, this configuration requires an additional energy input in the form of heat. While one part of the desiccant wheel is dehumidifying, the remainder must be regenerated to keep the dehumidification process constant. To do this, heat must be put into the desiccant through the process air stream. The hot process air causes the water to desorb and vaporize, thus drying out the desiccant. While the aforementioned heat exchanger serves to pre-cool the supply air, it also serves to pre-heat the process air; however, the temperature of the process air exiting the heat exchanger is often insufficient to regenerate the desiccant wheel. Thus, a heating element is placed downstream of the heat exchanger and is used to heat the process air to the required temperature. The following equation describes the rate of heat transfer required to achieve the regeneration temperature, where $\dot{m}_{process}$ is the mass flow rate of the dry process air, $T_{p,HX,o}$ is the temperature of the process air as it leaves the heat exchanger, $T_{p,regen}$ is the required regeneration temperature, $c_{p,a}$ is the specific heat of the dry air, $x_{p,i}$ is the humidity ratio of the process air, and $c_{p,v}$ is the specific heat of the water vapor in the process air.

$$\dot{Q}_{regen,des} = \dot{m}_{process}\left(c_{p,a} + x_{p,i}c_{p,v}\right)\left(T_{p,regen} - T_{p,HX,o}\right) \tag{2.14}$$

For this model, it is assumed that a lossless and perfectly efficient electric heater is used to heat the process air, such that all electrical input, $\dot{E}_{regen}$, is converted to sensible heat, $\dot{Q}_{regen,des}$, within the air stream. The following equation describes the COP of this configuration and reflects the second input to the system.

$$COP = \frac{\dot{Q}_{cool}}{\dot{W}_{VC} + \dot{E}_{regen}} \tag{2.15}$$

For some section of the desiccant wheel in the regenerating portion, the desiccant must be dried out by the time it rotates back to the dehumidifying portion. In order to dry out the desiccant at the proper rate, the process air stream must reach a certain regeneration temperature. The temperature necessary for a desired rate of desorption depends on system geometry and other aspects not considered in the models described within this paper; thus, the required regeneration temperature is not obvious. To account for this, the model for this configuration was designed to allow the user to select various regeneration temperatures in the graphical interface and observe the resulting system performance.

2.2.3 NIPAAm Dehumidification and Vapor Compression Cooling

As with the previous configuration, the model for the third configuration includes a dehumidifying wheel and a vapor compression air conditioner; however, in this configuration, the dehumidifying material is NIPAAm. This configuration is hereafter called the "NIPAAm dehumidification and vapor compression cooling" configuration. The general process of the supply air for this case is the same as the previous case. The overall system configuration for this model is very similar to the second configuration, with the exception of the regeneration process, as shown in the figure below.

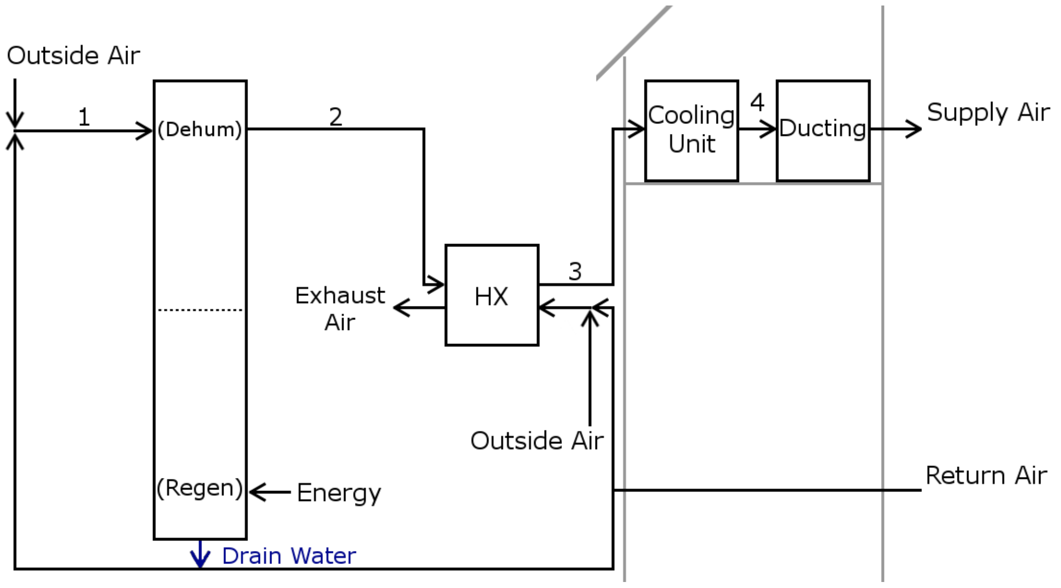


Fig. 2.2.3.1: NIPAAm dehumidification and vapor compression cooling schematic

While the regenerating portion of the desiccant wheel uses a hot air stream to dry the desiccant, this process is not ideal for NIPAAm regeneration. To regenerate a traditional desiccant, the absorbed water must be vaporized during the desorption process, which is why a hot air stream is used. With NIPAAm, however, regeneration can occur by raising the material above its LCST, at which point the NIPAAm transitions from hydrophilic to hydrophobic, and the absorbed water is expelled in liquid form. In this situation, it is desirable to have no evaporation occur while the NIPAAm is being heated, as any evaporation would require heat that would otherwise increase the temperature of the NIPAAm, which is the desired effect. Thus, for the NIPAAm dehumidification and vapor compression cooling configuration, the process air stream is exhausted to the outside after exiting the heat exchanger, and the regeneration heat is supplied directly to the regenerating portion of the wheel.

Because the NIPAAm becomes hydrophobic above the LCST, it is important that the NIPAAm stays below the LCST on the dehumidification side of the desiccant wheel. Because the regeneration process involves heating it to the LCST, it is likely that some supplemental cooling must be provided before the NIPAAm can begin the dehumidification process again. This process is shown in the figure below, which illustrates a potential NIPAAm dehumidification wheel design.

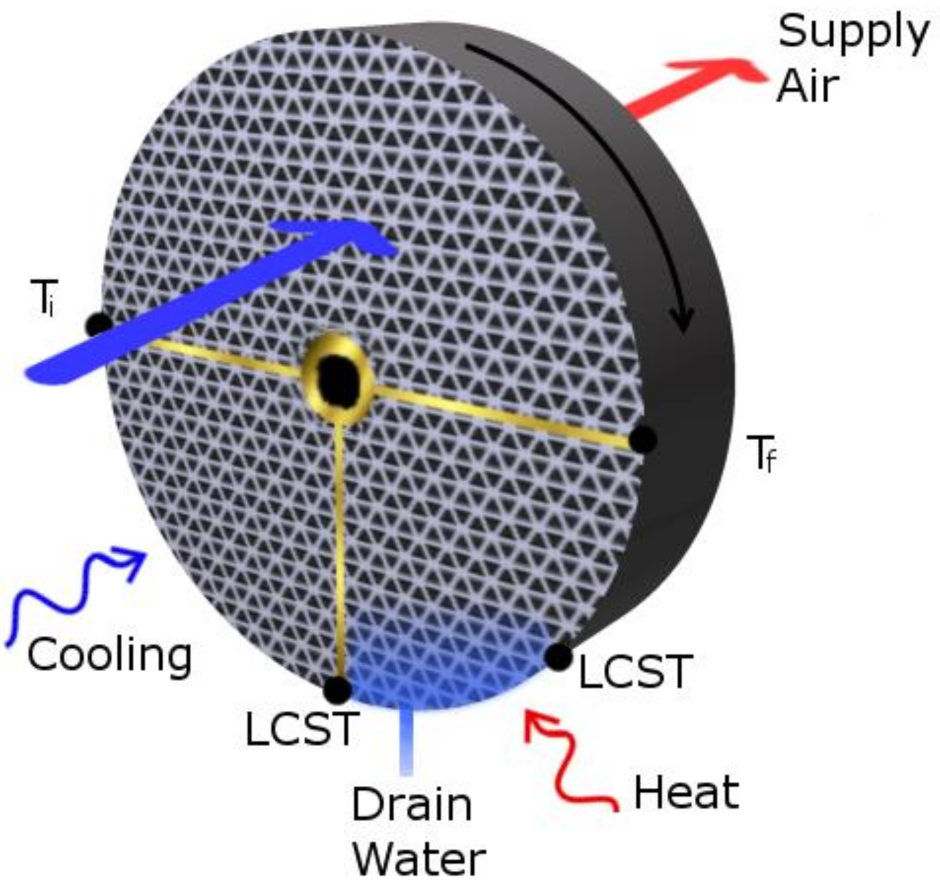


Fig. 2.2.3.2: Proposed design for a rotary NIPAAm dehumidifying wheel

The rotary NIPAAm wheel has three distinct sections. The upper section is the dehumidification portion, where moist air enters the dehumidifier and leaves hot and dry. The NIPAAm at the point where the upper section of the wheel begins is at some initial temperature, $T_{NIPAAm,i}$. As the wheel rotates and the NIPAAm travels clockwise, it is heated by the sorption process. At the end of the upper section, the NIPAAm is at some final temperature for the dehumidification process, $T_{NIPAAm,f}$. The next section of the wheel, shown in the bottom right of the figure, is where the NIPAAm is regenerated. Heaters are placed within the wheel housing to heat the NIPAAm to the LCST, at which point it begins draining the water. While the water is draining, some percentage of it could evaporate, which would cool the NIPAAm. Heaters would then be needed for the draining section as well, in order to maintain the LCST and offset any evaporative cooling that might occur. The final section of the wheel contains coolers to bring the NIPAAm from the LCST to the initial dehumidification temperature, $T_{NIPAAm,i}$. It is assumed that the vapor compression cooling system used to cool the supply air is able to provide some supplemental cooling to reduce the NIPAAm temperature.

The energy required to regenerate the NIPAAm is the sensible heat required to raise the NIPAAm and any absorbed water from the final dehumidification temperature, $T_{NIPAAm,f}$, to the LCST, plus the latent heat of any evaporation that occurs, plus the cooling required to bring the NIPAAm temperature back down to $T_{NIPAAm,i}$. The heaters are once again assumed to be lossless and perfectly efficient, meaning the required heat is also the required electricity input. For the cooler, the electricity input is the required cooling divided by the COP of the vapor compression cooler, as it is assumed that the vapor compression system is able to provide cooling to this portion of the wheel.

For the proposed NIPAAm wheel design, the amount of NIPAAm necessary is dependent on the rate of dehumidification, which depends on the supply air flow rate and humidity drop, the increase in water to NIPAAm mass ratio, ΔC_{NIPAAm}, and the rotational speed of the wheel, ω, which is expressed in deg/s in the equation below.

$$m_{NIPAAm} = \frac{\dot{m}_{supply}\left(x_{s,i} - x_{s,o}\right)}{\Delta C_{NIPAAm}} \frac{360}{\omega} \tag{2.16}$$

The required energy input can be separated into three parts: sensible heating, latent heating, and sensible cooling. The rate of heat transfer for each of these processes its dependent on the mass of NIPAAm and rotational speed. The equations for these energy inputs are shown below, where $c_{p,w}$ is the specific heat of liquid water and the evaporation fraction is the fraction of liquid water that evaporates during the draining process.

$$\dot{Q}_{NIPAAm,1} = \frac{m_{NIPAAm}}{360}\omega\left(c_{p,NIPAAm} + \Delta C_{NIPAAm,max}c_{p,w}\right)\left(LCST - T_{NIPAAm,f}\right) \tag{2.17}$$

$$\dot{Q}_{NIPAAm,2} = \frac{m_{NIPAAm}}{360}\omega(evaporation\ fraction)h_{fg} \tag{2.18}$$

$$\dot{Q}_{NIPAAm,3} = \frac{m_{NIPAAm}}{360}\omega\left(c_{p,NIPAAm} + \Delta C_{NIPAAm,min}c_{p,w}\right)\left(LCST - T_{NIPAAm,initial}\right) \tag{2.19}$$

The sensible heating contains the thermal mass of liquid water because the heaters must increase the temperature of the NIPAAm and the water contained within the sorbent, $\Delta C_{NIPAAm,max}$, but after this some of the water drains, so the sensible cooling needs to decrease the temperature of NIPAAm and whatever water remains within the NIPAAm after regeneration, $\Delta C_{NIPAAm,min}$. The regeneration energy for NIPAAm, expressed as required electrical input, is shown in the equation below, where *COP_{cool}* is the COP of the system that cools the NIPAAm in the lower left portion of the NIPAAm wheel (i.e.: the vapor compression cooling system).

$$\dot{E}_{regen,NIPAAm} = \dot{Q}_{NIPAAm,1} + \dot{Q}_{NIPAAm,2} + \frac{\dot{Q}_{NIPAAm,3}}{COP_{cool}} \tag{2.20}$$

The absorption capacity of NIPAAm is dependent on the inlet temperature and humidity of the supply air, as shown in the following figure.

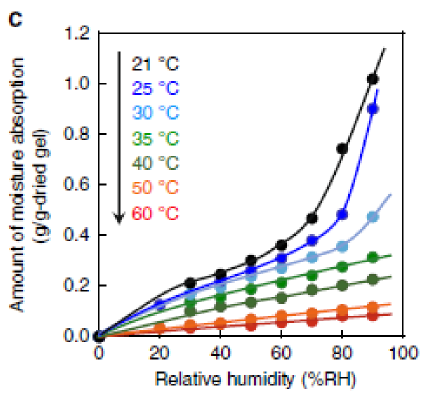


Fig. 2.2.3.3: NIPAAm moisture absorption capacity for various temperature and relative humidity values [7]

The data points were taken from these curves to form an array of NIPAAm absorption capacity for varying air temperature and humidity. The model uses the temperature and relative humidity of the air entering the dehumidifier to determine the absorption capacity of the NIPAAm for the given inlet conditions. The absorption capacity is the maximum uptake in water, relative to the mass of NIPAAm, that the NIPAAm can sustain before it becomes saturated. NIPAAm, however, cannot feasibly be dried out completely, as shown in the figure below.

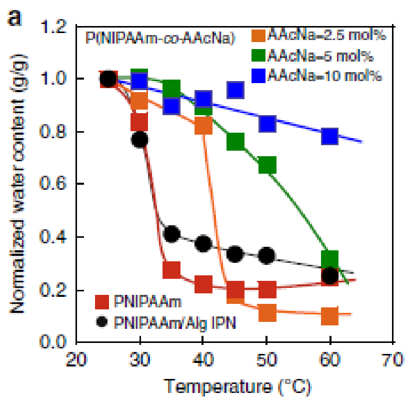


Fig. 2.2.3.4: Normalized water content as a function of temperature for various NIPAAm gels [7]

The black curve in the figure above is associated with the PNIPAAm/Alg IPN, which is the NIPAAm configuration that was considered for the models in this paper. The curve shows the normalized water content, which is the water absorbed over maximum absorption capacity. From the figure, it can be seen that the NIPAAm does not completely dry out from regeneration. After regeneration at 32 °C, the NIPAAm reaches approximately 40% of its absorption capacity, while at 60 °C it is still at approximately 30% of it's absorption capacity. Thus, the NIPAAm was modeled under the assumption that it could only drain 60% of its water content at a regeneration temperature of 32 °C. This means that NIPAAm entering the dehumidification portion of the wheel still has water content equal to 40% of its maximum capacity. The increase in relative water content across the dehumidification portion is then 60% of the absorption capacity, $\Delta C_{NIPAAm,max}$, as described in the equation below.

$$\Delta C_{NIPAAm} = 0.6 * \Delta C_{NIPAAm,max} \tag{2.21}$$

The water retained by the NIPAAm after regeneration, $\Delta C_{NIPAAm,min}$, is then 40% of the maximum water content.

Aside from only being able to remove 60% of the total water content within the NIPAAm, some percentage of the water that is removed will be evaporated. While it is desirable to drain and reclaim all of the regenerated water as a liquid, it is inevitable that some of the water will evaporate during the draining process. The figure below is from a study on the performance of a NIPAAm IPN gel, and it shows the ratio of water collected in liquid form during regeneration to the total water content before regeneration.

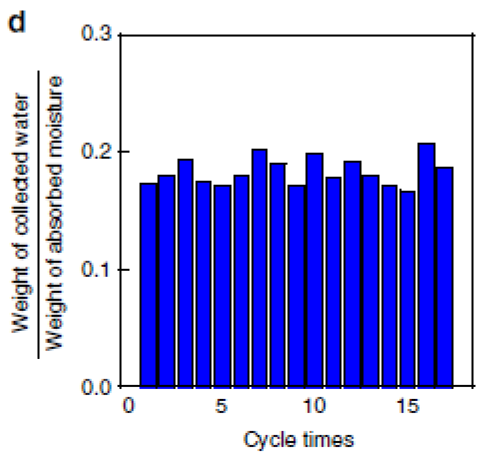


Fig. 2.2.3.5: Ratio of liquid water collected during regeneration to total water absorbed by the NIPAAm [7]

The figure above shows that approximately 20% of the total water content is reclaimed as liquid water during regeneration. When considering that only 60% of the water content is removed during regeneration, this means that 20% of the water content after dehumidification is reclaimed as liquid water, 40% is evaporated during regeneration, and 40% stays absorbed within the NIPAAm. The model was written to reflect these characteristics; however, it should be noted that the 2:1 mass ratio of evaporated water to reclaimed liquid water that was seen during regeneration was for regeneration at 50 °C [7]; thus, at lower regeneration temperatures, that ratio should decrease.

The electrical power required to regenerate the NIPAAm wheel can be used in Eq. 2.15 to find the COP for the NIPAAm system. It is assumed that the liquid water expelled from the wheel is simply drained, much like the water that is condensed on the evaporator coils in the vapor compression only case. Because the evaporation fraction and the temperatures $T_{NIPAAm,i}$ and $T_{NIPAAm,f}$ in Eq. 2.20 are not obvious for the general system configuration that was modeled, the model was created such that the user can vary these values in the graphical interface and observe the resulting regeneration energy and COP. Based on Fig. 2.2.3.4 above, it appears that the regeneration process begins around 25 °C, so for the cases described in Chapter 3, the value of $T_{NIPAAm,f}$ was set to 25 °C.

2.2.4 Desiccant Dehumidification and Evaporative Cooling

In the first three configurations, a vapor compression air conditioner was implemented for the cooling portion of the cycle. In the following two configurations, the vapor compression air conditioner is replaced with an evaporative cooler, which is commonly used in desiccant air conditioning systems. The fourth configuration is a "desiccant dehumidification and evaporative cooling" system, which, as shown in the figure below, has the same components as the desiccant dehumidification and vapor compression cooling configuration, except for the cooling unit.

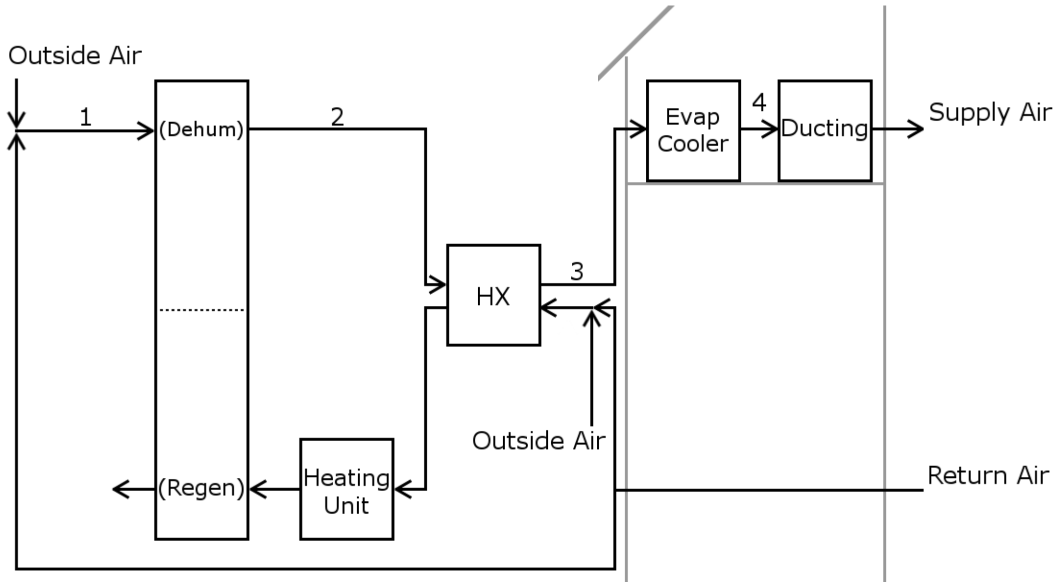


Fig. 2.2.4.1: Desiccant dehumidification and evaporative cooling schematic

To achieve the desired outlet supply air conditions, the supply air is dehumidified past the desired outlet humidity, thus over-heating and over-drying the air. The supply air is then pre-cooled at the heat exchanger, after which the evaporative cooler increases the humidity and further cools the air to the desired outlet humidity ratio and temperature. The general process of the supply air for this case is shown on the psychrometric chart in the figure below.

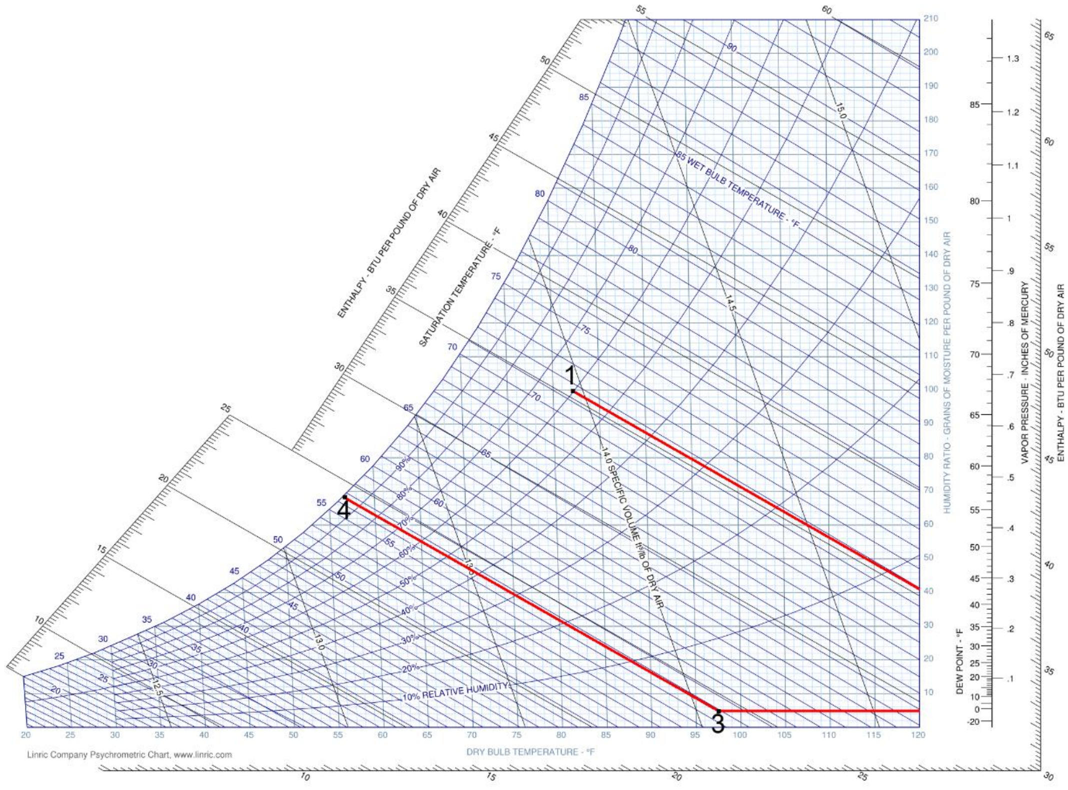


Fig. 2.2.4.2: Psychrometric chart for supply air in a desiccant dehumidification and evaporative cooling system

The desiccant dehumidifier and evaporative cooler configuration is representative of a traditional desiccant air conditioning system. In this configuration, water is consumed at the evaporative cooler, and the water that is absorbed by the desiccant wheel is eventually exhausted to the outside as vapor during regeneration. Thus, this type of system has a net consumption of water.

2.2.5 NIPAAm Dehumidification and Evaporative Cooling

The fifth and final configuration described in this report is the "NIPAAm dehumidification and evaporative cooling" configuration. As with the previous configuration, it has the same components as its vapor compression counterpart, with the exception of the cooling unit. The general process of the supply air for this configuration is the same as the previous one. The figure below shows the system schematic.

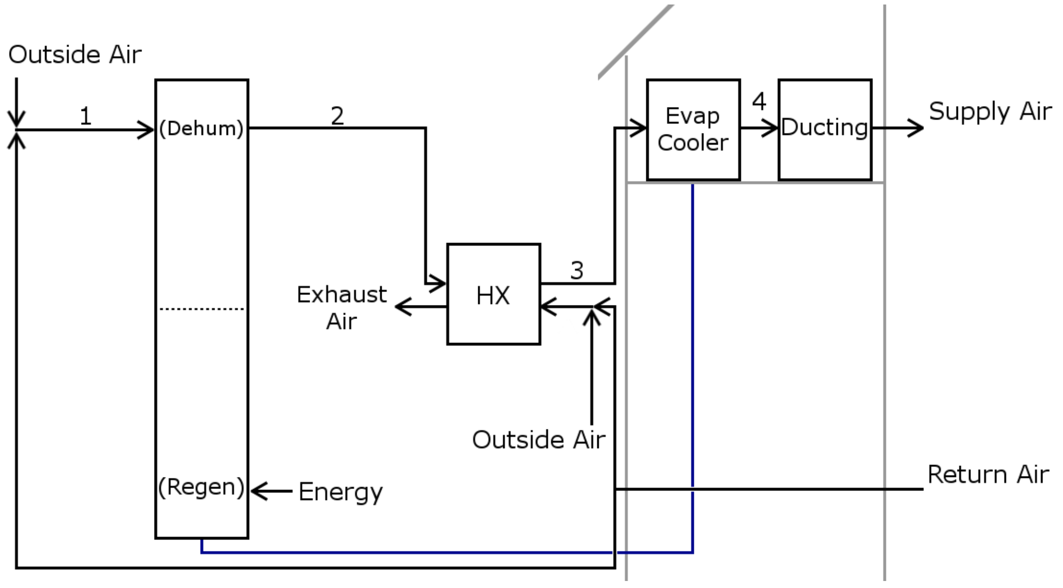


Fig. 2.2.5.1: NIPAAm dehumidification and evaporative cooling schematic

The vapor compression cooler uses electricity to implement a refrigeration cycle and produce cooling. The evaporative cooler in this system, however, uses the latent heat of evaporation to produce cooling. Thus, when blowers and other smaller electrical components are neglected, the evaporative cooler produces cooling without consuming electricity. In the other NIPAAm-based configuration, which utilizes a vapor compression cooling system, the power required for regeneration was described in Eq. 2.20 and includes the electricity required to produce cooling. In this configuration, however, there is no electricity required for cooling, so the term drops to zero.

Aside from the cooling unit, the only difference between this system and its vapor compression counterpart is the regeneration water. In the NIPAAm dehumidification and vapor compression configuration, it was assumed that the water expelled from the regenerating NIPAAm would be drained, due to a lack of obvious use. However, in this configuration, the evaporative cooler consumes liquid water, while the regenerating portion of the NIPAAm wheel expels liquid water. Thus, the blue line in Fig. 2.8 represents the expelled liquid water being sent to the evaporative cooler. Aside from the lower heat of regeneration, this reclamation of water during regeneration could serve as another benefit of using NIPAAm instead of a traditional desiccant.

2.3 Sub-system and Component Models

Most of the components were modeled in individual Python scripts as functions, with top level scripts organizing the variables and passing information from one component function to the next. Some components with simpler processes were modeled in the top level scripts, as a separate function was not needed. The vapor compression sub-system consists of a script for the evaporator coils, the condenser coils, and several top level scripts to pass information.

Regarding the dehumidifier model, the rate of moisture removed from the air is equivalent to the rate of absorption by the dehumidifying wheel. While the maximum capacity of moisture absorption is determined by the absorbent and the inlet air properties, the dynamic rate of absorption is dependent on the material, air temperature and relative humidity, instantaneous concentration of water within the absorbent, system geometry, as well as various flow characteristics [8]. Because the models described in this paper are intended for system-level analysis and do not incorporate detailed information about the components, some of these parameters are not known, thus making it difficult to accurately model a dynamic dehumidification response from the dehumidifying wheel. Because of this, and because the air stream properties were already chosen to be modeled as steady, it was decided that the components would be modeled for steady state performance. This means that the dehumidifying wheel model uses an energy balance to determine the outlet temperature for the required humidity drop, while the models for the vapor compression air conditioner determine the required refrigerant state points, and thus input power, to achieve the required outlet temperature.

2.3.1 Vapor Compression Evaporator

To model for the vapor compression evaporator requires several inputs: the temperature and humidity of the air just before it passes over the evaporator and the desired temperature and humidity of the air after it is cooled by the evaporator. The inlet air temperature and humidity are known, and the desired outlet temperature is known, so the AC script determines if the desired outlet temperature is lower than the saturation temperature for the given inlet air temperature and humidity. If the desired outlet temperature is not lower than the saturation temperature, then no condensation occurs, and the outlet humidity is the same as the inlet humidity. If the desired outlet temperature is lower than saturation, then the outlet humidity is the saturation humidity ratio for the outlet temperature. This is illustrated in the graphic below.

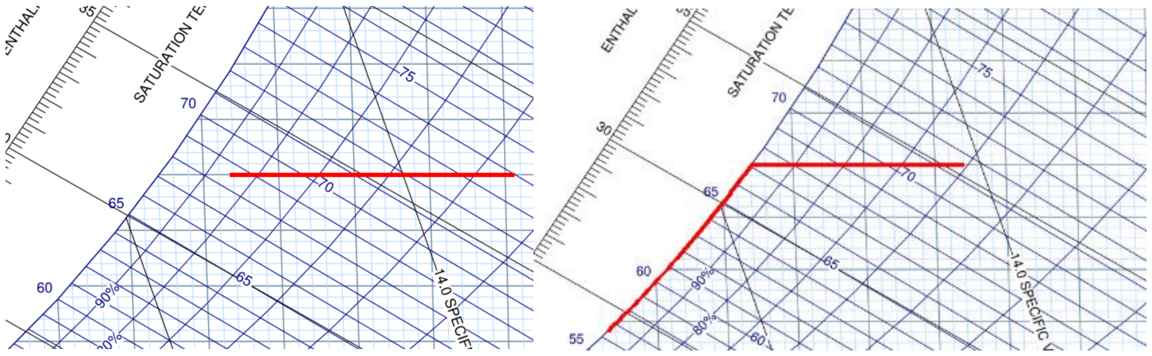


Fig. 2.3.1.1: Determining the outlet humidity ratio. Illustrated on the left is the case when the desired outlet temperature is higher than the saturation temperature, and the humidity ratio is constant. On the right is the process if the desired outlet temperature is lower than saturation, at which point the process follows the saturation curve and the outlet air is saturated at the desired outlet temperature.

For all of the cases that were modeled in this report, some dehumidification took place, meaning all of the processes took the general form of the process shown on the right in the figure above. The evaporator is modeled as a heat exchanger. The purpose of the evaporator model is to determine the evaporator temperature that will bring the inlet air to the desired outlet temperature.

First, the script determines the rate of heat transfer necessary to achieve the desired outlet conditions. When no dehumidification occurs, the necessary rate of heat transfer is described in the equation below, where $T_{VC,i}$ is the temperature of the air before it is cooled (i.e.: the supply inlet air for the vapor compression only case, or the air leaving the heat exchanger for the dehumidification and vapor compression cooling cases), and $T_{VC,o}$ is the temperature of the air after it is cooled (i.e.: the supply outlet temperature, also denoted as $T_{s,o}$).

$$\dot{Q}_{evaporator} = \dot{m}_{supply}\left(c_{p,a} + x * c_{p,v}\right)\left(T_{VC,i} - T_{VC,o}\right) \tag{2.22}$$

When dehumidification occurs across the evaporator coils, the required heat transfer rate becomes the following, where the first term is the heat transfer required to bring the air to saturation, the second term is sensible heat transfer that occurs during dehumidification (i.e.: the cooling that causes the air temperature to drop along the saturation curve), and the third term is the latent portion of the heat transfer that occurs during dehumidification (i.e.: the cooling that causes the moisture to condense), where T_{sat} is the saturation temperature for the inlet humidity ratio.

$$\dot{Q}_{evaporator} = \dot{Q}_{e,1} + \dot{Q}_{e,2} + \dot{Q}_{e,3} \tag{2.23}$$

$$\dot{Q}_{e,1} = \dot{m}_{supply}\left(c_{p,a} + x_{VC,i} * c_{p,v}\right)\left(T_{VC,i} - T_{sat}\right) \tag{2.24}$$

$$\dot{Q}_{e,2} = \dot{m}_{supply}\left(c_{p,a} + \frac{x_{VC,i} + x_{VC,o}}{2} * c_{p,v}\right)\left(T_{sat} - T_{VC,o}\right) \tag{2.25}$$

$$\dot{Q}_{e,3} = \dot{m}_{supply}\left(x_{VC,i} - x_{VC,o}\right)h_{fg} \tag{2.26}$$

Of the terms above, $\dot{Q}_{e,1}$ is the rate of heat transfer that occurs across the dry portion of the heat exchanger (before condensation occurs), while $\dot{Q}_{e,2}$ + $\dot{Q}_{e,3}$ is the rate of heat transfer that results across the portion where condensation occurs. The equations above must be satisfied for the air to reach the desired outlet properties, while the equations below describe the actual heat exchanger performance for some evaporator temperature, $T_{ref,evaporator}$.

$$UA_{evaporator} = UA_{dry} + UA_{wet}$$

$$\dot{Q}_{e,1} = UA_{dry} \frac{T_{VC,i} - T_{sat}}{\ln\left(\frac{T_{VC,i} - T_{ref,evaporator}}{T_{sat} - T_{ref,evaporator}}\right)} \tag{2.27}$$

$$\dot{Q}_{e,2} + \dot{Q}_{e,3} = UA_{wet} \frac{T_{sat} - T_{VC,o}}{\ln\left(\frac{T_{sat} - T_{ref,evaporator}}{T_{VC,o} - T_{ref,evaporator}}\right)} \tag{2.28}$$

The model uses an iterative scheme to find the evaporator temperature that satisfies the two sets of equations for rate of heat transfer across the evaporator.

It is assumed that the thermal conductivity of the evaporator coils is high, such that the temperature of the refrigerant is the same as the temperature at the exterior of the coils. Thus, the temperature of the refrigerant flowing through the evaporator is now known. It is also assumed that the vapor compression system regulates the mass flow rate of the refrigerant such that the refrigerant leaving the evaporator is saturated vapor, meaning the quality at this point is known. From the known temperature and quality, the specific enthalpy and specific entropy of the refrigerant leaving the evaporator is known. Additionally, since the refrigerant undergoes phase change in the evaporator, the pressure is the saturation pressure at the known evaporator temperature.

2.3.2 Vapor Compression Condenser and Expansion Valve

The vapor compression condenser was modeled as a heat exchanger, much like the evaporator. The evaporator model was implemented to determine the required evaporator temperature, as well as the resulting refrigerant pressure and the enthalpy of the refrigerant as it leaves the evaporator. Similarly, the condenser model was implemented to find the condenser temperature required to complete the cycle, based on the outside air temperature. Because the refrigerant in the evaporator is undergoing phase change, the evaporator effectiveness is maximized for the given heat transfer coefficient and heat capacity rate of the incoming air. The equation for heat exchanger effectiveness, ε, under this condition is given in the equation below, where UA_{cond} is the heat transfer coefficient for the condenser.

$$\varepsilon = 1 - \exp\left(-\frac{UA_{cond}}{\dot{m}_{air,cond} * \left(c_{p,a} + x_{outside} c_{p,v}\right)}\right) \tag{2.29}$$

It is assumed that the refrigerant leaving the condenser must be saturated liquid, so the condenser temperature must be high enough such that the heat exchanger facilitates the necessary amount of heat transfer to bring the refrigerant to saturated liquid at the condenser exit. An iterative solver was implemented to find the temperature that satisfies several conditions. First, the rate of heat transfer at the heat exchanger must equal the rate of heat transfer required to bring the refrigerant from the specific enthalpy as it enters the condenser, $h_{ref,3}$, to the specific enthalpy as it leaves the condenser, $h_{ref,4}$.

$$\varepsilon * \dot{m}_{air,cond} * c_p\left(T_{ref,condenser} - T_{outside}\right) = \dot{m}_{ref}\left(h_{ref,3} - h_{ref,4}\right) \tag{2.30}$$

The specific enthalpy of the refrigerant as it enters the condenser must also satisfy the following equation describing the compressor with some isentropic efficiency, where

$h_{ref,3,isen}$ is the refrigerant specific enthalpy at the same pressure as $h_{ref,3}$, but with the same entropy as $h_{ref,2}$.

$$\eta_{isen} = \frac{h_{ref,3,isen} - h_{ref,2}}{h_{ref,3} - h_{ref,2}} \tag{2.31}$$

Additionally, the pressure is assumed to be constant throughout the condenser. The specific enthalpy of the refrigerant leaving the condenser can then be found from the pressure of the refrigerant entering the condenser and a quality of zero. Because the expansion valve is assumed to be isenthalpic, the specific enthalpy of the refrigerant leaving the condenser must also be the specific enthalpy of the refrigerant as it enters the evaporator. Now that the specific enthalpies of the refrigerant at the beginning and end of the evaporator are known, and the required rate of heat transfer at the evaporator is known as well, the mass flow rate of the refrigerant through the system can be found, as described in the equation below.

$$\dot{m}_{ref} = \frac{\dot{Q}_{evaporator}}{h_{ref,1} - h_{ref,2}} \tag{2.32}$$

The iterative scheme starts with a temperature just above the outside ambient and evaluates the specific enthalpy of the refrigerant as it enters the condenser, using Eq. 2.31. It then computes the rest of the values and finds the error produced by Eq. 2.30. The iterations continue until the error is sufficiently small, at which point the resulting condenser values are found. At this point, all of the state in the vapor compression cycle are defined, and all information regarding the refrigerant is known.

2.3.3 Air Conditioning Compressor

The models for the evaporator and condenser are used to find the specific enthalpy of the refrigerant leaving the evaporator and the specific enthalpy of the refrigerant entering the condenser. These values are used to find the power used by the compressor the raise the refrigerant pressure and temperature, as described previously in Eq. 2.8.

2.3.4 Dehumidifier

It is assumed that the dehumidifying wheel operates at steady state. Desiccant dehumidification is close to an isenthalpic procedure [4], meaning all latent heat is converted to sensible heat, and the enthalpy of the moist air before dehumidification is equal to the enthalpy of the drier, hot air after the process. While simple models account only for the latent heat [4], there is some binding energy that must also be released as heat during sorption [6]. However, this binding energy is often small compared to the latent heat (less than 25% of the total heat released for a certain example of desiccation [6]). To maintain the simplicity of the model, and because the heat of adsorption for NIPAAm must be found experimentally, the dehumidification process is modeled as isenthalpic, and only the latent heat is considered. Additionally, it is assumed that all of the heat produced during dehumidification is transferred to the air, and the sorbent material is assumed to stay at a constant temperature. In reality, some of the heat will be transferred to the sorbent, and the temperature of the sorbent will change with time. The amount of heat that is transferred to the sorbent is based on the air properties, geometric properties, specific heat of the sorbent, and the instantaneous adsorption uptake of the sorbent [9]. As with the dynamic rate of sorption, the dynamic temperature change of the sorbent material is not considered in this model, as many of the necessary parameters are not known. For traditional desiccants, the temperature of the desiccant material could nominally affect the dehumidification performance; however, for NIPAAm, the temperature of the NIPAAm could greatly affect performance. If the NIPAAm temperature reaches the LCST during dehumidification, the NIPAAm will stop dehumidifying the air, as it will have transitioned from hydrophilic to hydrophobic. Thus, the dehumidification model assumes that the NIPAAm temperature does not exceed the LCST during operation. In Chapter 3, a discussion is presented on potential alternate configurations for the case where it is found that the NIPAAm temperature reaches the LCST before the conditioning process is complete.

The dehumidifier model is given an input for the known inlet air temperature and humidity ratio, as well as the desired outlet humidity ratio, and it uses an iterative solver to find the outlet temperature that satisfies the following equation describing the isenthalpic process, where the specific enthalpies are found through CoolProp with temperature as an input.

$$h_{a,i} + x_i h_{v,i} = h_{a,o} + x_o h_{v,o} \tag{2.33}$$

2.3.5 Heat Exchanger

The heat exchanger downstream of the dehumidifier was modeled to determine the temperature drop of the supply air and the temperature rise of the process air. The model for this heat exchanger was written to assume a constant effectiveness of 0.99, as the effectiveness approaches unity when the heat exchanger becomes sufficiently large, and certain desiccant air conditioning cycles were described in literature with heat exchanger effectiveness values close to unity [4]. The heat exchanger model uses the following equation to find the rate of heat transfer between the supply and process air streams, where C_{min} is the lesser of the two heat capacity rates, as determined in the script, $T_{s,HX,i}$ is the supply air temperature as it enters the heat exchanger, and $T_{p,HX,i}$ is the process air temperature as it enters the heat exchanger.

$$\dot{Q}_{HX} = \varepsilon * C_{min}\left(T_{s,HX,i} - T_{p,HX,i}\right) \tag{2.34}$$

After the rate of heat transfer is calculated, the script calculates the supply and process air outlet temperatures as shown in the equations below, where $T_{s,HX,o}$ is the supply air temperature as it leaves the heat exchanger, and $T_{p,HX,o}$ is the process air temperature as it leaves the heat exchanger.

$$T_{s,HX,o} = T_{s,HX,i} - \frac{\dot{Q}_{HX}}{\dot{m}_{supply}(c_{p,a} + x_{s,i}c_{p,v})} \tag{2.35}$$

$$T_{p,HX,o} = T_{p,HX,i} + \frac{\dot{Q}_{HX}}{\dot{m}_{process}(c_{p,a} + x_{p,i}c_{p,v})} \tag{2.36}$$

2.3.6 Regeneration

As previously mentioned, the regeneration temperature of the desiccant wheel is not directly obvious and varies on a case-to-case basis. Thus, the model for the heat of regeneration was implemented such that it requires the regeneration temperature as an input. This allows the user to input the regeneration temperature for a known scenario, or it allows the user to vary the regeneration temperature when it is unknown, such that the performance of the desiccant systems can be determined for a range of possible regeneration temperatures.

Regeneration of the NIPAAm dehumidifier requires three inputs: the NIPAAm stop temperature, the percent evaporation, and the NIPAAm start temperature, as seen in Eq. 2.17 through 2.19. The NIPAAm stop temperature is known to be 25 °C, as this is the temperature of NIPAAm before which dehumidification will occur. The NIPAAm start temperature (or the temperature as it enters the dehumidification section of the wheel) and the percent evaporation are less obvious. The model user was given the ability to vary these parameters, such that the system performance could be observed for various values.

2.3.7 Evaporative Cooler

Like the dehumidification process, the evaporative cooling was modeled as an isenthalpic process, which means the enthalpy of the air leaving the heat exchanger should equal the enthalpy of the air leaving the cooler. Thus, the script performs an iterative process to determine the conditions under which this is satisfied. The process of the supply air before it enters the evaporative cooler is as follows: the supply air enters the dehumidifier, and when it leaves the temperature is higher and the humidity ratio is lower; the air is then pre-cooled by the heat exchanger and the temperature, and thus enthalpy, drops. The only parameter that can be varied in this process is the humidity ratio of the air leaving the dehumidifier; thus, during the iterative process, the humidity ratio drop across the dehumidifier is varied until the value is found that results in the required enthalpy to satisfy the desired outlet conditions. This iterative process is illustrated in the figure below.

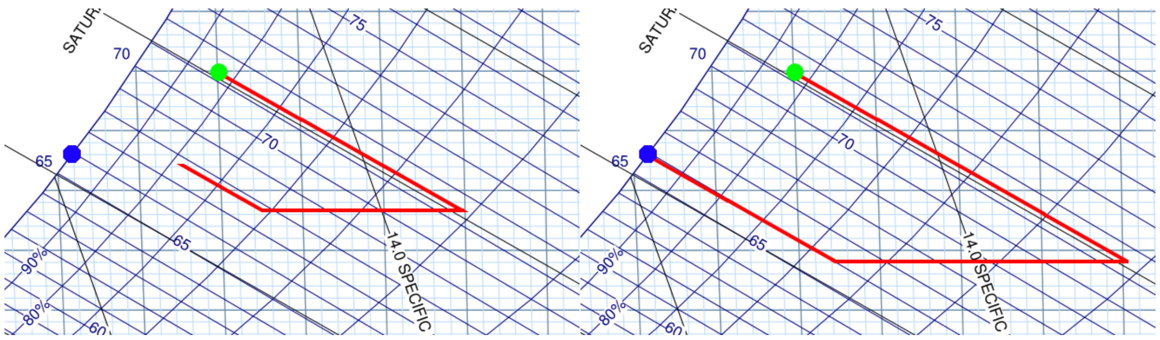


Fig. 2.3.7.1: Psychrometric charts of the supply air process, with a process that could not meet the desired output (left) and a process that could (right). The points in green are the inlet states, and the points in blue are the desired outlet states.

2.4 Conditioned Space Transient Models

When first constructing the model, it was unclear if the dynamic performance of the system components could be evaluated, such as a dynamic rate of absorption at the dehumidifier wheel. It was eventually determined that, for the purpose of highlighting the potential of a general NIPAAm system, it did not make sense to create a complicated numerical model based on a specific system design. Thus, simple steady state models were implemented. However, before it was decided that steady operation would be modeled, a transient model for the temperature and humidity of the air within the house was created. The transient model for the house air is still used within the graphical interface, but currently it is entirely cosmetic. It shows how the average temperature and humidity within the space drop over time, as well as the time it takes the air within the house to reach the desired cool temperature, but this has no bearing on the steady state performance of the systems. Thus, the transient model serves no current purpose and has no effect on the results presented in the following chapter. To keep the methods section succinct, an explanation of the transient model is not presented in this chapter. However, a lengthy explanation of the transient model is presented in Appendix A, along with a discussion of the graphical interface setup and all of the python scripts used.

2.5 Graphical User Interface

As previously mentioned, a graphical user interface was created to allow a user to select which system configurations to model, input the required temperature and humidity values, and observe the results for the various system configurations. A detailed explanation of the graphical user interface is provided in Appendix A, along with an explanation of all of the Python scripts used to model the system and create the graphical interface. The appendices after Appendix A contain the code used to create the Python scripts.

2.6 Selection of Various Values

Many of the values used in the system models were defined as constants, such as absorption capacity of the sorbents, evaporator and condenser heat transfer coefficients, and the maximum desiccant regeneration temperature. The following sections describe how these values were selected for all of the models.

2.6.1 Selection of NIPAAm Regeneration Temperature

For most cases, the regeneration temperature of NIPAAm was defined as 305.15 K (32 °C); however, the model was run for once case with the regeneration temperature as

323.15 K (50 °C). This was done based on experiments in which the NIPAAm was regenerated at 50 °C (assumedly to ensure a sufficient regeneration time) [7]. The minimum temperature for the NIPAAm “Start” and “Stop” temperature sliders was set to 20 °C, as it was assumed that the NIPAAm would not need to be cooled much further than that in order to sustain absorption for the entire dehumidification portion of the wheel. If necessary, however, these constants, as well as all others mentioned, can be easily changed.

2.6.2 Selection of Other Values Regarding the Various Air Conditioning System Configurations

The mass flow rate of supply air was selected to be 0.7 kg/s of dry air. This was calculated from a rule-of-thumb that states the average air flow provided for a certain amount of cooling is 400 cfm/ton [10], so with a cooling power of 3 tons, the volumetric flow rate was set as 1200 cfm, which corresponds to approximately 0.7 kg/s of air. The supply air flow consists of a mix of return air from the space and outside air. For the cases analyzed, the ratio of return air to outside air was varied. While the rule-of-thumb for the air flow rate over the evaporator was given as 400 cfm/ton, the flow rate used to cool the condenser was set as double the flow rate over the evaporator, as the condenser is larger, and the process needs a greater rate of heat transfer. Thus, the mass flow rate of outside air over the condenser coils was set to be 1.4 kg/s.

Silica gel is the material that was chosen for the desiccant wheel that was modeled. From literature, it was found that silica gel has a moisture absorption capacity of approximately 0.38 kg of water per kg of silica gel [6]. The specific heat of silica gel was reported to be in the range of 0.92 and 1.00 kJ/kg K [6], so a mean value of 0.96 kJ/kg K was chosen. This was also chosen as the specific heat of NIPAAm, as a value for the specific heat of the NIPAAm IPN was not found in literature; however, if a more accurate value is found, the specific heat can easily be changed within the Python scripts by changing the value of the variable “c_p_NIPAAm”.

A rotational speed of 0.75 deg/s was selected for the desiccant and NIPAAm wheels. Based on literature, it was found that most desiccant wheels operate between 5 and 10 revolutions per hour, in order to maximize performance [11]. Thus, a mean value of 7.5 revolution per hour was selected, which is equivalent to 0.75 deg/s. To analyze a particular desiccant wheel with a known speed, the value of the variable “omega” can easily be changed within the Python scripts.

2.6.3 Selection of Values for the Vapor Compression System

The air pressure for both inside and outside of the house was selected as 101325 Pa (standard pressure). A value of 80% was chosen for the isentropic efficiency for the compressor of the vapor compression cycle. The only other values to be defined in this script were the evaporator and condenser heat transfer coefficients. Based on minimum air conditioning efficiency standards set by the DOE in 2015, a 3 ton air conditioner can have an energy efficiency ratio (EER) of no less than 11.7 for the southwest region [12], which corresponds to a COP of 3.43. The efficiency of an air conditioner is determined under standard indoor and outdoor conditions specified by ASHRAE, which state that the system is to be tested with an indoor dry bulb temperature of 80 °F and a wet bulb temperature of

67 °F, while the outdoor air must be at a dry bulb temperature of 95 °F and a wet bulb temperature of 75 °F [13]. Because it was difficult to find heat transfer coefficient values for an average air conditioning evaporator or condenser, the values were determined iteratively through the use of the model. The vapor compression model was run with the air properties described in the ASHRAE standard rating conditions, as well as an outlet temperature of 53 °F, and the heat transfer coefficients were varied until the COP of the air conditioning system reached the minimum allowable value of 3.43. Because the condenser is traditionally larger than the evaporator, it was decided that the heat transfer coefficient for the condenser would be double the value for the evaporator. Eventually, it was found that an evaporator heat transfer coefficient of 1810 W/K and a condenser heat transfer coefficient of 3620 W/K produced a COP of 3.43, and these were the values that were selected. While these values are somewhat arbitrary, they produce a realistic COP for the AC and in turn produce a relatively realistic model for the AC system. It should be noted that the COP of the vapor compression AC system is not the same as the COP previously mentioned; the COP of the vapor compression system is shown in the equation below and only depends on the specific enthalpy values of the refrigerant.

$$COP_{AC} = \frac{h_{ref,2} - h_{ref,1}}{h_{ref,3} - h_{ref,2}} \tag{2.40}$$

The ASHRAE test conditions do not specify a required outlet temperature that the air conditioning system must achieve. The aforementioned outlet temperature of 53 °F was selected through the following procedure. One ton of refrigeration is equivalent to 3.5 kW, and the rule-of-thumb for selecting the flow rate of air over air conditioning evaporator coils is 400 cfm/ton [10]. The volumetric flow rate of 400 cfm/ton can be converted to a mass flow rate, and the cooling of 3.5 kW must equal the product of mass flow rate, specific heat, and temperature drop. The mass flow rate and specific heat of humid air are known, resulting in a temperature drop of 27 °F. Thus, from the definition of a ton of refrigeration, the outlet air was determined to be 53 °F for an air conditioning system under the ASHRAE standard conditions that follows the 400 cfm/ton air flow rate guideline.

3 RESULTS AND DISCUSSION

3.1 List of Cases Analyzed

To analyze the performance of the NIPAAm dehumidification configurations, several cases were run for each of the five configurations, and the COP values of the different configurations were compared. For each case, the following values were defined: the properties of the air within the conditioned space (return air), system outlet air properties (the conditioned supply air), outside air properties, percent of the supply air flow that comes from return air, percent of the supply air flow that comes from outside air, percent of the process air flow that comes from return air, and percent of the process air flow that comes from outside air. Thus, each case represents a scenario of air, with certain properties, entering the overall system and being conditioned to a cooler, drier state. The models then determine the sub-processes and energy input required for each configuration to achieve the same overall process for a given case. Four cases were analyzed, and the description of each case can be found in Table 3.1 below. The outlet air properties are the properties of the air as it leaves the final component of the system (i.e.: as it leaves the evaporator coils for the vapor compression cooling configurations or as it leaves the evaporative cooler for the other two configurations). The supply air stream is comprised of a mix of return and outside air, so the supply stream percent return air and the supply stream percent outside air sums to 100% for all cases. The process air stream is also comprised of a mix of return and outside air, so the two percentages should sum to 100% for the process air as well.

Table 3.1.1: Description of cases analyzed

	Case 1: Pennington Cycle	Case 2: Recirculation Cycle	Case 3: Hybrid Cycle	Case 4: Supermarket
Return air temperature	22.50 °C	27.00 °C	22.00 °C	10.00 °C
Return air relative humidity	76%	52%	50%	92%
Outlet air temperature	13.27 °C	9.74 °C	11.00 °C	3.90 °C
Outlet air humidity ratio	9.50 g/kg	7.50 g/kg	8.16 g/kg	5.00 g/kg
Outside air temperature	35.00 °C	27.00 °C	25.00 °C	25.00 °C
Outside air relative humidity	40%	78%	100%	75%
Supply stream return air %	0	100	50	64
Supply stream outside air %	100	0	50	36
Process stream return air %	100	0	50	36
Process stream outside air %	0	100	50	64

Following the analysis of the cases described in the previous table, the third case was revisited, and the NIPAAm configurations were analyzed for a regeneration temperature of 50 °C, instead of the LCST of 32 °C. This analysis was performed to see how much the performance would be affected if the NIPAAm was required to be regenerated at a temperature significantly greater than the LCST, as indicated in certain literature [7]. Additionally, a parametric study was performed to observe the effect of changing the NIPAAm dehumidification start temperature on the system performance.

3.2 Case 1: The Pennington Cycle

A very common air conditioning cycle associated with desiccant air conditioning is the Pennington cycle, as it was the first cycle introduced for rotary desiccant air conditioning [4]; a generalized version of the Pennington cycle was the first case that was modeled. In the Pennington cycle, outside air is conditioned and provided to the building, while return air is used as the process air. This cycle is illustrated in the figure below.

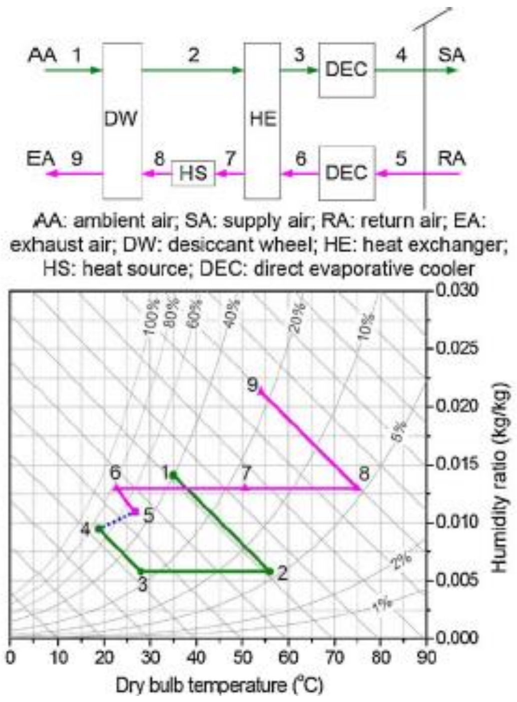

AA 1
2
3
DEC
4
SA
DW
HE
EA 9
8
HS
7
6
DEC
5
RA
AA: ambient air; SA: supply air; RA: return air; EA: exhaust air; DW: desiccant wheel; HE: heat exchanger; HS: heat source; DEC: direct evaporative cooler
100%
80%
60%
40%
20%
10%
5%
2%
1%
Humidity ratio (kg/kg)
Dry bulb temperature (°C)

Fig. 3.2.1: Pennington cycle schematic (top) and psychrometric process (bottom) [4]

The figure above, taken from literature, uses an evaporative cooler at the beginning of the process air stream, while the systems modeled in this report did not incorporate an evaporative cooler at this point in the cycle. Other than that, the overall process of the Pennington cycle was modeled using the conditions listed in Table 3.1 under Case 1. For this case, the supply air consisted entirely of outside air, and the process air consisted entirely of return air. Based on Fig. 3.1 above, the regeneration temperature for the desiccant was set to 75 °C. The NIPAAm configurations were modeled for evaporation percentages of 0%, 17%, 33%, 50%, 67%, 83%, and 100%. As previously mentioned, the NIPAAm temperature must remain sufficiently below the LCST during the dehumidification section to allow for continuous absorption. Based on Fig. 2.2.3.4, the NIPAAm was set to have a temperature of 25 °C at the end of dehumidification. The temperature of the NIPAAm at the beginning of dehumidification that results in the temperature at the end of dehumidification being 25 °C is unknown, and it was set somewhat arbitrarily to 20 °C. The results from this case are presented in the tables below; Table 3.2 contains the values for the vapor compression only, desiccant dehumidification and vapor compression cooling, and desiccant dehumidification and evaporative cooling configurations, Table 3.3 contains the values for NIPAAm dehumidification and vapor compression cooling, and Table 3.4 contains the values for NIPAAm dehumidification and evaporative cooling.

Table 3.2.1: Case 1 results for vapor compression and desiccant systems

	Vapor Compression Only	Desiccant + Vapor Compression Cooling	Desiccant + Evaporative Cooling
Vapor compression compressor power	7.23 kW	0.99 kW	0.00 kW
Rate of regeneration energy	0.00 kW	20.99 kW	14.21 kW
Total electrical power required	7.23 kW	21.98 kW	14.21 kW
COP	1.78	0.58	0.9

Table 3.2.2: Case 1 results for NIPAAm dehumidification and vapor compression cooling

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
Vapor compression compressor power	0.99 kW	0.99 kW	0.99 kW	0.99 kW	0.99 kW	0.99 kW	0.99 kW
Rate of regeneration energy	0.45 kW	1.74 kW	2.97 kW	4.27 kW	5.56 kW	6.79 kW	8.08 kW
Total electrical power required	1.44 kW	2.73 kW	3.96 kW	5.26 kW	6.55 kW	7.78 kW	9.07 kW
COP	8.97	4.70	3.25	2.45	1.96	1.65	1.42

Table 3.2.3: Case 1 results for NIPAAm dehumidification and evaporative cooling

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
Rate of regeneration energy	0.70 kW	3.11 kW	5.38 kW	7.79 kW	10.20 kW	12.47 kW	14.88 kW
COP	18.39	4.13	2.39	1.65	1.26	1.03	0.86

The latent load, or amount of cooling required to produce the desired dehumidification, for this case is relatively high compared to the sensible load, or amount of cooling required to drop the air to the desired temperature. This is because the rate of ventilation in this cycle is very high; all of the supply air comes from fresh ambient air. Because of this, the vapor compression only COP in the table above is not very high. As seen in the results, the desiccant dehumidification is more efficient when in line with the evaporative cooler than the vapor compression system; this is because the evaporative cooling configuration required a lower intermediate supply air humidity, which corresponds to a higher intermediate supply air temperature. This higher supply air temperature heats up the process air more at the heat exchanger, which greatly improves efficiency. Thus, the desiccant dehumidifier would only make sense to be place in line with the vapor compression cooler if the desiccant were regenerated with a cheaper heat source than electricity.

From the results above, it can be seen that the NIPAAm systems performed quite well at low percent evaporation values. At 0% evaporation, the NIPAAm dehumidifier in line with the evaporative cooler performs best, as it requires no vapor compression work. However, by 17% evaporation, the NIPAAm dehumidifier and vapor compression cooling configuration becomes the most efficient. This is because the evaporative cooling configuration requires much more dehumidification, which increases the water content of the NIPAAm, as well as the required mass of NIPAAm, both of which increase the thermal mass and regeneration heat. Before 50% evaporation, both of the NIPAAm systems are more efficient than the other three systems. The NIPAAm dehumidification and vapor compression cooling system remains the most efficient past evaporation values greater than 67%. The NIPAAm dehumidification and evaporative cooling system is more efficient than the desiccant equivalent until the evaporation percent approaches 100%. It should be noted, however, that it is yet unknown if the NIPAAm can sustain the high temperature dehumidification associated with the evaporative cooling configuration. There is potential for both NIPAAm configurations to perform better than the traditional configurations, but the NIPAAm dehumidification seems to be most promising when used in conjunction with a vapor compression cooling system. It should also be noted that the NIPAAm dehumidification would become most promising, even at 100% evaporation during regeneration, if waste heat were used during regeneration, instead of the assumed electric heating.

The following plots provide a helpful visualization of the results from this case. The first plot, shown in Fig. 3.2.1 below, shows the COP of the vapor compression cooling configurations, as plotted for various percent evaporation values.

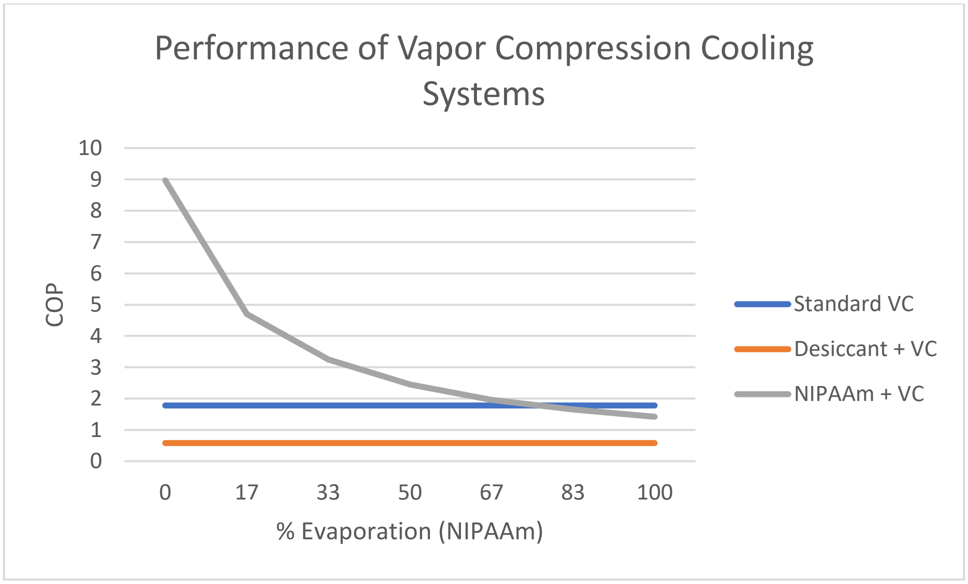


Fig. 3.2.1: Case 1 COP results for the vapor compression cooling configurations

The plot shown in Fig. 3.2.2 below shows the COP of the two evaporative cooling configurations for the first case.

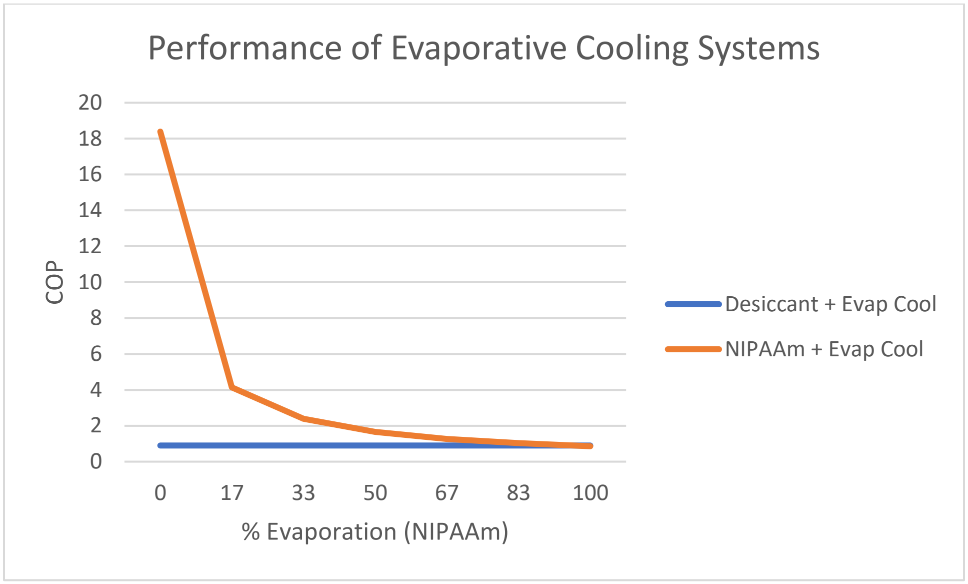


Fig. 3.2.2: Case 1 COP results for the evaporative cooling configurations

From the plots above, it can be seen that the percent evaporation significantly affects the COP of the NIPAAm configurations. For low percent evaporation, the NIPAAm systems are by far the most efficient.

3.3 The Recirculation Cycle

Another common cooling cycle is the recirculation cycle. While the Pennington cycle represents an extreme case in ventilation, where all of the supply air is comprised of outside air, the recirculation cycle trades ventilation for efficiency. In this cycle, all of the supply air is return air from the building, and the process air consists entirely of outside air [4]. Thus, in the many locations where the ambient air outside is more humid than the inside air, this cycle requires a less intensive process than the Pennington cycle, and is thus more efficient. It is only applicable, however, in buildings with low ventilation requirements. The figure below describes this cycle.

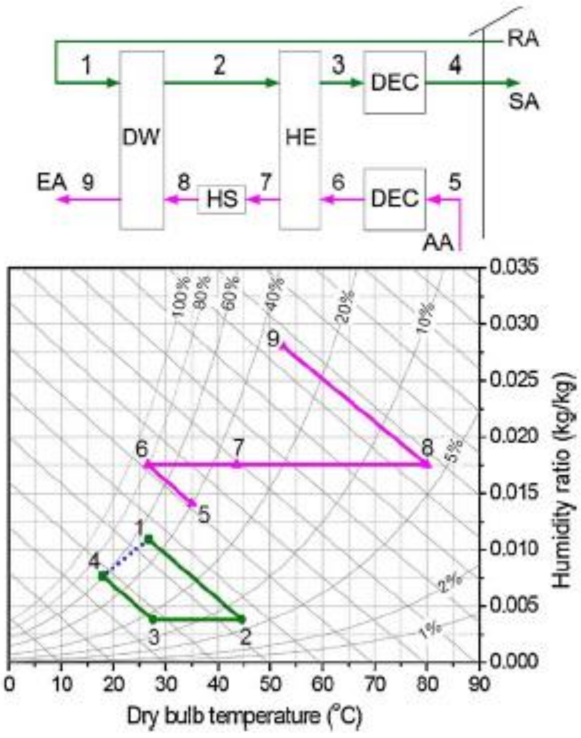


Fig. 3.3.1: Recirculation cycle schematic (top) and psychrometric process (bottom) [4]

A scenario representative of the recirculation cycle was modeled and is described under Case 2 in Table 3.1. Based on Fig. 3.3.1, the desiccant regeneration temperature was set to 80 °C, while the NIPAAm temperature and the beginning and end of dehumidification were kept at 20 °C and 25 °C, respectively. The models for each configuration were run with the prescribed conditions, and the results are described in the tables below.

Table 3.3.1: Case 2 results for vapor compression and desiccant systems

	Vapor Compression Only	Desiccant + Vapor Compression Cooling	Desiccant + Evaporative Cooling
Vapor compression compressor power	4.32 kW	1.87 kW	0.00 kW
Rate of regeneration energy	0.00 kW	31.48 kW	19.12 kW
Total electrical power required	4.32 kW	33.35 kW	19.12 kW
COP	4.51	0.58	1.02

Table 3.3.2: Case 2 results for NIPAAm dehumidification and vapor compression cooling

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
Vapor compression compressor power	1.87 kW	1.87 kW	1.87 kW	1.87 kW	1.87 kW	1.87 kW	1.87 kW
Rate of regeneration energy	0.31 kW	1.46 kW	2.55 kW	3.70 kW	4.86 kW	5.94 kW	7.09 kW
Total electrical power required	2.18 kW	3.33 kW	4.42 kW	5.57 kW	6.73 kW	7.81 kW	8.96 kW

COP	7.51	5.20	4.03	3.25	2.73	2.37	2.07

Table 3.3.3: Case 2 results for NIPAAm dehumidification and evaporative cooling

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
Rate of regeneration energy	0.72 kW	3.89 kW	6.89 kW	10.06 kW	13.24 kW	16.23 kW	19.41 kW
COP	27.12	5.00	2.83	1.93	1.47	1.20	1.00

Because the latent load was lower in this case, the COP of the vapor compression only configuration increased significantly. For this scenario, the NIPAAm systems only outperform standalone vapor compression when the percent evaporation is below 33%. The performance of the NIPAAm systems relative to the desiccant systems, however, is very similar to what was seen in the previous case. Thus, it can be seen that the NIPAAm outperforms traditional desiccants in most scenarios, except for significantly high evaporation during regeneration. NIPAAm systems with higher evaporation percentages can outperform vapor compression only if the latent load is high; if the latent load is low, the NIPAAm performs better than traditional vapor compression only if the water oozed during regeneration does not evaporate at a significant rate.

The following plots summarize the results presented in the tables above. The first figure below illustrates the results of the vapor compression cooling configurations, while the second figure illustrates the results of the evaporative cooling configurations.

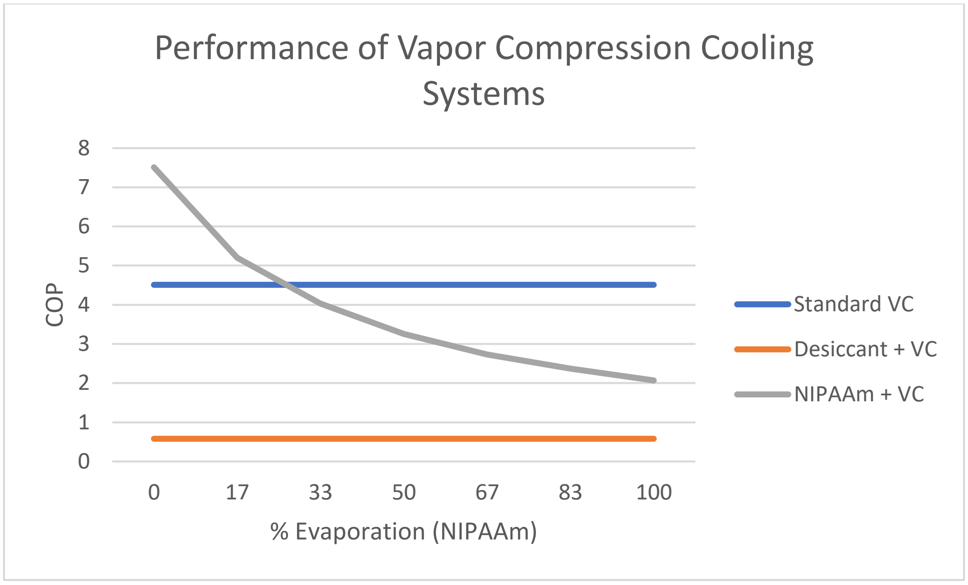


Fig. 3.3.1: Case 2 COP results for the vapor compression cooling configurations

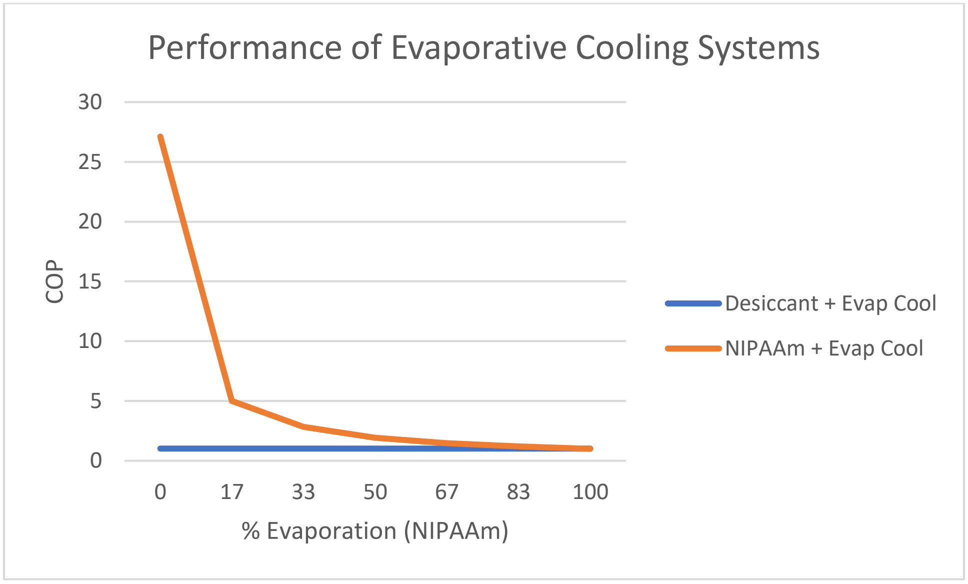


Fig. 3.3.2: Case 2 COP results for the evaporative cooling configurations

The plots above illustrate the COP of the different systems for a lower latent load than in the first case. As expected, the COP of each NIPAAm configuration drops below the COP of the standard vapor compression at a significantly lower percent evaporation than the previous case, which had a higher latent load. Thus, the NIPAAm configurations are more desirable in scenarios where the latent load is high.

3.4 The Hybrid Cycle

The third case was created as a hybrid of the two previous cases; it has substantial ventilation while still using some return air to maintain efficiency. The air properties in the third case represent a feasible scenario of operation in a humid location. This scenario, described as Case 3 in Table 3.1, models a supply air stream that is comprised of equal parts return and outside air. This allows for significant ventilation to the space while utilizing some return air to decrease the latent load. The outside air was set to 25 °C and 20.09 g/kg absolute humidity, which is 77 °F, 100% relative humidity air, a condition that can frequently occur during summer months in humid locations. The inside air was set to 22 °C and 8.16 g/kg absolute humidity, or 71.6 °F at approximately 50% relative humidity, which falls well within the guidelines for thermal comfort [1]. The cooling system is set to achieve a temperature of 11 °C, which has an absolute humidity of 8.16 g/kg at saturation. This provides a reasonably realistic scenario for a humid location, with significant ventilation, comfortable indoor conditions, and realistic conditions at the cooling system outlet. The process air humidity is in between the first and second case, so a regeneration temperature of 78 °C was selected for the desiccant. The NIPAAm start and stop temperatures were once again kept at 20 °C and 25°C, respectively. This case was run for all five configurations, and the results are summarized in the tables below.

Table 3.4.1: Case 3 results for vapor compression and desiccant systems

	Vapor Compression Only	Desiccant + Vapor Compression Cooling	Desiccant + Evaporative Cooling
Vapor compression compressor power	3.92 kW	1.01 kW	0.00 kW
Rate of regeneration energy	0.00 kW	28.90 kW	19.86 kW
Total electrical power required	3.92 kW	29.91 kW	19.86 kW
COP	2.01	0.26	0.4

Table 3.4.2: Case 3 results for NIPAAm dehumidification and vapor compression cooling

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
Vapor compression compressor power	1.01 kW	1.01 kW	1.01 kW	1.01 kW	1.01 kW	1.01 kW	1.01 kW
Rate of regeneration energy	0.32 kW	2.04 kW	3.66 kW	5.38 kW	7.10 kW	8.72 kW	10.44 kW
Total electrical power required	1.33 kW	3.05 kW	4.67 kW	6.39 kW	8.11 kW	9.73 kW	11.45 kW
COP	5.89	2.57	1.68	1.23	0.97	0.81	0.69

Table 3.4.3: Case 3 results for NIPAAm dehumidification and evaporative cooling

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
Rate of regeneration energy	0.53 kW	3.74 kW	6.75 kW	9.95 kW	13.16 kW	16.17 kW	19.37 kW
COP	14.71	2.10	1.16	0.79	0.6	0.49	0.41

This case and the second case are more realistic for standard air conditioning operation than the first case, as the amount of ventilation seen in the first case is uncommon in many scenarios. This case differs significantly from the second case, however, as 50% of the supply air comes from outside air in this case, compared to 0% in the second case. Additionally, the return air is cooler and drier in this case, which better aligns with typical human comfort than the conditions presented in the second case. Despite these differences, the trends seen from the results of this case are very similar to the trends seen in the second case. The NIPAAm systems perform better than the other three when the evaporation is below 33%, The NIPAAm dehumidification and evaporative cooling configuration drops off more quickly than the NIPAAm in line with vapor compression, and the NIPAAm and evaporative cooling configuration approaches the COP of the desiccant and evaporative cooling configuration as the water evaporated during NIPAAm regeneration approaches 100%. The plots in the figures below illustrate the results from the tables above.

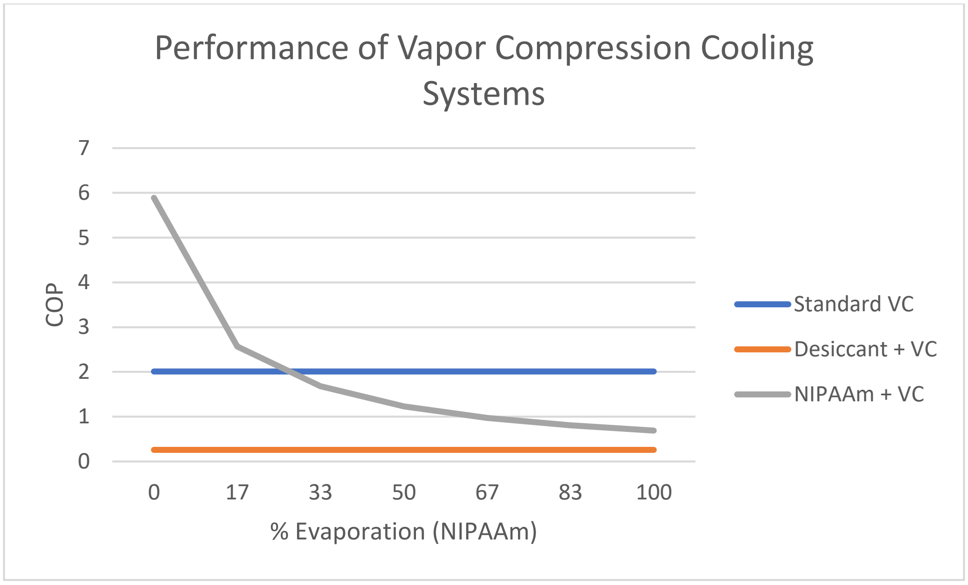


Fig. 3.4.1: Case 3 COP results for the vapor compression cooling configurations

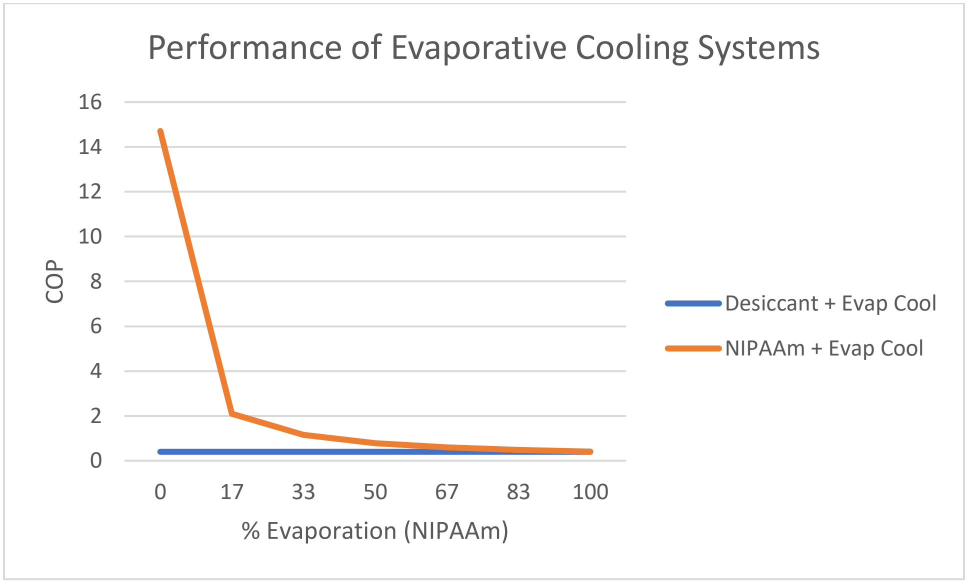


Fig. 3.4.2: Case 3 COP results for the evaporative cooling configurations

3.5 Supermarket/Ice Rink Case Study

While the previously examined cases give insight on the performance of the various air conditioning methods in certain scenarios, the following case is presented as one of the prime applications of desiccant dehumidification. The conditioning of air in supermarkets and ice rinks has been identified as one of the most promising applications of desiccant dehumidification, due to the low temperatures encountered in these scenarios. For these types of buildings, the air is required to be very cold and dry, which means that, in a standard air conditioning system that consists only of vapor compression components, the evaporator coils will reach a temperature below 0 °C, and the water condensed on the coils will freeze, which significantly reduces efficiency and performance [2]. Thus, in applications where the supply air must be dehumidified to a dew point close to 0 °C, a standard vapor compression cycle will not suffice. Additionally, desiccant dehumidification combined with evaporative cooling will not suffice in many of these scenarios either, because the air cannot get dry enough to complete the desiccant air conditioning cycle. Thus, the main method of conditioning that works for this scenario is dehumidification followed by vapor compression cooling. For this reason, only the desiccant dehumidification and vapor compression cooling and NIPAAm dehumidification and vapor compression cooling configurations were considered for this case. The proper ventilation rate for a supermarket was found based on ASHRAE codes [14], and the return air temperature for an ice rink and certain parts of a supermarket was found to be approximately 10 °C [15], [16]. The desired supply air humidity was found to be approximately 5 g/kg for the supermarket, which corresponds with a dew point of 3.90 °C for the supply air temperature. It should be noted that the supply air for this case is cooled to the dew point but not beyond, so no frost should build up on the evaporator coils. It was unclear what the regeneration temperature for an average desiccant wheel would be in this scenario, so the desiccant system was evaluated for three different regeneration temperatures: 60 °C, 70 °C, and 80 °C, while the NIPAAm temperatures were set to the same values as the previous cases. The results from this case are displayed in the tables below.

Table 3.5.1: Case 4 results for desiccant dehumidification and vapor compression cooling

Desiccant Regeneration Temperature	60 °C	70 °C	80 °C
Vapor compression compressor power	1.92 kW	1.92 kW	1.92 kW
Rate of regeneration energy	23.48 kW	30.68 kW	37.89 kW
Total electrical power required	25.40 kW	32.60 kW	39.81 kW
COP	0.31	0.24	0.2

Table 3.5.2: Case 4 results for NIPAAm dehumidification and vapor compression cooling

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
Vapor compression compressor power	1.92 kW	1.92 kW	1.92 kW	1.92 kW	1.92 kW	1.92 kW	1.92 kW
Rate of regeneration energy	0.31 kW	1.72 kW	3.04 kW	4.45 kW	5.86 kW	7.19 kW	8.59 kW
Total electrical power required	2.23 kW	3.64 kW	4.96 kW	6.37 kW	7.78 kW	9.11 kW	10.51 kW
COP	3.52	2.16	1.58	1.23	1.01	0.86	0.75

Until this point in the analysis, the effect of changing the desiccant regeneration temperature had not been observed. Because a specific desiccant dehumidifier was not modeled, the previous regeneration temperatures were selected from literature [4]. However, the results in Table 3.5.1 show that even if a regeneration temperature of only 60 °C is needed, which is somewhat of a lower bound for desiccant regeneration temperatures [4], the system efficiency is not greatly improved for the desiccant dehumidifier in line with vapor compression cooling.

Desiccant dehumidifiers are very useful in scenarios like the one described in the case above, as they prevent frost from building up on the vapor compression evaporator coils. However, the results above indicate that the NIPAAm dehumidifier could result in a higher COP for any evaporation percent. The low temperature application seen in supermarkets and ice rinks is also good for NIPAAm, as there is much less risk of the NIPAAm exceeding the LCST during dehumidification. The plot in the figure below illustrates the results presented in the tables above.

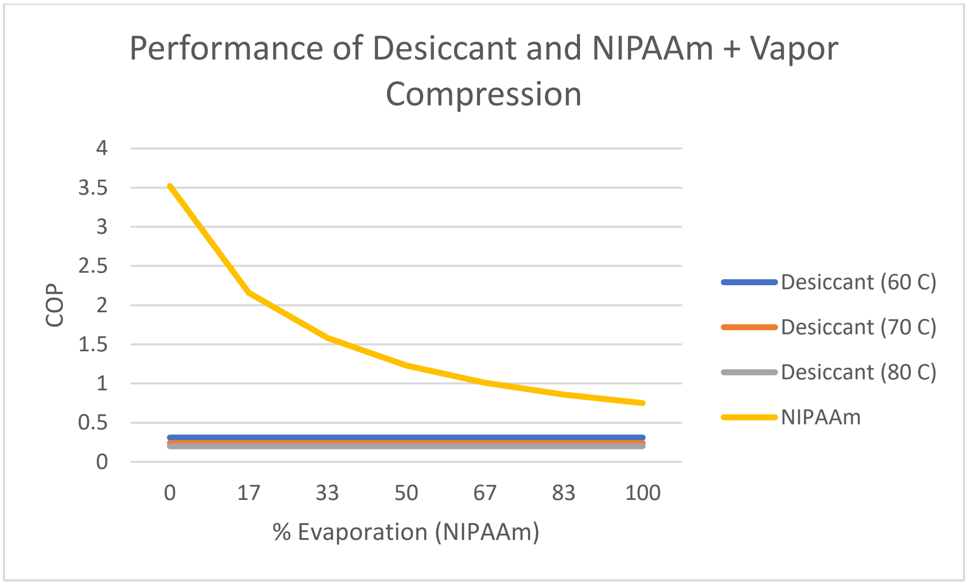


Fig. 3.5.1: Case 4 COP results for the vapor compression cooling configurations

From the plot, it can be seen that, for this scenario, the NIPAAm performs better than the desiccant, even when the desiccant regeneration temperature is as low as 60 °C. There is significant potential for NIPAAm dehumidificaiton in low temperature applications, like those seen in supermarkets and ice rinks.

3.6 Analyzing the NIPAAm Systems for a Higher Regeneration Temperature

While the LCST of NIPAAm is 32 °C, and the gel even begins to give off water at temperatures somewhat lower than 32 °C, regeneration tests on the material described in literature were conducted at 50 °C [7]. While it is currently unclear if this higher temperature is necessary for regeneration or if it was done to decrease the time of the regeneration process, the NIPAAm configurations were modeled again with the conditions described in the third case (the hybrid cycle), this time with a regeneration temperature of 50 °C. For this scenario, the NIPAAm would rotate into the dehumidification section of the wheel with a temperature of 20 °C, dehumidify the air, leave the dehumidification section with a temperature of 25 °C (as was modeled in the third case), but then the NIPAAm would need to be heated to 50 °C instead of 32 °C in the regeneration section of the wheel. The results for the NIPAAm dehumidification and vapor compression cooling and NIPAAm dehumidification and evaporative cooling configurations with the new regeneration temperature are shown in the tables below.

Table 3.6.1: Case 3 results for NIPAAm dehumidification and vapor compression cooling with a regeneration temperature of 50 °C

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
COP	3.69	2.04	1.44	1.09	0.88	0.75	0.64

Table 3.6.2: Case 3 results for NIPAAm dehumidification and evaporative cooling with a regeneration temperature of 50 °C

Percent Evaporation	0%	17%	33%	50%	67%	83%	100%
COP	4.12	1.54	0.97	0.69	0.54	0.45	0.38

When comparing these numbers to the numbers in Section 3.4, it can be seen that an increase in the regeneration temperature has the most effect at low percent evaporation values. For example, the COP for NIPAAm dehumidification and vapor compression cooling at 0% evaporation falls from 5.89 to 3.69. However, as the percent evaporation goes up, the sensible heating of the NIPAAm is a smaller portion of the total energy needed in regeneration, so the higher regeneration temperature has less of an effect. At 100% evaporation, the COP only falls from 0.69 to 0.64. The from Table 3.6.1 above, regarding the vapor compression cooling configurations, are illustrated in the following figure.

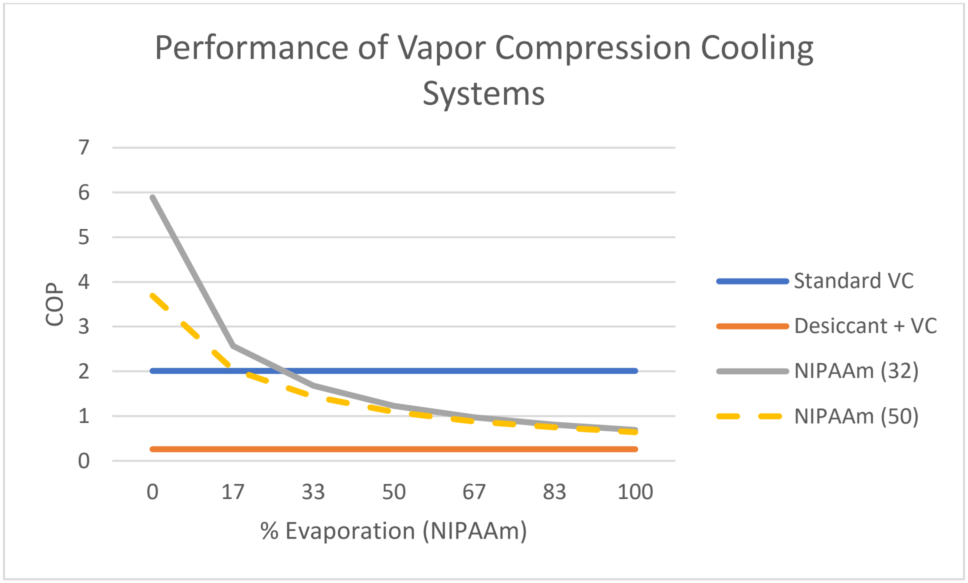


Fig. 3.6.1: Case 3 COP results for the vapor compression cooling configurations, with an added curve for NIPAAm regeneration at 50 °C

The solid lines in the figure above are the same as in Fig. 3.4.1, while the dashed line is the NIPAAm dehumidification and vapor compression cooling COP for the 50 °C regeneration temperature.

It can also be seen that the increased regeneration temperature has a greater effect on the evaporative cooling configuration; this can be explained by the fact that the evaporative cooling configuration requires more dehumidification; thus, more water is absorbed. Because more water is absorbed in the evaporative cooling configuration, the thermal mass is greater than in the vapor compression cooling configuration, and a change in regeneration temperature has a much greater effect on the system with greater thermal mass. While the total power required for the NIPAAm dehumidification and vapor compression cooling configuration is split between the air conditioning compressor and the heating and cooling required to regenerate the NIPAAm, all of the power in the evaporative cooling configuration is used to either heat or cool the NIPAAm. Thus, a change in the temperature to which the NIPAAm must be heated has a greater effect on the evaporative cooling configuration.

Something that should also be considered is that a higher regeneration temperature will also increase the amount of evaporation that takes place during regeneration. A NIPAAm wheel could have 33% evaporation when the regeneration temperature is 32 °C; however, it might have 67% evaporation when the regeneration temperature is increased to 50 °C. Thus, based on the NIPAAm dehumidification and vapor compression cooling numbers, the COP would fall from 1.68 to 0.88 when moving from 32 °C and 33% evaporation to 50 °C and 67% evaporation. It is desirable to keep the regeneration temperature as low as possible, but the temperature must be high enough to cause the regeneration process to complete in the time that the NIPAAm rotates through the regeneration section of the wheel.

3.7 Analyzing the Effect Seen by Changing the NIPAAm Start Temperature

As previously mentioned, it is known that the NIPAAm must remain below the LCST while it is in the dehumidification section of the wheel. To allow for proper dehumidification, a conservative value of 25 °C was chosen for the temperature of the NIPAAm as it leaves the dehumidification section of the wheel and enters the regeneration section. However, it is unclear at what temperature the NIPAAm must be when it enters the dehumidification section, such that it is heated only to 25 °C when it leaves the section. Thus, a value of 20 °C was chosen as the NIPAAm "start" temperature, or the temperature when it enters the dehumidification section. Because this value is somewhat arbitrary, the NIPAAm system performance was reevaluated, under the conditions described in the third case, with various start temperatures. For this analysis, the regeneration evaporation percent was kept at 67%, the NIPAAm "stop" temperature was once again set to 25 °C, and the regeneration temperature was set back to 32 °C. The NIPAAm "start" temperature was varied between 0 °C and 25 °C. The case in which the NIPAAm is cooled to 0 °C is unrealistic, as any retained water would freeze, but it is presented to demonstrate a lower bound of the NIPAAm start temperature. A value of 25 °C represents the upper bound, as this is the scenario where the NIPAAm does not receive any heat during dehumidification and does not require any pre-cooling. The table below shows the results for the reevaluated Case 3 scenario with the NIPAAm dehumidification and vapor compression cooling configuration.

Table 3.7.1: Case 3 results for NIPAAm dehumidification and vapor compression cooling with various NIPAAm start temperatures

NIPAAm Start Temperature	0 °C	5°C	10 °C	15 °C	20 °C	25 °C
COP	0.95	0.96	0.96	0.97	0.97	0.97

It can be seen from the table above that the temperature to which the NIPAAm must be cooled has a very minimal effect on the overall efficiency when a realistic evaporation percentage is used. This is because the thermal mass of the NIPAAm in this section is low, as the water content of the NIPAAm is at a minimum in the cooling section, and the cooling is assumed to be provided to the NIPAAm at the COP of the vapor compression sub-system, which often has a COP value significantly greater than 1. Thus, while the selection of the NIPAAm start temperature was somewhat arbitrary for the previous cases examined, it can be seen that the NIPAAm start temperature does not have a significant effect on the system efficiency.

This analysis was not performed for the evaporative cooling method for several reasons. First, the evaporative cooling method does not require significant consumption of power, so an increase in the amount of NIPAAm pre-cooling will only affect the consumption of water, not the system efficiency. Additionally, the evaporative cooler cannot achieve the low temperatures that were evaluated in this case. For the scenario described in Case 3, the lower limit of the air temperature that the evaporative cooler can produce is approximately 8 °C. This is possible only if the air used for cooling the NIPAAm is supply air that was dried to 0% relative humidity and then humidified to 100% relative humidity, which is not feasible in practice. If ambient air is used for cooling the NIPAAm, instead of dried supply air, the lowest achievable temperature ranges from approximately 11 °C, for ambient air at 20 °C and 30% relative humidity, to 25 °C, for ambient air at 30 °C and 70% relative humidity. Thus, the evaporative cooler cannot achieve temperatures as low as the vapor compression system. If it is found that the NIPAAm must be cooled to a significantly low temperature to sustain continuous operation, then an evaporative cooler will not suffice in providing the cooling to the NIPAAm.

3.8 Revisiting Case 3 with Waste Heat Used for Dehumidifier Regeneration

While the previous analyses regard the "worst case scenario", or the scenario where the desiccant and NIPAAm configurations are regenerated with electric heating, the best case for a desiccant or NIPAAm dehumidifier can be made when the regeneration heat source is a significantly abundant source, like waste heat. To analyze this scenario, the third case was revisited with the assumption that the NIPAAm and desiccant dehumidifiers could be regenerated "for free" (i.e.: with a heat source that is abundant and requires no extra consumption of fuel, such as waste or solar heat). The plot below shows the COP for vapor compression cooling configurations, where the COP only accounts for electricity input.

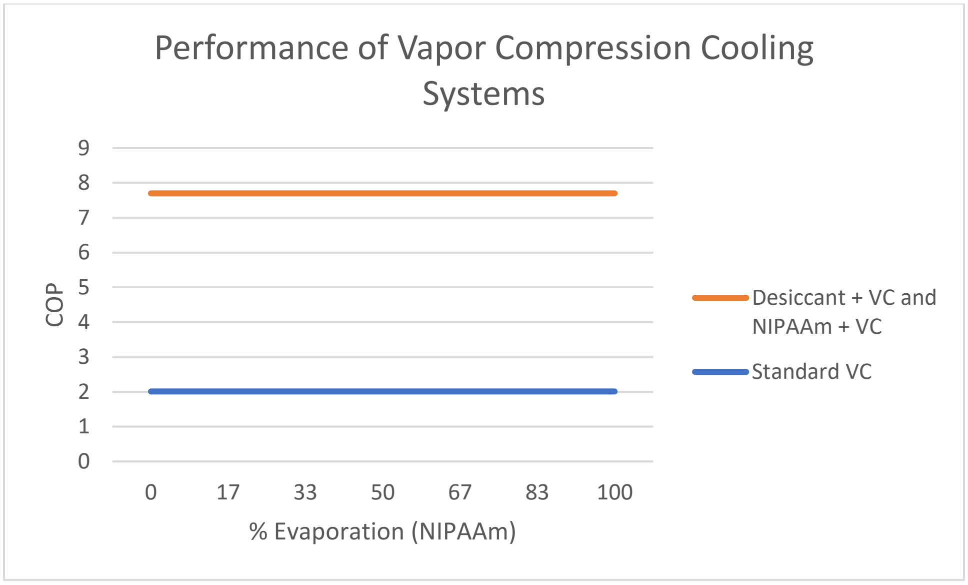


Fig. 3.8.1: Case 3 COP results for the vapor compression cooling configurations, with a waste heat source considered for regeneration

Because the heat of regeneration is not supplied by electricity in this scenario, the only input to the system is the electricity required to run the vapor compression compressor. This means that the desiccant and NIPAAm dehumidifiers have the same COP. It also means that the evaporative cooling configurations have an infinite COP, as the evaporative coolers do not require electricity input (when neglecting any pumps or blowers). For this scenario, it might seem like the NIPAAm dehumidifier offers no benefits; however, the NIPAAm dehumidifier still requires less heat, even if the heat is supplied by a waste or solar heat source. This could lead to smaller and cheaper components and could be applicable in cases where the waste heat source is not significantly large.

From the analyses in this section, the potential of a NIPAAm dehumidifier can be seen. When electric heating is used to regenerate the dehumidifier, the NIPAAm configurations require less electricity input than standard vapor compression or desiccant dehumidification when the latent load is high, or the percent of water evaporated during NIPAAm regeneration is low. Additionally, it was found that the NIPAAm dehumidifier would be more efficient than a desiccant dehumidifier when used with an evaporative cooler, provided that the NIPAAm dehumidifier could properly operate at the high temperatures associated with the evaporative cooling configurations. Increasing the NIPAAm regeneration temperature from the LCST of 32 °C to the value of 50 °C seen in literature, it can be seen that the NIPAAm requires a non-trivial increase in regeneration heat, but even at the higher regeneration temperature, the NIPAAm is still quite efficient. Finally, it was shown that if the NIPAAm were regenerated with waste heat, it would have the same electricity-based COP as the desiccant system, but the NIPAAm would require less heat and could regenerate at a lower temperature than a traditional desiccant. This is desirable in scenarios where the waste heat is not so abundant as to become trivial, and it is also desirable in scenarios where the waste heat source is available at temperatures higher than the NIPAAm regeneration temperature but lower than the regeneration temperature of a traditional desiccant. Because the source of regeneration heat exists at some temperature, waste and solar heat can only be used when they are available at a temperature greater than the regeneration temperature of the dehumidifier. Since the regeneration temperature of NIPAAm is less than that of a traditional desiccant, the NIPAAm dehumidifier would be applicable in a greater number of scenarios.

3.9 Revisiting Case 3 with Various Heat Exchanger Effectiveness Values

For the analyses that were conducted, a heat exchanger effectiveness of 0.99 was assumed; however, not all systems will utilize a heat exchanger with an effectiveness this high. Thus, Case 3 was reevaluated, and the heat exchanger effectiveness was varied to observe the impact that a lower effectiveness has on the NIPAAm and traditional desiccant systems. For this analysis, both of the NIPAAm configurations and both of the desiccant configurations were reexamined for heat exchanger effectiveness values of 0.80, 0.85, 0.90, and 0.95, along with the original value of 0.99. Two plots were created: one plot demonstrates the performance of the vapor compression cooling configurations, and the other demonstrates the performance of the evaporative cooling configurations. For the NIPAAm configurations, separate curves were created for each heat exchanger effectiveness value. For the traditional desiccant, the data associated with the five separate heat exchanger effectiveness values were gathered into a shaded area, as it was found that the change in COP for the desiccant configurations was minimal. The plots are shown below.

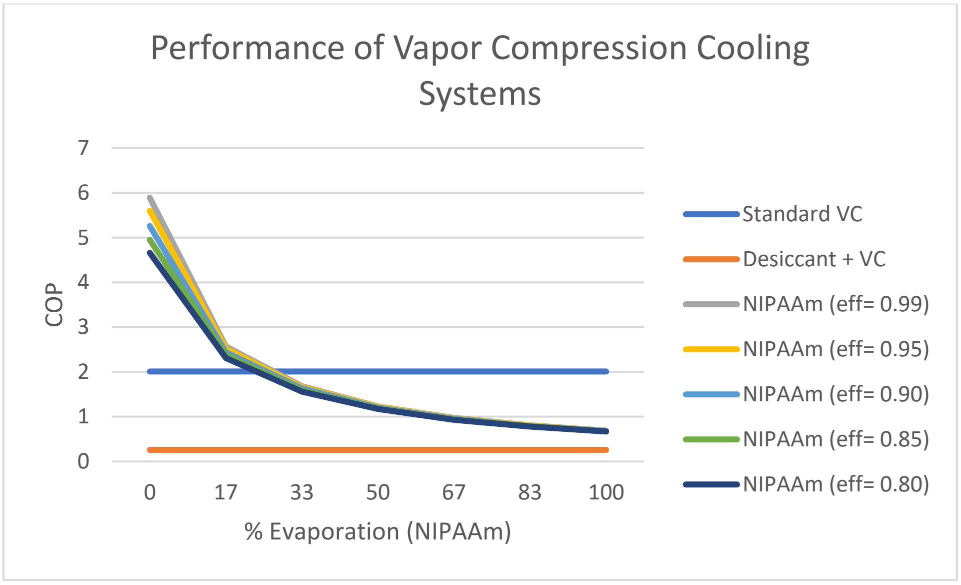


Fig. 3.9.1: Case 3 COP results for the vapor compression cooling configurations, with heat exchanger effectiveness varied

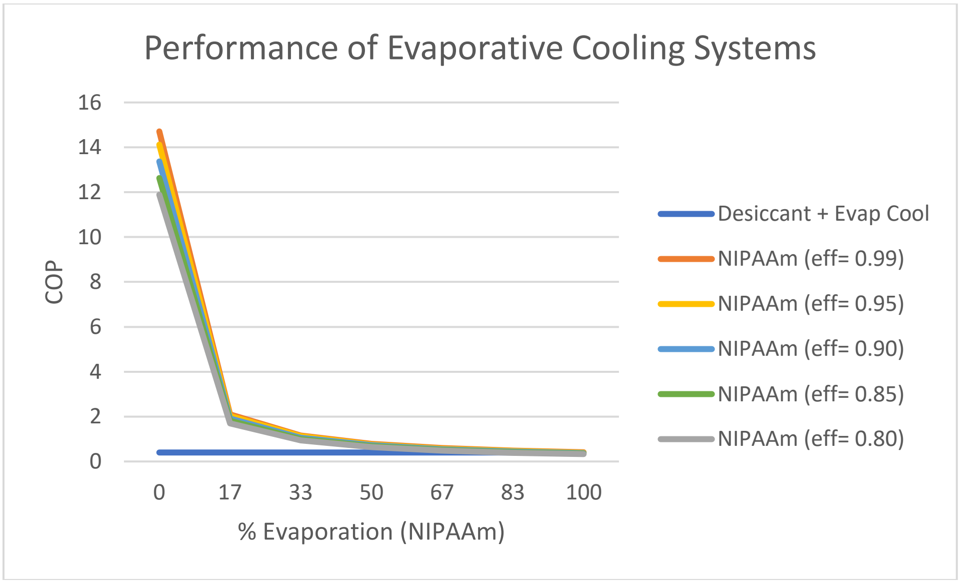


Fig. 3.9.2: Case 3 COP results for the evaporative cooling configurations, with heat exchanger effectiveness varied

For both vapor compression and evaporative cooling, it can be seen that the change in COP for the NIPAAm is relatively small, ranging from 7 to 1% decrease in COP (depending on the percent evaporation) for a decrease in heat exchanger effectiveness of 0.05. The change in desiccant COP is even smaller, ranging from 3% to less than 1%.

The heat exchanger effectiveness was not decreased below 0.8, as it was found, for this specific case, that the evaporative cooling cycles would not be possible with heat exchanger effectiveness values much lower than 0.8 (the air would have to be dried below 0 g/kg humidity to achieve the cycle, which cannot happen).

4 CONCLUSIONS AND RECOMMENDATIONS

From the results of the system-level models, it is clear that a NIPAAm dehumidifier has the potential to perform better than traditional alternatives. The following list summarizes the findings of this report, based on the results in the previous chapter:

- NIPAAm dehumidification increases system efficiency when added to a standard vapor compression system if the latent load is high
- For low latent loads, NIPAAm dehumidification increases vapor compression system efficiency when the percent evaporation seen during regeneration is below approximately 25%
- NIPAAm dehumidification is more efficient than traditional desiccant dehumidification for almost any percent evaporation seen during NIPAAm regeneration
- NIPAAm has great potential in low temperature applications where standard vapor compression cannot be used, and the NIPAAm is less likely to exceed the LCST during dehumidification for low temperature applications

NIPAAm shows the most obvious promise when used in conjunction with vapor compression cooling in low temperature applications, as exemplified in the supermarket/ice rink scenarios. These applications would require the least nuanced design and would be the most likely to provide significant improvement over current system configurations. The NIPAAm dehumidifier still holds promise in other applications, but the considerations regarding the dehumidifier design (such as the percent evaporation of regeneration water and the NIPAAm temperature rise) hold greater weight in scenarios other than the low temperature cases.

The models detailed in this report indicate that a NIPAAm dehumidifier could significantly improve overall system efficiency in many scenarios, given that the dehumidifier behaves as it was modeled. For a NIPAAm dehumidifier to work, the sorbent must remain below a certain temperature (roughly 25 to 30 °C) throughout the entirety of the dehumidification section of the wheel. To achieve this, it was proposed that the NIPAAm wheel be cooled immediately before it enters the dehumidification section; however, it is unknown how easily this could be achieved, and there are limitations regarding the temperature to which the NIPAAm could be cooled. Therefore, a NIPAAm wheel that is internally cooled during the dehumidification section is proposed. In this configuration, some tubes would run through the NIPAAm wheel, such that the NIPAAm would fill the annular space around the tubes. A cooling fluid, such as the process air, could be flowed through the tubes, thus ensuring the NIPAAm does not heat up as latent heat is released during dehumidification. This would essentially combine the downstream heat exchanger with the NIPAAm wheel, and it would cause the dehumidification process to become isothermal or near-isothermal. Internally cooled desiccant wheels have been demonstrated in literature, meaning that an internally cooled NIPAAm wheel is likely possible. This could end up being the most feasible design for a NIPAAm wheel, as it would ensure that the NIPAAm does not exceed a certain temperature during dehumidification, and the conceptual design is generally obvious. The figure below shows a conceptual design for an internally cooled NIPAAm wheel.

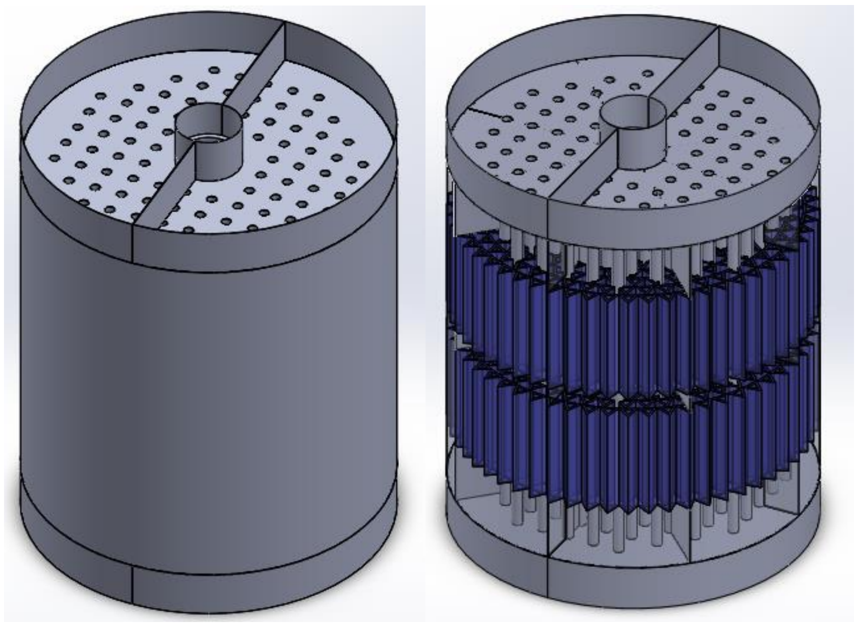


Fig. 4.1: Conceptual design for a NIPAAm wheel with housing shown (left) and not shown (right). NIPAAm is placed in the annular space (shown in blue), supply air enters through the axle and flows through the annular space, and process air flows through the heat exchanger tubes.

When pursuing future work on a NIPAAm dehumidifier, it is recommended that the initial efforts focus on the evaporation fraction and thermal response of the NIPAAm. The percent of water evaporated during regeneration should be more rigorously examined for NIPAAm regeneration at different temperatures, and efforts to improve system performance should focus on reducing the percent evaporation. Additionally, the transient temperature response of the NIPAAm should be observed for dehumidification over some period of time. If it is found that the NIPAAm does not significantly change in temperature as it absorbs moisture from the air, then the pre-cooling of the NIPAAm before dehumidification might not be an issue. However, if it is found that the NIPAAm significantly changes in temperature during dehumidification, then considerations must be made regarding the prevention of excessive NIPAAm temperature rise during dehumidification.

Going forward, a more complex model can be made. The vapor compression model presented in this paper can be easily modified to account for dynamic changes in inlet air properties, but numerical models would be required for the desiccant and NIPAAm wheels. The conditioned space could be modeled in a CFD program, such as ANSYS Fluent, and a top-level Python script could be set up to pass the air properties, as determined by the air conditioning component Python models, directly to Fluent. The Fluent simulation can be set-up and initialized entirely through the use of a Python script, so the overall model, consisting of Python models for the air conditioning components and a Fluent model for the air within the space, could be initialized and run from one command in the computer's command line.

REFERENCES

[1] Centers for Disease Control and Prevention, "Indoor Environmental Quality: Building Ventilation Resources - NIOSH Workplace Safety and Health Topic," 1 September 2015. [Online]. Available: https://www.cdc.gov/niosh/topics/indoorenv/temperature.html. [Accessed 20 March 2019].

[2] A. A. Pesaran, "A Review of Desiccant Dehumidification Technology," in *EPRI's Electric Dehumidification: Energy Efficient Humidity Control for Commercial and Institutional Buildings Conference*, New Orleans, 1993.

[3] W. Goetzler, R. Zogg, J. Young and C. Johnson, "Energy Savings Potential and RD&D Opportunities for Non-Vapor Compression HVAC Technologies," U.S. Department of Energy Office of Scientific and Technical Information, Burlington, 2014.

[4] D. La, Y. Dai, Y. Li, R. Wang and T. Ge, " Technical development of rotary desiccant dehumidification and air conditioning: A review," *Renewable & Sustainable Energy Reviews,* vol. 14, no. 1, pp. 130-147, 2010.

[5] United States Environmental Protection Agency, "Understanding Global Warming Potentials," [Online]. Available: https://www.epa.gov/ghgemissions/understanding-global-warming-potentials. [Accessed 20 March 2019].

[6] A. Hauer, "Sorption Theory for Thermal Energy Storage," in *Thermal Energy Storage for Sustainable Energy Consumption*, H. Ö. Paksoy, Ed., Dordrecht, Springer, 2007, pp. 393-408.

[7] K. Matsumoto, N. Sakikawa and T. Miyata, "Thermo-responsive gels that absorb moisture and ooze water," *Nature Communications,* vol. 9, no. 1, 2018.

[8] T. Ge, F. Ziegler and R. Wang, "A mathematical model for predicting the performance of a compound desiccant wheel (A model of compound desiccant wheel)," *Applied Thermal Engineering,* vol. 30, no. 8-9, pp. 1005-1015, 2010.

[9] M. Sultan, T. Miyazaki, S. Koyama and Z. M. Khan, "Performance evaluation of hydrophilic organic polymer sorbents for desiccant air-conditioning applications," *Adsorption Science & Technology,* vol. 36, no. 1-2, pp. 311-326, 2018.

[10] Missouri Division of Energy, "Why 400 CFM per Ton is used for Determining "Standard Air" Conditions Air Volumes," [Online]. Available: https://energy.mo.gov/sites/energy/files/61-why-400-cfm-per-ton.pdf. [Accessed 20 March 2019].

[11] G. Angrisani, C. Roselli and M. Sasso, "Effect of rotational speed on the performances of a desiccant wheel," *Applied Energy,* vol. 104, pp. 268-275, 2013.

[12] U.S. Department of Energy, "Explaining Central Air Conditioner & Heat Pump Standards," [Online]. Available: https://www.energy.gov/sites/prod/files/2015/11/f27/CAC%20Brochure.pdf. [Accessed 20 March 2019].

[13] Air-Conditioning, Heating, and Refrigeration Institute, "2015 Standard for Performance Rating of Commercial and Industrial Unitary Air-conditioning and Heat Pump Equipment," [Online]. Available: http://www.ahrinet.org/App_Content/ahri/files/STANDARDS/AHRI/AHRI_Standard_340-360_2015.pdf. [Accessed 20 March 2019].

[14] ASHRAE, "Ventilation for Acceptable Indoor Air Quality," 2015. [Online]. Available: https://www.ashrae.org/File%20Library/Technical%20Resources/Standards%20and%20Guidelines/Standards%20Addenda/62_1_2013_p_20150707.pdf. [Accessed 25 March 2019].

[15] A. Palmowska and B. Lipska, "Research on improving thermal and humidity conditions in a ventilated ice rink arena using a validated CFD model," *International Journal of Refrigeration,* vol. 86, pp. 373-387, 2017.

[16] D. B. Jani, M. Mishra and P. K. Sahoo, " Performance analysis of hybrid solid desiccant-vapor compression air conditioning system in hot and humid weather of India," *Building Services Engineering Research & Technology,* vol. 37, no. 5, pp. 523-538, 2016.

[17] D. Clark, C. Hurley and C. Hill, "Dynamic Models for HVAC System Components," *ASHRAE Transactions,* vol. 91, no. 1, pp. 737-751, 1985.

[18] Linric.com, "PSYCHROMETRICS for Engineers," [Online]. Available: http://www.linric.com/demos.htm. [Accessed 25 March 2019].

[19] M. J. Moran and B. R. Munson, MAE 240 - Thermofluids I, Hoboken: John Wiley & Sons, Inc., 2015.

[20] B. Tashtoush, M. Molhim and M. Al-Rousan, "Dynamic model of an HVAC system for control analysis," *Energy,* vol. 30, pp. 1729-1745, 2005.

APPENDIX A

EXPLANATION OF GRAPHICAL INTERFACE AND PYTHON SCRIPTS

After inputting the initial indoor, the outdoor, and the desired outlet air conditions, the program calculates the dynamic response of the control volume. A diagram of the control volume is shown in the figure below.

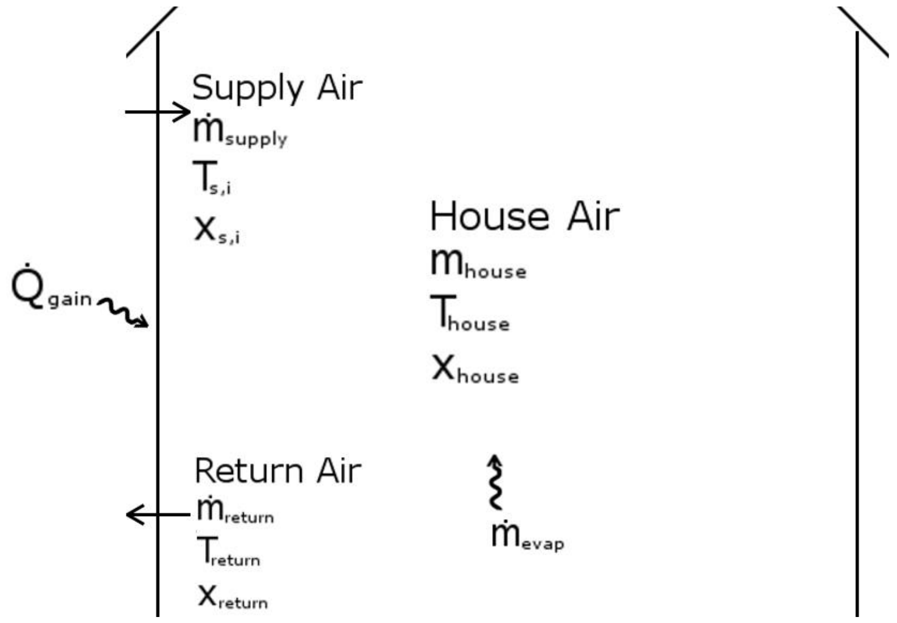


Fig. A.1: Control volume for the air within the house that is being conditioned, with mass flows, external heat gain, and internal moisture gain

For the models that were created, the heat gain and indoor evaporation rate were set to zero. The heat gain was set to zero because it was found to be miniscule compared to the thermal mass of the house and the cooling provided by the air conditioning, and the evaporation rate was set to zero for simplicity, as this is the case when the house is unoccupied.

For some volume of air within the house and the selected mass flow rates for the various air streams, the temperature and humidity drop of the control volume is determined for a given time-step, as described later in Section 2.4. The simulation ends when the thermal mass weighted average temperature of the conditioned space reaches a value 1 °F lower than the thermostat set temperature. To simplify calculations, it is assumed that the supply air entering the conditioned space does not mix with the remaining air. This was done so that the temperature and humidity ratio of the return air are kept constant throughout the transient process, such that the air conditioning components can be modeled for constant inlet properties. Because the air stream properties are kept constant, the modeling of the components can then be seen as a simulation of steady state performance, while the simplified transient response of the air within the conditioned space gives an estimate of the system operation time. The operation time can then be used to find the total amount of water removed from the air during operation, which can be used to determine how much desiccant and NIPAAm is required.

After receiving the model inputs, the graphical interface calls one of two functions. If the user selected vapor compression as the cooling method, the first function is called. This function, written in house_air.py, gathers relevant values regarding the steady state performance of the three models that contain vapor compression cooling, given the desired conditions. The second function, which was written in house_air_evap_cool.py and is utilized if the user selected evaporative cooling as the desired method, gathers values regarding the steady state performance of the two evaporative cooling configurations. Each model, after gathering temperature and humidity values at each state in the cycle, as well as heat and power inputs to the devices, calculates the temperature and humidity drop of the conditioned space over time.

After the supply air exits the air conditioning system, it is provided to the conditioned space. Before it reaches the control volume, however, it travels though ducting. The ducting is initially at the temperature of the control volume, and when the cold supply air passes through, the supply air is heated while the ducting is cooled. This process is described in the equation below, which was taken from an ASHRAE paper on dynamic modeling for air conditioning components [17].

$$\frac{dT_{duct}}{dt} = \frac{T_{s,o} - T_{duct}}{\frac{U_{duct,i}}{U_{duct,i} + U_{duct,o}} * \frac{m_{duct} c_{p,duct}}{\dot{m}_{supply}\left(c_{p,a} + x_{s,o} c_{p,v}\right)}} \tag{A.1}$$

For the scenarios that were modeled, it is assumed that the heat transfer coefficient on the inside of the ducting is far greater than the heat transfer coefficient on the outside of the ducting, due to the occurrence of forced convection within the ducting and natural convection on the ducting exterior. Thus, the external heat transfer coefficient is neglected, and it is assumed that the ducting approaches a steady state temperature equal to the temperature of the supply air as it enters the duct.

The temperature and humidity of the air within the conditioned space are at some initial values $T_{house,i}$ and $x_{house,i}$, respectively. Some percentage of the total volume is occupied by solids, which are at the same initial temperature as the air. This scenario is illustrated in the figure below, where $m_{house,i}$ is the initial mass of air in the space, which, for the scenario modeled, is the mass of air in the space for the entire process, as the supply and return mass flow rates are constant.

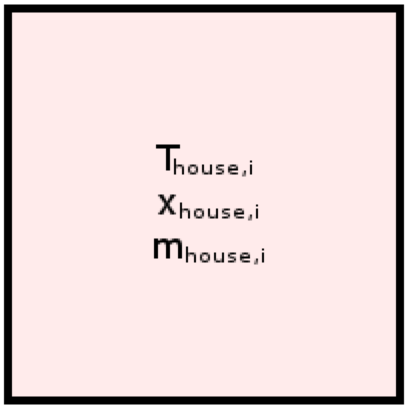


Fig. A.2: Initial properties for the air within the conditioned space

The cool and dry supply air enters the control volume through some inlet and the return air exits through an outlet. Because the various components of the system are modeled for steady state performance, the properties of the air that enters these components should remain constant with time. Because the supply air is some mix of return and outside air, the return air properties must then stay constant. To achieve this, it is assumed that the supply air and return air do not mix or exchange heat, as shown in the figure below.

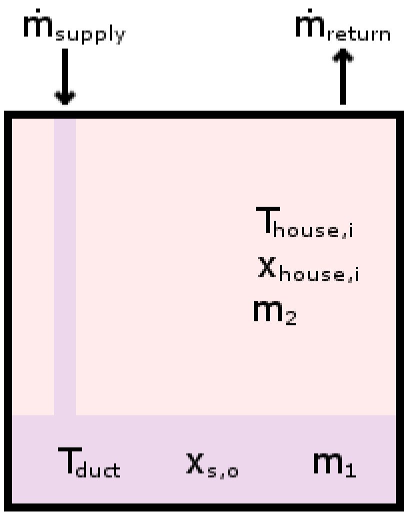


Fig. A.3: Separated supply and return air within the conditioned space

The mass of supply air within the space, m_1, is simply the product of the mass flow rate and the time that the system has been running, $\dot{m}_{supply}\Delta t$. The mass of initial air that remains within the space, m_2, is the initial air mass minus the product of mass flow rate and operation time, $m_{house,i} - \dot{m}_{return}\Delta t$.

While the air is kept stratified for the purpose of maintaining steady conditions at the outlet, the mass weighted average temperature and humidity ratio are calculated at each time step. These are the values that would result if the air would mix completely and reach equilibrium with the solids in the room. These values are computed for display on the graphical interface, and for the stop criterion. A loop in the house_air and house_air_evap_cool functions adds some mass of supply air and removes some mass of return air from the conditioned space at every time step. Once the average temperature reaches a value 1 °F lower than the thermostat set temperature that was input by the user, the loop breaks, and the model stops. The following equations describe the mass weight average humidity ratio and temperature for the conditioned space at any time t, where the specific heat of the solids, $c_{p,solid}$, is given per unit volume instead of mass.

$$x_{house,avg} = \frac{m_1 x_{s,o} + m_2 x_{house,i}}{m_1 + m_2} \tag{A.2}$$

$$T_{house,avg} = \frac{m_1\left(c_{p,a} + x_{s,o}c_{p,v}\right)T_{duct} + \left(m_2\left(c_{p,a} + x_{house,i}c_{p,v}\right) + V_{solid}c_{p,solid}\right)T_{house,i}}{(m_1 + m_2)\left(c_{p,a} + x_{house,avg}c_{p,v}\right) + V_{solid}c_{p,solid}} \tag{A.3}$$

It should be noted that the validity of the stratified model is contingent on the operation time. At a certain time, $m_{house,i}/\dot{m}_{return}$, all of the air within the conditioned space will have been replaced with supply air, at which point it is impossible for the return air to be at the initial conditions.

As mentioned previously, the vapor compression cooling configurations were modeled in house_air.py. The model starts by defining various properties, including the NIPAAm regeneration temperature, total volume within the conditioned space, indoor evaporation rate, supply air total mass flow rate, fraction of the supply air mass flow that comes from outside air, mass flow rate of the air that cools the vapor compression condenser, specific heats of the desiccant and NIPAAm, moisture absorption capacity for the desiccant and NIPAAm, as well as the percent of the control volume which is solid and volumetric specific heat of the solids. After defining these properties and receiving the inputs from the graphical interface, the model accounts for the mixing of the return and outdoor air streams to form the supply air and calculates the temperature, $T_{s,i}$, and humidity ratio, $x_{s,i}$, of the mixed supply air. The code then models the mixing of return and outside air to form the process air and calculates the temperature, $T_{p,i}$, and humidity ratio, $x_{p,i}$, of the process air stream.

For the vapor compression only configuration, the model calls the AC.py function with the supply air properties, desired outlet properties, and outside air properties as inputs. The AC function returns the power required, as well as several values used to create graphics in the graphical interface.

For the desiccant dehumidification and vapor compression cooling configuration and the NIPAAm dehumidification and vapor compression cooling configuration, the mixed supply air is sent first to the dehumidifier. For these configurations, the Dehum function is called, with the mixed supply air temperature and humidity ratio, as well as the desired outlet humidity ratio, as inputs. Based on the required outlet humidity, the temperature is found for the supply air as it leaves the dehumidifier. The dehumidifier outlet properties are used as the inlet properties for the heat exchanger. The heat exchange process is modeled by calling the HX function, which uses the dehumidified supply air properties as inputs for the hot-side inlet properties and uses the mixed process air properties as inputs for the cold-side inlet properties. The function returns the temperatures for the supply and process air streams as they exit the heat exchanger. Now that the supply air has been dehumidified and pre-cooled, it is sent to the vapor compression cooling system. The properties of the supply air as it exits the heat exchanger are used as inputs for the AC function, along with the outside air and desired outlet air conditions. Once again, the AC function returns the power required at the compressor. Additionally, the model determines the total amount of water that was absorbed during the process.

Aside from collecting values for the various states in the system, the house_air function also calculates the dynamic temperature change within the conditioned space, as described in the previous section. After a certain number of time steps, the lumped temperature of the space reaches the desired value and the loop breaks. At this point, the time required to cool the conditioned space is recorded. During the loop, the lumped temperature and humidity ratio for the conditioned space is recorded at each time step. This information is passed back to the graphical interface so that an interactive timeline of the temperature and humidity within the house can be viewed.

Modeling of the vapor compression air conditioner was accomplished with three separate scripts. The first script, AC.py, is a top level script that contains certain information that is passed to the other two scripts, such as the refrigerant, heat transfer coefficients, outdoor and indoor air pressure, and isentropic efficiency of the compressor. Aside from containing this information, the AC script also calculates the power used by the compressor and creates the arrays for the T-s and P-h diagrams that are displayed in the graphical interface.

To model the vapor compression air conditioning evaporator, the script HX_AC_evap.py is used. Before calling the HX_AC_evap function, the AC script calculates the humidity of the air after it is cooled by the evaporator coils. If the desired outlet temperature is greater than the dew point, the outlet humidity ratio is equal to the inlet humidity ratio. If the desired outlet temperature is less than the dew point of the inlet air, then the outlet humidity ratio is the saturation humidity ratio associated with the outlet temperature, as determined by the script x_s.py. Once the outlet humidity ratio has been determined, the AC script calls the HX_AC_evap script with inputs for the mass flow rate of air over the evaporator coils, the air pressure within the building, the inlet air temperature and humidity, the outlet air temperature and humidity, the refrigerant, and the heat transfer coefficient of the evaporator.

The vapor compression condenser was modeled in HX_AC_cond.py. Much like the script that models the evaporator, the script for the condenser uses an iterative scheme to solve the relevant heat exchanger equations, as described in Section 2.3.2.

The air conditioning compressor was not modeled in its own script; rather, it was modeled in the script AC.py.

The model for dehumidification was implemented in the script Dehum.py. The Dehum script is used to find the outlet temperature of the supply air for a given amount of dehumidification.

The heat exchanger that pre-cools the supply air and pre-heats the process air was modeled in HX.py.

The energy required for the regeneration process is different for the desiccant and NIPAAm configurations and was not modeled in a separate script; instead, the regeneration for both configurations was modeled in the graphical interface script (GUI.py). This was done because the regeneration energy is dependent on certain parameters that cannot be determined with the models' current level of sophistication. In the graphical interface, the user can adjust a slider to control the required regeneration temperature for the desiccant, after which the GUI.py script utilizes the selected regeneration temperature in Eq. 2.14 to calculate the rate of heating required for regeneration. The script then multiplies the rate of heating by the cycle time to find the total regeneration heat, which it then displays, along with the COP. The slider allows the user to vary the regeneration temperature between two extremes. The lower bound of the slider is the temperature of the process air leaving the heat exchanger, which represents the case in which the desiccant can be regenerated at the temperature of the process air after it is pre-heated in the heat exchanger, meaning no heat is required from the electric heater. The upper bound of the slider was set to a temperature of 140 °C, as the desiccant regeneration temperatures found in literature were all lower than 140 °C, meaning that the slider should encompass the entire range of possible regeneration temperatures.

For the NIPAAm regeneration, each rate of heat transfer in Eq. 2.20 is dependent on an unknown parameter. Three sliders are displayed in the GUI, such that the user can control the NIPAAm temperature at the beginning of the dehumidification portion, the evaporation fraction, and the NIPAAm temperature at the end of the dehumidification portion. The GUI script then calculates the three rates of heat transfer and divides the rate of cooling by the cooling system COP to find the required electrical power. It should be noted that the evaporative cooling configuration does not consume electricity to produce cooling; rather, it consumes water. Thus, the COP in terms of cooling rate per unit electrical power is infinite for the evaporative cooler (when neglecting any blowers used to induce airflow), and the cooling term drops out of regeneration energy equation. At the moment, it is not obvious how the vapor compression or evaporative cooling systems would interface with the NIPAAm wheel. It is assumed that cooling is provided to the wheel at the COP of cooling system and is done by means that do not involve the NIPAAm absorbing any moisture before entering the dehumidification section.

If the user selects the evaporative cooling configurations, the transient cooling process for the air within the conditioned space is evaluated using the model in house_air_evap_cool.py. This script, like the house_air script, defines various properties and models the mixing of return and outside air to form the supply and process air streams. For these configurations, an iterative process is used to find the amount of supply air dehumidification required to reach the desired outlet conditions, as illustrated previously in Fig. 2.3.7.1. First the mixed supply air properties are given as inputs to the Dehum function, the outputs of which are given as inputs to the HX function, along with the process air properties. The error for each iteration is the difference between the enthalpy of the supply air exiting the heat exchanger and the desired enthalpy of the supply air after evaporative cooling, as the evaporative cooling is an isenthalpic process. The iterative solver varies the humidity of the supply air leaving the dehumidifier until the error is sufficiently small. At this point, all states in the system process are known, and the house_air_evap_cool function models the transient behavior of the conditioned space, as previously described. The evaporative cooler was also not given a separate script and was instead modeled directly within the house_air_evap_cool script.

The highest level script in the overall model is GUI.py, which creates the graphical user interface and, based on user input, decides which scripts to run and what conditions to model. When run, the GUI script first creates a window that allows the user to pass inputs and view various plots, graphics, and values. The first page displayed by the GUI script is an explanation of the tool and a disclaimer about the nature of the models that were implemented. After clicking the "Next" button, the user is taken to a configuration selection page, in which the user can choose between simulating the vapor compression configurations (vapor compression only, desiccant dehumidification and vapor compression cooling, and NIPAAm dehumidification and vapor compression cooling), or the evaporative cooler models (desiccant dehumidification and evaporative cooling and NIPAAm dehumidification and evaporative cooling). This page is shown in the figure below.

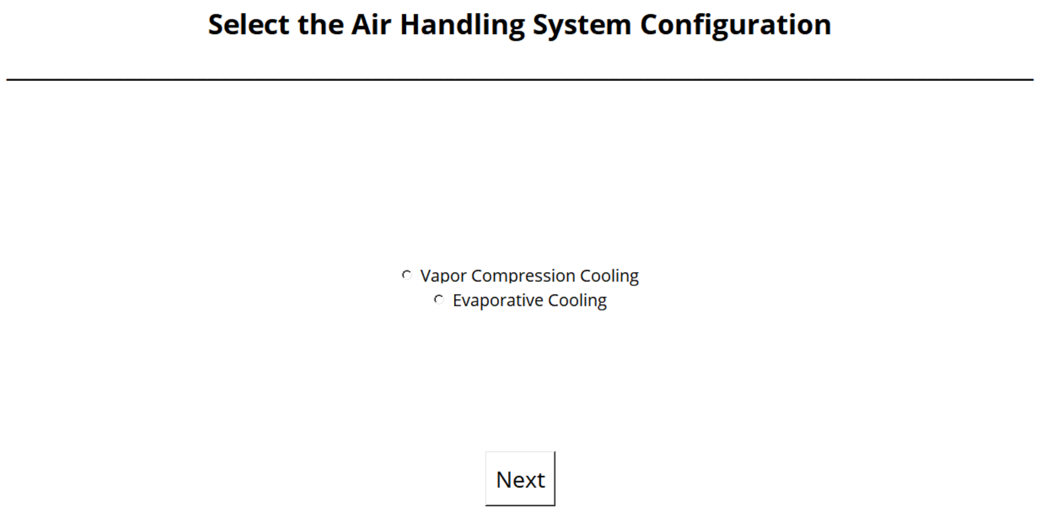
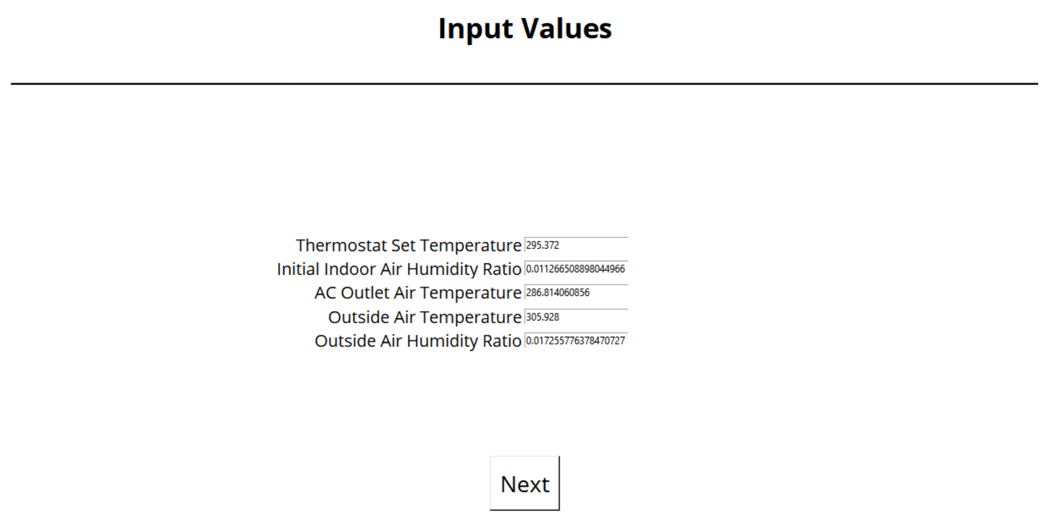

Select the Air Handling System Configuration
Vapor Compression Cooling
Evaporative Cooling
Next

Fig. A.4: Page two of the GUI – configuration selection

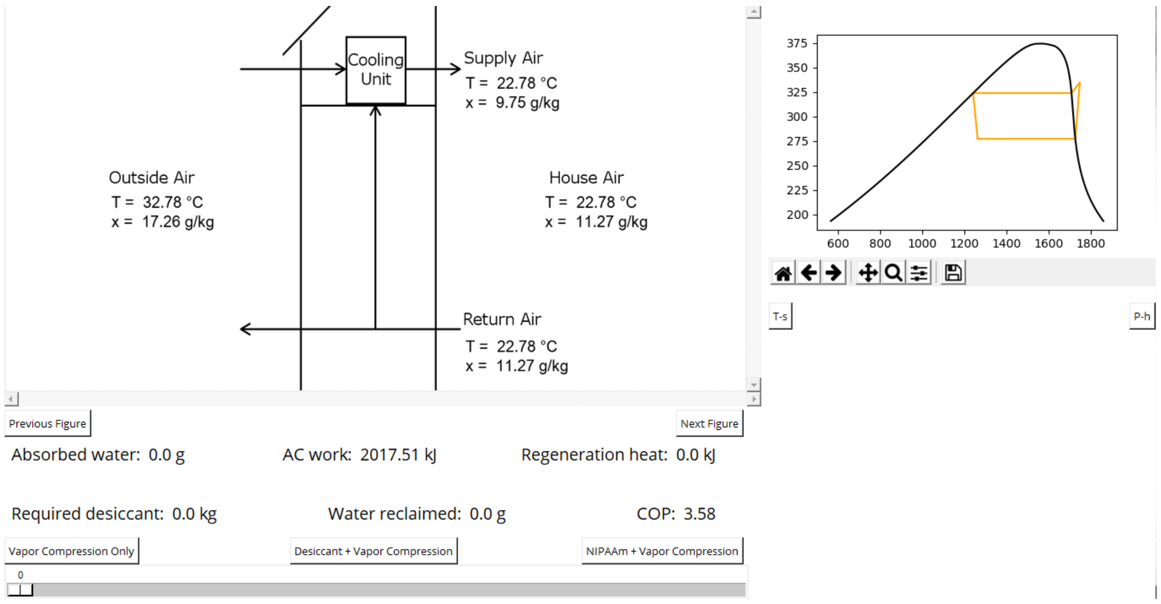
After selecting the cooling method and advancing to the next page, the user is prompted to input several values: the thermostat set temperature, initial indoor humidity ratio, desired temperature and humidity ratio of the cool air supplied to the house, and outdoor air temperature and humidity ratio. All temperatures are to be input in Kelvin and all humidity ratios are to be input in kg/kg. The figure below shows this page of the GUI.


Input Values

Thermostat Set Temperature	295.372
Initial Indoor Air Humidity Ratio	0.011266508898044966
AC Outlet Air Temperature	286.814060856
Outside Air Temperature	305.928
Outside Air Humidity Ratio	0.017255776378470727

Fig. A.5: Page three of the GUI – inputs

After inputting the values and clicking the "Next" button, the GUI script passes the inputs to the next appropriate script; if vapor compression cooling was selected, the inputs are passed to house_air.py, and if evaporative cooling was selected, the inputs are passed to house_air_evap_cool.py. Within the appropriate script, several other scripts are run, and the processes at each component are modeled, along with the transient temperature and humidity response of the air within the house. Additionally, several arrays of graphics are created in this process, which include system diagrams and psychrometric charts. After the scripts are finished running, the final page is displayed. The figure below shows an example of the page that is displayed for the vapor compression cooling configurations.

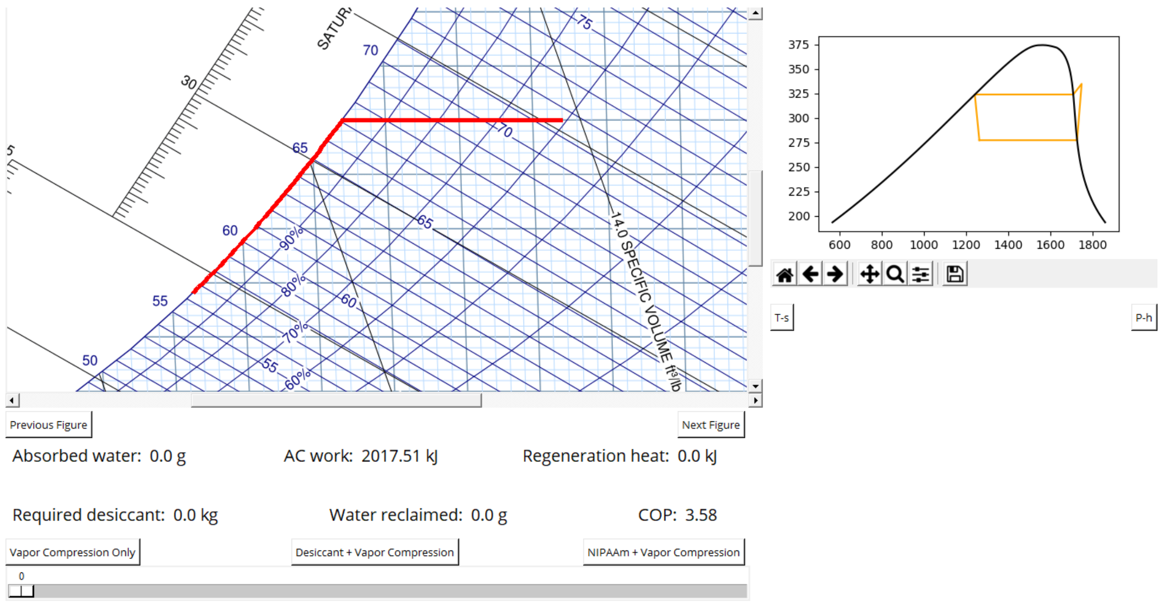


Absorbed water: 0.0 g AC work: 2017.51 kJ Regeneration heat: 0.0 kJ

Required desiccant: 0.0 kg Water reclaimed: 0.0 g COP: 3.58

Fig. A.6: Page four of the GUI with vapor compression configuration selected

Several key features are displayed on the page: the main graphic, the live plots, important values, the configuration selection buttons, and the time slider. The default graphic in the main frame is the system schematic with temperatures and humidity ratios printed on the image. The user can click and drag the slider at the bottom of the window to adjust the system time and watch the house air temperature and humidity ratio change. Additionally, the user can click the “Next figure” button to switch the graphic in the main frame to a psychrometric chart of the steady state processes performed by the total system, as shown in the figure below.

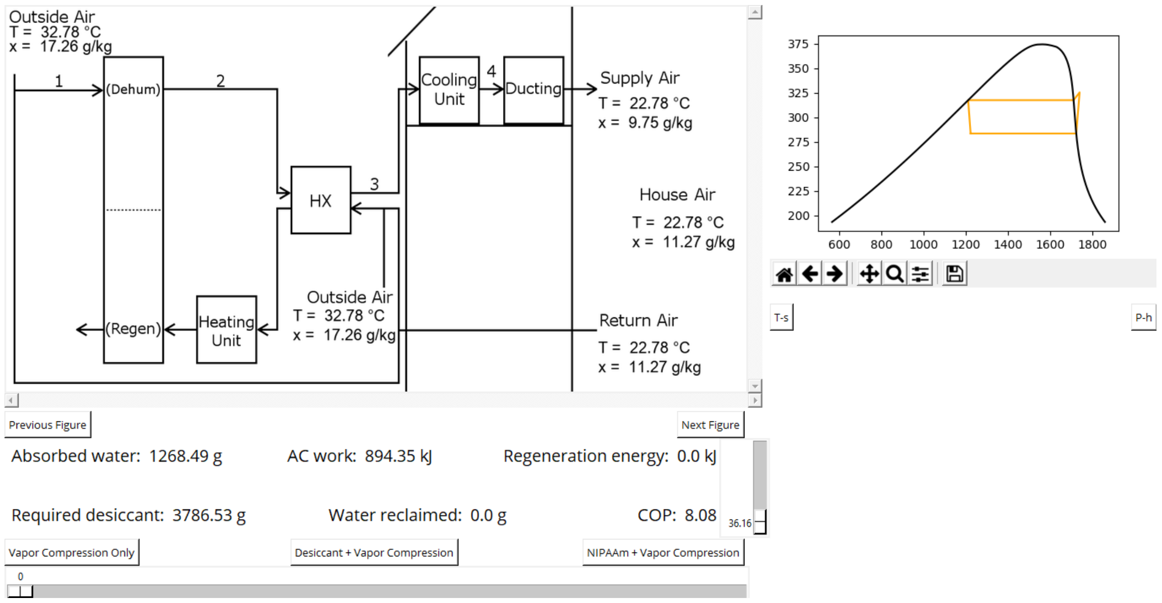

Previous Figure
Next Figure
T-s
P-h
Vapor Compression Only
Desiccant + Vapor Compression
NIPAAm + Vapor Compression
14.0 SPECIFIC VOLUME ft³/lb
SATUR.

Absorbed water: 0.0 g
AC work: 2017.51 kJ
Regeneration heat: 0.0 kJ
Required desiccant: 0.0 kg
Water reclaimed: 0.0 g
COP: 3.58

Fig. A.7: Page four of the GUI with the psychrometric chart selected as the main figure

The psychrometric chart was found on linric.com [11]. When viewing the psychrometric chart, the user can click the "Previous figure" button to switch the main figure back to the system schematic. The live plots to the right of the main figure are T-s and P-h diagrams of the vapor compression refrigeration cycle used to cool the air. By default, the T-s diagram is shown when the page loads, but the user can switch between the two diagrams by clicking the "T-s" and "P-h" buttons.

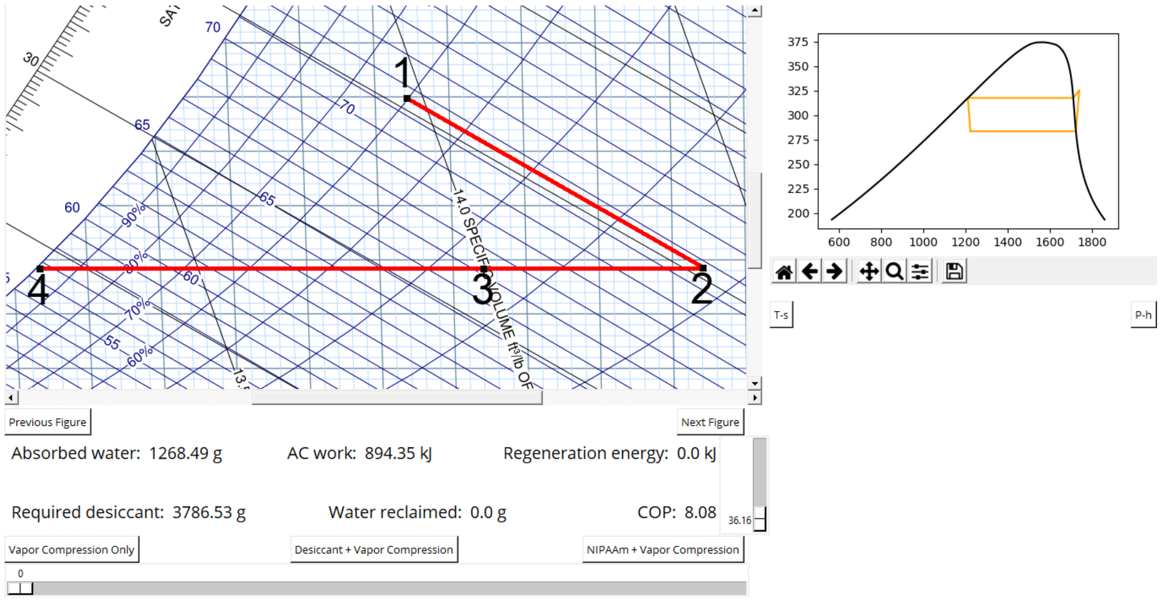
Additionally, the user can click one of the buttons at the bottom of the page to change the system configuration. The default configuration shown when the page loads is the "Vapor Compression Only" configuration. By clicking the "Desiccant + Vapor Compression" button, the page changes to that which is shown in the figure below.



Absorbed water: 1268.49 g
AC work: 894.35 kJ
Regeneration energy: 0.0 kJ
Required desiccant: 3786.53 g
Water reclaimed: 0.0 g
COP: 8.08

Fig. A.8: Page four of the GUI with the desiccant dehumidification and vapor compression cooling configuration selected

The dehumidification process was modeled to bring the humidity of the supply air to the same humidity that was achieved with the vapor compression only configuration, and the vapor compression cooling for this configuration was modeled to bring the outlet temperature of the supply air to the same temperature as the supply air in the vapor compression only configuration. Thus, although the intermediate processes differ, the air entering and exiting the system for this configuration is the same as the air in the vapor compression only configuration. Because of this, the time it takes for the house to reach the desired temperature is the same as in the previous configuration. Because dehumidification and pre-cooling occurs before the air is sent to the vapor compression air conditioner in this configuration, less cooling is required to achieve the same outlet temperature, and the plots on the right of the page change. Additionally, because a different process is taken to reach the supply air outlet properties, the psychrometric chart is different for this configuration than it was for the vapor compression only configuration, as shown in the figure below.



Absorbed water: 1268.49 g

AC work: 894.35 kJ

Regeneration energy: 0.0 kJ

Required desiccant: 3786.53 g

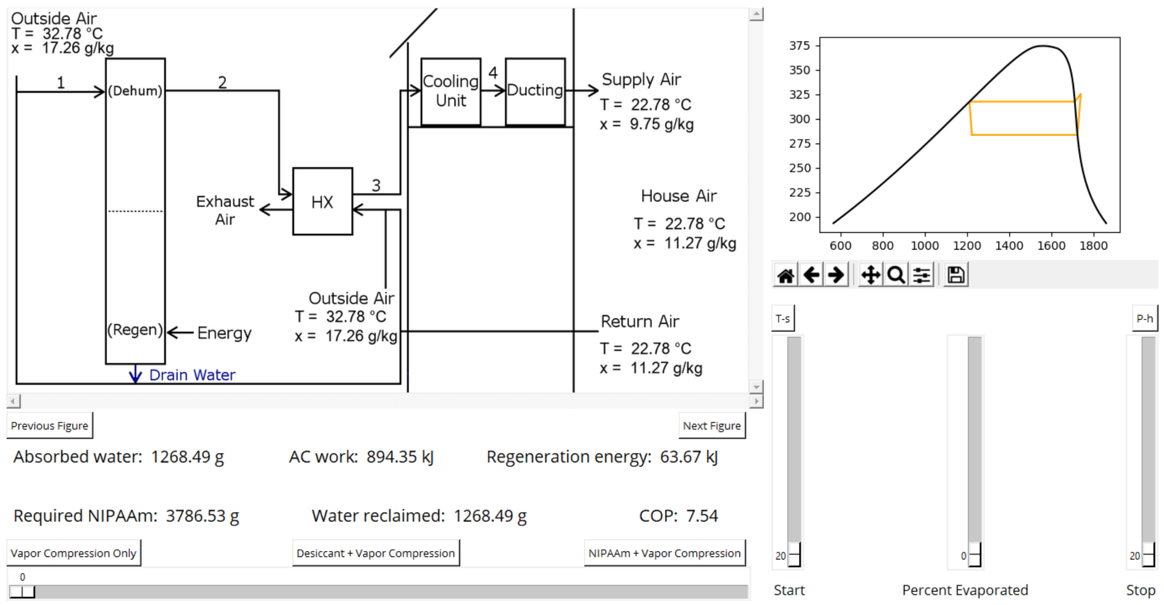
Water reclaimed: 0.0 g

COP: 8.08

Fig. A.9: Page four of the GUI with the psychrometric chart as the main figure and the desiccant dehumidification and vapor compression cooling configuration selected

The vertical slider that appears when changing to the desiccant configuration allows the user to control the regeneration temperature of the desiccant. Adjusting this slider will change the regeneration energy and COP values.

By clicking the “NIPAAm + Vapor Compression” button, the page changes again, as shown in the figure below.



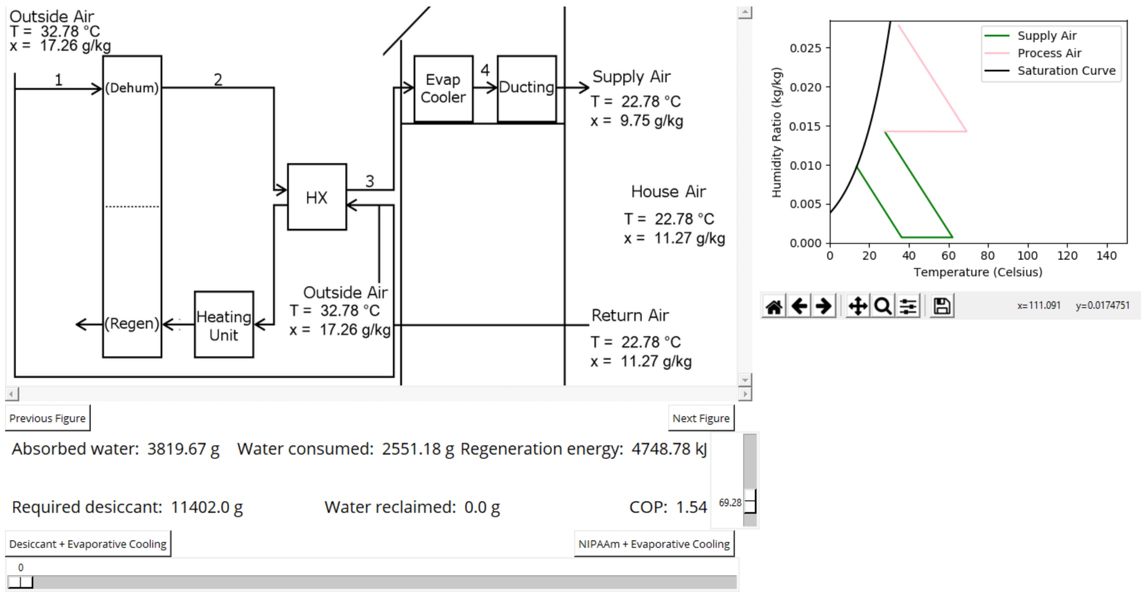
Absorbed water: 1268.49 g AC work: 894.35 kJ Regeneration energy: 63.67 kJ

Required NIPAAm: 3786.53 g Water reclaimed: 1268.49 g COP: 7.54

Fig. A.10: Page four of the GUI with the NIPAAm dehumidification and vapor compression cooling configuration selected

Because the steady state dehumidification process for this configuration is the same as in the previous configuration, the psychrometric chart and vapor compression plots for this configuration are the same as in the desiccant configuration. The only process that differs between the desiccant and NIPAAm configurations is regeneration, so the system schematic and vertical sliders change, along with the regeneration heat and COP values. For this configuration, the user can manipulate the three sliders shown in the figure above. The first slider, labeled "Start", is the temperature at which the NIPAAm leaves the cooling portion and enters the dehumidification portion of the wheel, $T_{NIPAAm,i}$. The lower this temperature is required to be, the more energy is required for cooling. The second slider allows the user to adjust the percent of water that is evaporated during regeneration. As more water is evaporated during regeneration, more heat must be transferred to the wheel to maintain the NIPAAm at the LCST. Finally, the third slider, labeled "Stop", controls the temperature at which the NIPAAm leaves the dehumidification portion and enters the regeneration portion of the wheel, $T_{NIPAAm,f}$. This value should be as close as possible to the LCST without exceeding it. The closer the temperature is to the LCST, the less the temperature has to be raised with external heating to allow it to regenerate; however, the NIPAAm cannot exceed this temperature in the dehumidification portion, otherwise it will transition to hydrophobic and stop dehumidifying the air.

If the evaporative cooling method is selected on the second page, instead of vapor compression cooling, the user is prompted with the same inputs on the third page, after which the GUI script sends the inputs to house_air_evap_cool.py. The script calls the appropriate functions and determines the processes for the evaporative cooling configurations. After the simulation is complete, the final page is displayed, as shown in the figure below.



Absorbed water: 3819.67 g Water consumed: 2551.18 g Regeneration energy: 4748.78 kJ

Required desiccant: 11402.0 g Water reclaimed: 0.0 g COP: 1.54

Fig. A.11: Page four of the GUI for the evaporative cooling configurations

For the evaporative cooling method, there are only two configurations: desiccant dehumidification and evaporative cooling and NIPAAm dehumidification and evaporative cooling. The desiccant configuration is the default when the page loads. The layout of this page is very similar to the layout seen in the desiccant dehumidification and vapor compression cooling page. The system schematic is slightly changed to reflect the evaporative cooler, but the horizontal time slider and vertical regeneration temperature slider function in the same manner as they do in the desiccant dehumidification and vapor compression cooling display. Because there is no vapor compression air conditioner in this cycle, the T-s and P-h diagrams are replaced with a plot of humidity ratio vs temperature. The axes of this plot are the same as in the psychrometric chart, which can still be seen by clicking the "Next figure" button; however, the live plot on the right side of the page are simplified and do not contain the extra information that is shown on the psychrometric chart, such as the lines of constant humidity and enthalpy. Additionally, the plot on the right has units of kg/kg for humidity ratio and °C for temperature, while the psychrometric chart has units of grains/lb. for humidity ratio and °F for temperature. The plot on the right can be seen as a basic illustration of the process the supply air undergoes, while the psychrometric chart is more complex but provides more information and insight if the user understands how to read it. Furthermore, the plot on the right also displays a curve for the process air as it is pre-heated by the heat exchanger, heated further by the electric heater, and then humidified by the regenerating portion of the desiccant wheel as it absorbed moisture from the desiccant. To ensure the desorption in the regenerating portion occurs at the same rate as the absorption in the dehumidifying portion, the increase in process air humidity ratio across the regenerating portion is the same as the supply air humidity ratio drop across the dehumidifying portion. As the user moves the vertical slider, which controls the regeneration temperature, the plot on the right changes to reflect the new regeneration temperature, as shown in the figure below.

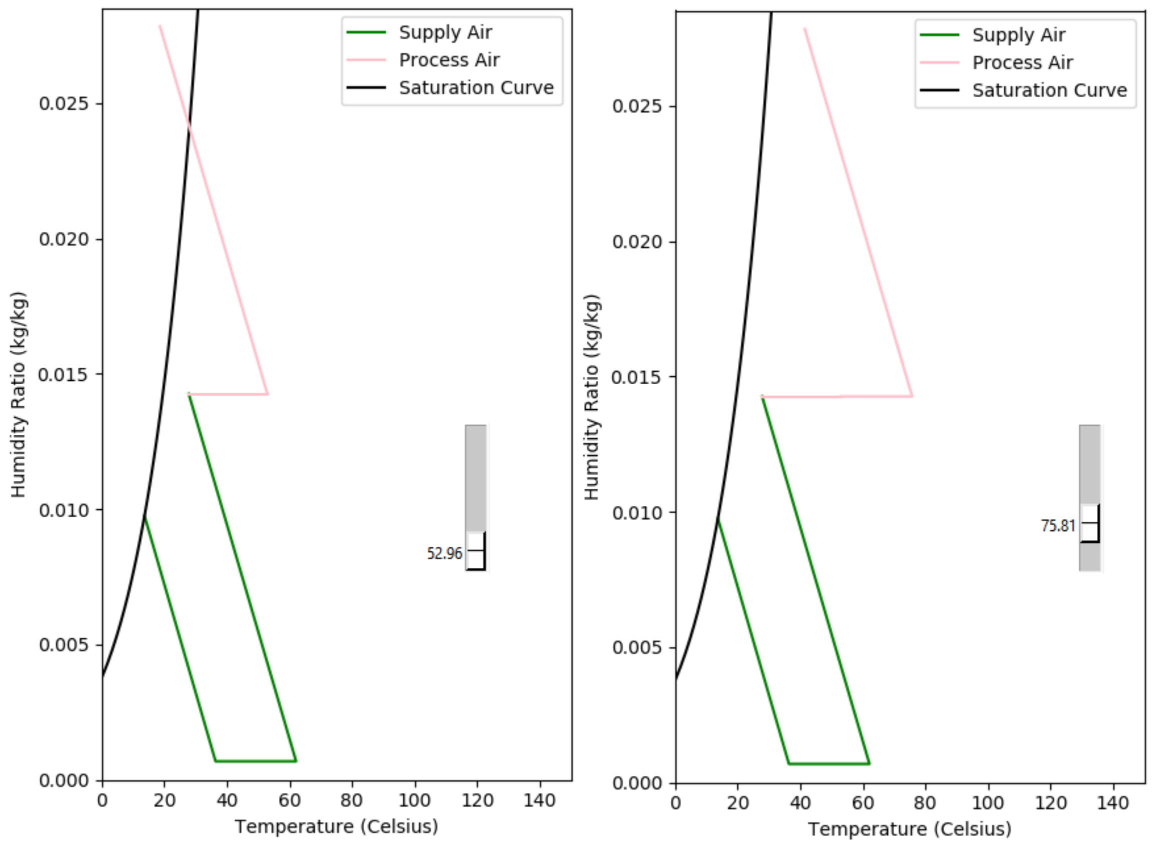
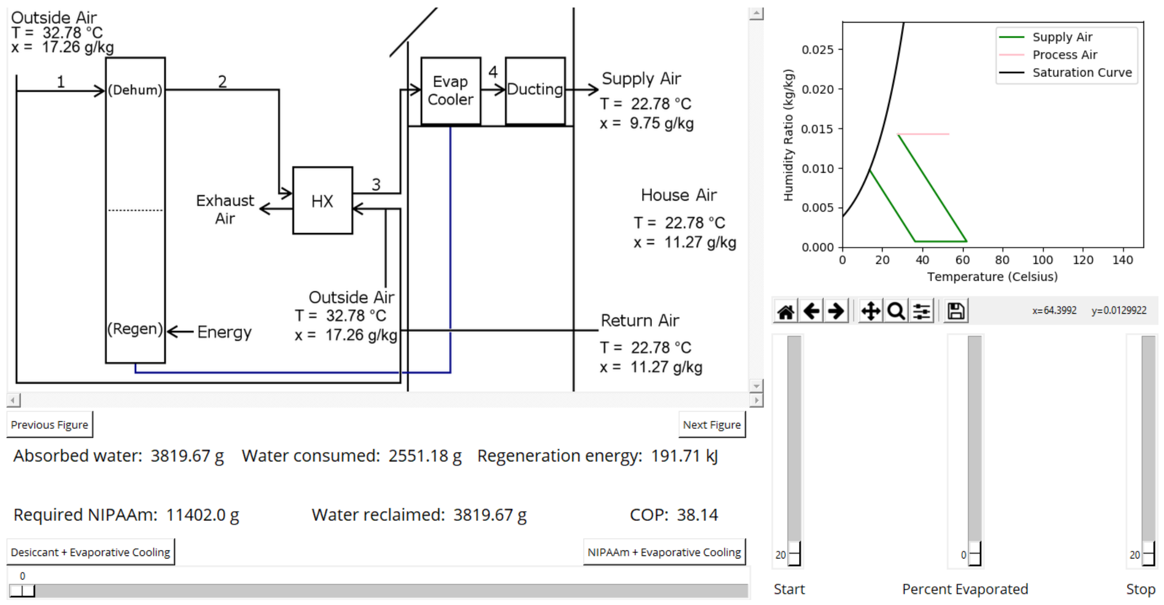


Fig. A.12: Humidity ratio vs temperature GUI plot changing with regeneration temperature. Plots are shown for a regeneration temperature that is impossible (left) and possible (right) based on the position of the process air outlet temperature relative to the saturation curve. The inset is the slider that controls regeneration temperature.

By clicking the “NIPAAm + Evaporative Cooling” button, the user display changes to reflect the NIPAAm configuration, as shown in the figure below.



Absorbed water: 3819.67 g Water consumed: 2551.18 g Regeneration energy: 191.71 kJ

Required NIPAAm: 11402.0 g Water reclaimed: 3819.67 g COP: 38.14

Fig. A.13: Page four of the GUI with the NIPAAm dehumidification and evaporative cooling configuration selected

The system schematic changes to reflect the different method of regeneration, and the vertical sliders change to once gain control the NIPAAm temperature at the beginning and end of the dehumidification portion of the NIPAAm wheel, as well as the evaporation fraction. Additionally, the process air curve is different on the plot to the right, as the NIPAAm is regenerated directly with a heater, and the process air is vented after passing through the heat exchanger.

All of the following values relate to the transient model of the air within the conditioned space, which, again, does not affect the results in Chapter 3; these values are simply presented for clarity. A 1200 sq. ft floor area was selected for the conditioned space that was modeled, and a 3 ton air conditioning system was selected to cool this space. Based on the floor space and an average ceiling height of 8 ft, the total volume of the space was set to 271.84 m^3. The air pressure within the space was set to 101325 Pa. The solids in the space were set to take up 0.5% of the total volume, as this allowed the system operation time to remain low enough to use the stratified air model. The specific heat of the solid was defined as 903600 J/m^{3}K, as this is roughly the volumetric specific heat of wood [19].

The internal heat transfer coefficient for the duct, specific heat for the duct material, and duct mass per unit length were all taken from a study on the dynamic modeling of an HVAC system [20]. The length of the ducting was set to 9.14 m, which is approximately 30 ft, such that the ducting could span the width of a 40 ft x 30 ft room.

The time step for the transient model was set to 1 s. As the percent solid volume decreases, the average room temperature changes at a greater rate, and a smaller time step may be desired. However, if the time step is too small for a given percentage of solid volume, the Python lists will become too large, and the computer on which the model is running will be likely to crash.

APPENDIX B

PYTHON CODE FOR “GUI.PY”

```
from house_air import house_air
from house_air_evap_cool import house_air_evap_cool
import numpy as np
from x_s import x_s
from T_s import T_s
from Tkinter import *
from PIL import Image, ImageTk, ImageFont, ImageDraw
import matplotlib
matplotlib.use('TkAgg')

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2TkAgg
# implement the default mpl key bindings
from matplotlib.backend_bases import key_press_handler


from matplotlib.figure import Figure
print_query = 'no'
fnt = ImageFont.truetype("C:\Windows\Fonts\ARIAL.TTF", 25)
window = Tk()

window.title("Air Conditioned House Model")
w = window.winfo_screenwidth()
h = window.winfo_screenheight() - 80
window.geometry('%dx%d+0+0' % (w, h))
#window.geometry('961x418')

window.configure(background='white')

window_1_title = Label(window, bg='white', text='Air Conditioning Modeling',
font=("Open Sans Bold", 25))
window_1_title.place(x=w/2., y=0, anchor="n")

window_1_underline = Label(window, bg='white',
text='_______________________________________________________________________________
_______________________', font=("Open Sans Bold", 25))
window_1_underline.place(x=w/2., y=50, anchor="n")

v = IntVar()
#v.set(1)

window_1_body = Label(window, bg='white', text='This is a graphical user interface that
can be used to model various air conditioning system configurations. The components are
modeled for steady state performance, and the user can scroll through to watch how the
average temperature and humidity within the house change. The models in this interface
rely on certain approximations and best-case assumptions and are only intended to show
the potential of various hypothetical system configurations.', font=("Open Sans", 15),
wraplength=w*7./8.)
window_1_body.place(relx=0.5, rely=0.5, anchor=CENTER)




def third_window():
        window_2_title = Label(window, bg='white', text='Select the Air Handling
System Configuration', font=("Open Sans Bold", 25))
        window_2_title.place(x=w/2., y=0, anchor="n")
        window_2_underline = Label(window, bg='white',
text='__________________________________________________________________
_____________________', font=("Open Sans Bold", 25))
        window_2_underline.place(x=w/2., y=50, anchor="n")
        v = IntVar()
        #v.set(1)
        Radiobutton(window, bg='white', text='Vapor Compression Cooling',
font=("Open Sans", 15), padx=20, variable=v, value=1).place(relx=0.5, rely=0.46,
anchor=CENTER) #VC
        Radiobutton(window, bg='white', text='Evaporative Cooling', font=("Open Sans",
15), padx=20, variable=v, value=2).place(relx=0.5, rely=0.5, anchor=CENTER) #Evap
        def fourth_window():
                if v.get() == 1:
                        window_5_title = Label(window, bg='white', text='Input Values',
font=("Open Sans Bold", 25))
                        window_5_title.place(x=w/2., y=0, anchor="n")

                        window_5_underline = Label(window, bg='white',
text='__________________________________________________________________
_____________________', font=("Open Sans Bold", 25))
                        window_5_underline.place(x=w/2., y=50, anchor="n")
                        text_1 = Label(window, bg='white', text='Thermostat Set
Temperature', font=("Open Sans", 15))
                        text_1.place(relx=0.5, rely=0.40, anchor="e")
                        text_2 = Label(window, bg='white', text='Initial Indoor Air
Humidity Ratio', font=("Open Sans", 15))
                        text_2.place(relx=0.5, rely=0.44, anchor="e")
                        text_3 = Label(window, bg='white', text='Outside Air
Temperature', font=("Open Sans", 15))
                        text_3.place(relx=0.5, rely=0.52, anchor="e")
```

```
                    text_4 = Label(window, bg='white', text='Outside Air Humidity
Ratio', font=("Open Sans", 15))
                    text_4.place(relx=0.5, rely=0.56, anchor="e")
                    text_5 = Label(window, bg='white', text='AC Outlet Air
Temperature', font=("Open Sans", 15))
                    text_5.place(relx=0.5, rely=0.48, anchor="e")
                    text_5 = Label(window, bg='white', text='Percent Supply Air from
Outside', font=("Open Sans", 15))
                    text_5.place(relx=0.5, rely=0.6, anchor="e")
                    txt_1 = Entry(window,width=20)
                    txt_1.insert(INSERT,"294.59444")
                    txt_1.place(relx=0.5, rely=0.40, anchor=W)
                    txt_2 = Entry(window,width=20)
                    txt_2.insert(INSERT,"0.008164972225996216")
                    txt_2.place(relx=0.5, rely=0.44, anchor=W)
                    txt_3 = Entry(window,width=20)
                    txt_3.insert(INSERT,"298.15")
                    txt_3.place(relx=0.5, rely=0.52, anchor=W)
                    txt_4 = Entry(window,width=20)
                    txt_4.insert(INSERT,"0.02009")
                    txt_4.place(relx=0.5, rely=0.56, anchor=W)
                    txt_5 = Entry(window,width=20)
                    txt_5.insert(INSERT,"284.15")
                    txt_5.place(relx=0.5, rely=0.48, anchor=W)
                    txt_7 = Entry(window,width=20)
                    txt_7.insert(INSERT,"50")
                    txt_7.place(relx=0.5, rely=0.6, anchor=W)

                    def fifth_window():
                        #Vapor Compression
                        global count
                        global count_2
                        global canvas_image
                        global main_figure
                        global main_frame
                        global main_canvas
                        global w_frame
                        global h_frame
                        global vbar
                        global hbar
                        global val_save
                        global T_set
                        global x_i
                        global T_outside
```

```
            global x_outside
            global T_air_o
            global Sb
            global mass_H2O
            global work_AC
            global heat_regen
            global W_AC
            global W_des
            global W_NIPAAm
            global Q_regen_NIPAAm
            global h_fg_NIPAAm
            global m_NIPAAm
            global delta_m_h2o
            global des_amount
            global COP
            global sub_frame_1
            global sub_frame_2
            global t_f
            global delta_t
            global T_HX_preheat_o
            global C_p_regen
            global mass_H2O_reclaimed
            global count_3
            global c_p_NIPAAm_dry
            global c_p_NIPAAm_wet
            global T_regen_NIPAAm
            global COP_AC_NIPAAm
            global m_des
            global percent_vent
            main_figure_1 = []
            [delta_t, t_f, T_h_array_AC, x_h_array_AC,
T_duct_array_AC, x_duct_array_AC, T_return_array_AC, x_return_array_AC,
T_h_array_des, x_h_array_des, T_duct_array_des, x_duct_array_des,
T_return_array_des, x_return_array_des, T_h_array_NIPAAm, x_h_array_NIPAAm,
T_duct_array_NIPAAm, x_duct_array_NIPAAm, T_return_array_NIPAAm,
x_return_array_NIPAAm, delta_m_h2o, W_AC, W_des, W_NIPAAm,
c_p_NIPAAm_dry, c_p_NIPAAm_wet, h_fg_NIPAAm,
T_1_AC,T_3_AC,T_4_AC,s_1_AC,s_2_AC,s_3_AC,s_4_AC,s_g_AC,P_evap_AC,P_co
nd_AC,h_1_AC,h_2_AC,h_3_AC,h_4_AC,T_1_des,T_3_des,T_4_des,s_1_des,s_2_des,
s_3_des,s_4_des,s_g_des,P_evap_des,P_cond_des,h_1_des,h_2_des,h_3_des,h_4_des,T_
1_NIPAAm,T_3_NIPAAm,T_4_NIPAAm,s_1_NIPAAm,s_2_NIPAAm,s_3_NIPAAm,s
_4_NIPAAm,s_g_NIPAAm,P_evap_NIPAAm,P_cond_NIPAAm,h_1_NIPAAm,h_2_NI
PAAm,h_3_NIPAAm,h_4_NIPAAm,Q_AC_cool,Q_des_cool,Q_NIPAAm_cool,s_array
,T_array,h_array,P_array,T_HX_preheat_o,C_p_regen,Q_useful,omega,m_NIPAAm,CO
P_AC_NIPAAm,m_des,delta_C_NIPAAm] =
house_air(T_set,x_i,T_outside,x_outside,(T_set +
5./9.),x_i,T_outside,x_outside,T_air_o,percent_vent)
            T_regen_NIPAAm = 32 + 273.15
            final_index_1 = len(T_h_array_AC)
            final_index_3 = len(T_h_array_des)
            final_index_5 = len(T_h_array_NIPAAm)
            main_figure_2 = []
            main_figure_3 = []
            main_figure_4 = []
            main_figure_5 = []
            main_figure_6 = []
            img_2 =
Image.open('output\psychrom\psychrom_AC_out.png')
            img_4 =
Image.open('output\psychrom\psychrom_desiccant_out.png')
            img_6 =
Image.open('output\psychrom\psychrom_NIPAAm_out.png')
            for ind in range(final_index_1):
                    img_1 = Image.open('diag_AC_only.png')
                    draw = ImageDraw.Draw(img_1)
                    s = " "
                    seq = ("T = ", str(round(T_h_array_AC[ind] -
273.15, 2)), u'\xb0'"C")
                    T_i_str = s.join(seq)
                    seq = ("T = ", str(round(T_return_array_AC[ind] -
273.15, 2)), u'\xb0'"C")
                    T_r_str = s.join(seq)
                    seq = ("T = ", str(round(T_duct_array_AC[ind] -
273.15, 2)), u'\xb0'"C")
                    T_f_str = s.join(seq)
                    seq = ("x = ",
str(round(x_return_array_AC[ind]*1000, 2)), "g/kg")
                    x_r_str = s.join(seq)
                    seq = ("x = ", str(round(x_h_array_AC[ind]*1000,
2)), "g/kg")
                    x_i_str = s.join(seq)
                    seq = ("x = ",
str(round(x_duct_array_AC[ind]*1000, 2)), "g/kg")
                    x_f_str = s.join(seq)
                    seq = ("T = ", str(round(T_outside - 273.15, 2)),
u'\xb0'"C")
                    T_out_str = s.join(seq)
                    seq = ("x = ", str(round(x_outside*1000, 2)),
"g/kg")
                    x_out_str = s.join(seq)
                    draw.text((875,288), T_i_str, font = fnt, fill =
(0,0,0))
                    draw.text((746,513), T_r_str, font = fnt, fill =
(0,0,0))
                    draw.text((746,102), T_f_str, font = fnt, fill =
(0,0,0))
                    draw.text((875,318), x_i_str, font = fnt, fill =
(0,0,0))
                    draw.text((746,543), x_r_str, font = fnt, fill =
(0,0,0))
                    draw.text((746,132), x_f_str, font = fnt, fill =
(0,0,0))
                    draw.text((171,288), T_out_str, font = fnt, fill =
(0,0,0))
                    draw.text((171,318), x_out_str, font = fnt, fill =
(0,0,0))
                    main_figure_1.append(img_1)


                    main_figure_2.append(img_2)


            for ind in range(final_index_3):
                    img_3 = Image.open('diag.png')
                    draw = ImageDraw.Draw(img_3)
                    s = " "
                    seq = ("T = ", str(round(T_h_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                    T_i_str = s.join(seq)
                    seq = ("T = ", str(round(T_return_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                    T_r_str = s.join(seq)
                    seq = ("T = ", str(round(T_duct_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                    T_f_str = s.join(seq)
                    seq = ("x = ",
str(round(x_return_array_des[ind]*1000, 2)), "g/kg")
                    x_r_str = s.join(seq)
                    seq = ("x = ", str(round(x_h_array_des[ind]*1000,
2)), "g/kg")
                    x_i_str = s.join(seq)
                    seq = ("x = ",
str(round(x_duct_array_des[ind]*1000, 2)), "g/kg")
                    x_f_str = s.join(seq)
                    seq = ("T = ", str(round(T_outside - 273.15, 2)),
u'\xb0'"C")
                    T_out_str = s.join(seq)
                    seq = ("x = ", str(round(x_outside*1000, 2)),
"g/kg")
                    x_out_str = s.join(seq)
                    draw.text((1015,318), T_i_str, font = fnt, fill =
(0,0,0))
                    draw.text((960,513), T_r_str, font = fnt, fill =
(0,0,0))
                    draw.text((960,132), T_f_str, font = fnt, fill =
(0,0,0))
                    draw.text((1015,348), x_i_str, font = fnt, fill =
(0,0,0))
                    draw.text((960,543), x_r_str, font = fnt, fill =
(0,0,0))
                    draw.text((960,162), x_f_str, font = fnt, fill =
(0,0,0))
                    draw.text((5,21), T_out_str, font = fnt, fill = (0,0,0))
                    draw.text((5,43), x_out_str, font = fnt, fill = (0,0,0))
                    draw.text((465,463), T_out_str, font = fnt, fill =
(0,0,0))
                    draw.text((465,493), x_out_str, font = fnt, fill =
(0,0,0))
                    main_figure_3.append(img_3)
                    main_figure_4.append(img_4)
            for ind in range(final_index_5):
                    img_5 = Image.open('diag2.png')
                    draw = ImageDraw.Draw(img_5)
                    s = " "
                    seq = ("T = ", str(round(T_h_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                    T_i_str = s.join(seq)
                    seq = ("T = ", str(round(T_return_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                    T_r_str = s.join(seq)
                    seq = ("T = ", str(round(T_duct_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                    T_f_str = s.join(seq)
                    seq = ("x = ",
str(round(x_return_array_des[ind]*1000, 2)), "g/kg")
```

```
                                        x_r_str = s.join(seq)
                                        seq = ("x = ", str(round(x_h_array_des[ind]*1000, 
2)), "g/kg")
                                        x_i_str = s.join(seq)
                                        seq = ("x = ", 
str(round(x_duct_array_des[ind]*1000, 2)), "g/kg")
                                        x_f_str = s.join(seq)
                                        seq = ("T = ", str(round(T_outside - 273.15, 2)), 
u'\xb0'"C")
                                        T_out_str = s.join(seq)
                                        seq = ("x = ", str(round(x_outside*1000, 2)), 
"g/kg")
                                        x_out_str = s.join(seq)
                                        draw.text((1015,318), T_i_str, font = fnt, fill = 
(0,0,0))
                                        draw.text((960,513), T_r_str, font = fnt, fill = 
(0,0,0))
                                        draw.text((960,132), T_f_str, font = fnt, fill = 
(0,0,0))
                                        draw.text((1015,348), x_i_str, font = fnt, fill = 
(0,0,0))
                                        draw.text((960,543), x_r_str, font = fnt, fill = 
(0,0,0))
                                        draw.text((960,162), x_f_str, font = fnt, fill = 
(0,0,0))
                                        draw.text((5,21), T_out_str, font = fnt, fill = (0,0,0))
                                        draw.text((5,43), x_out_str, font = fnt, fill = (0,0,0))
                                        draw.text((465,463), T_out_str, font = fnt, fill = 
(0,0,0))
                                        draw.text((465,493), x_out_str, font = fnt, fill = 
(0,0,0))
                                        main_figure_5.append(img_5)
                                        main_figure_6.append(img_6)
                                count = 0
                                count_2 = 1
                                main_figure = main_figure_1
                                main = main_figure[0]
                                w_main, h_main = main.size
                                main = main.resize([int(w - h/2. - 145), 
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                w_main, h_main = main.size
                                w_frame = w_main
                                h_frame = h_main
                                s = " "
                                mass_H2O = Label(window, bg='white',
text=s.join(("Absorbed water: ", str(round(0*1000, 2)), "g")), font=("Open Sans", 15))
                                mass_H2O.place(x=5, y=(h_frame + 61), anchor="nw")
                                work_AC = Label(window, bg='white', text=s.join(("AC
work: ", str(round(W_AC/1000., 2)), "kJ")), font=("Open Sans", 15))
                                work_AC.place(x=(w_frame - 35)/2., y=(h_frame + 61),
anchor="n")
                                heat_regen = Label(window, bg='white',
text=s.join(("Regeneration heat: ", str(round(0*1000, 2)), "kJ")), font=("Open Sans", 15))
                                heat_regen.place(x=(w_frame - 30), y=(h_frame + 61),
anchor="ne")
                                mass_H2O_reclaimed = Label(window, bg='white',
text=s.join(("Water reclaimed: ", str(round(0*1000, 2)), "g")), font=("Open Sans", 15))
                                mass_H2O_reclaimed.place(x=(w_frame + 100)/2.,
y=(h_frame + 131), anchor="n")
                                des_amount = Label(window, bg='white',
text=s.join(("Required desiccant: ", str(round(0*delta_m_h2o*1000./0.4, 2)), "g")),
font=("Open Sans", 15))
                                des_amount.place(x=5, y=(h_frame + 131), anchor="nw")
                                COP = Label(window, bg='white', text=s.join(("COP: ",
str(round((Q_AC_cool/(W_AC + 0)), 2)))), font=("Open Sans", 15))
                                COP.place(x=(w_frame - 30), y=(h_frame + 131),
anchor="ne")
                                main = ImageTk.PhotoImage(main)
                                main.image = main

        main_frame=Frame(window,width=w_main,height=h_main)
                                main_frame.grid(row=0,column=0)

        main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))
                                def AC_only_button():
                                        global count
                                        global count_2
                                        global main_figure
                                        global canvas_image
                                        global main_frame
                                        global main_canvas
                                        global w_frame
                                        global h_frame
                                        global vbar
                                        global hbar
                                        global mass_H2O
                                        global work_AC
global heat_regen
global W_AC
global des_amount
global COP
global t_f
global delta_t
global mass_H2O_reclaimed
global GUI_txt_1
global GUI_txt_2
global GUI_txt_3
#global canvas_image
if count_2 == 1:
        1
else:
        global Sb_2
        if count_2 == 3:
                Sb_2.destroy()
                Sb_3.destroy()
                Sb_4.destroy()
                GUI_txt_1.destroy()
                GUI_txt_2.destroy()
                GUI_txt_3.destroy()
        elif count_2 == 2:
                Sb_2.destroy()
        else:
                1
        global val_save
        global Sb
        global sub_frame_1
        global sub_frame_2
        count = 0
        count_2 = 1
        count_3 = 1
        main_canvas.destroy()
        vbar.destroy()
        hbar.destroy()
        main_figure = main_figure_1
        if val_save > (final_index_1 - 1):
                val_save = final_index_1 - 1
        else:
                1
        main = main_figure[val_save]
        w_main, h_main = main.size
```

```
                    main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                    w_main, h_main = main.size
                    main = ImageTk.PhotoImage(main)
                    main.image = main

    main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

    hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                    hbar.pack(side=BOTTOM,fill=X)
                    hbar.config(command=main_canvas.xview)

    vbar=Scrollbar(main_frame,orient=VERTICAL)
                    vbar.pack(side=RIGHT,fill=Y)
                    vbar.config(command=main_canvas.yview)

    main_canvas.config(width=w_main,height=h_main)

    main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

    main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                    canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                    mass_H2O.destroy()
                    work_AC.destroy()
                    heat_regen.destroy()
                    des_amount.destroy()
                    COP.destroy()
                    mass_H2O_reclaimed.destroy()
                    s = " "
                    mass_H2O = Label(window, bg='white',
text=s.join(("Absorbed water: ", str(round(0*1000, 2)), "g")), font=("Open Sans", 15))
                    mass_H2O.place(x=5, y=(h_frame + 61),
anchor="nw")
                    work_AC = Label(window, bg='white',
text=s.join(("AC work: ", str(round(W_AC/1000., 2)), "kJ")), font=("Open Sans", 15))
                    work_AC.place(x=(w_frame - 35)/2.,
y=(h_frame + 61), anchor="n")
                    heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round(0/1000., 2)), "kJ")), font=("Open Sans",
15))
                    heat_regen.place(x=(w_frame - 30),
y=(h_frame + 61), anchor="ne")
```

```
                                        mass_H2O_reclaimed = Label(window,
bg='white', text=s.join(("Water reclaimed: ", str(round(0*1000, 2)), "g")), font=("Open
Sans", 15))
                                        mass_H2O_reclaimed.place(x=(w_frame +
100)/2., y=(h_frame + 131), anchor="n")
                                        des_amount = Label(window, bg='white',
text=s.join(("Required desiccant: ", str(round(0*delta_m_h2o*1000./0.4, 2)), "g")),
font=("Open Sans", 15))
                                        des_amount.place(x=5, y=(h_frame + 131),
anchor="nw")
                                        COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_AC_cool/(W_AC + 0)), 2)))), font=("Open Sans",
15))
                                        COP.place(x=(w_frame - 30), y=(h_frame +
131), anchor="ne")
                                        Sb.destroy()
                                        Sb =
Scale(window,orient=HORIZONTAL,bg='white',from_=0,to=t_f,command=slider,length
=w_frame,resolution=delta_t)
                                        Sb.set(val_save*delta_t)
                                        Sb.place(relx=0,rely=1,anchor='sw')
                                        sub_frame_1.destroy()
                                        sub_frame_1=Frame(window,width=(w -
w_frame - 30),height=h/2.,bg='white')

        sub_frame_1.place(relx=1,rely=0,anchor='ne')

                                        f = Figure(figsize=((w - w_frame - 30)/100.,
h/200), dpi=100)
                                        a = f.add_subplot(111)

                                        a.plot([s_1_AC, s_2_AC, s_3_AC, s_g_AC,
s_4_AC, s_1_AC],[T_1_AC, T_1_AC, T_3_AC, T_4_AC, T_4_AC, T_1_AC],
label='Refrigerant', color='orange')
                                        a.plot(s_array,T_array, color='k')

                                        # a tk.DrawingArea
                                        canvas = FigureCanvasTkAgg(f,
master=sub_frame_1)
                                        canvas.show()
                                        canvas.get_tk_widget().pack(side=TOP,
expand=0)
```

```
                                                toolbar =
NavigationToolbar2TkAgg(canvas, sub_frame_1)
                                                toolbar.update()
                                                canvas._tkcanvas.pack(side=TOP,
expand=0)

                                Button(window, text='Vapor Compression Only',
command=AC_only_button, bg='white', font=("Open Sans", 10)).place(x=0, y=(h - 25),
anchor='sw') #AC Only button

                                def des_button():
                                        global count
                                        global count_2
                                        global main_figure
                                        global canvas_image
                                        global main_frame
                                        global main_canvas
                                        global w_frame
                                        global h_frame
                                        global vbar
                                        global hbar
                                        global mass_H2O
                                        global work_AC
                                        global heat_regen
                                        global W_des
                                        global des_amount
                                        global COP
                                        global t_f
                                        global delta_t
                                        global mass_H2O_reclaimed
                                        global GUI_txt_1
                                        global GUI_txt_2
                                        global GUI_txt_3
                                        #global canvas_image
                                        if count_2 == 2:
                                                1
                                        else:
                                                global Sb_2
                                                if count_2 == 3:
                                                        Sb_2.destroy()
                                                        Sb_3.destroy()
                                                        Sb_4.destroy()
                                                        GUI_txt_1.destroy()
                                                        GUI_txt_2.destroy()
```

```
                                                  GUI_txt_3.destroy()
                                        else:
                                                  1
                                        global val_save
                                        global Sb
                                        global sub_frame_1
                                        global sub_frame_2
                                        global m_des
                                        count = 0
                                        count_2 = 2
                                        count_3 = 1
                                        main_canvas.destroy()
                                        vbar.destroy()
                                        hbar.destroy()
                                        main_figure = main_figure_3
                                        main = main_figure[val_save]
                                        w_main, h_main = main.size
                                        main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                        w_main, h_main = main.size
                                        main = ImageTk.PhotoImage(main)
                                        main.image = main

        main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

        hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                        hbar.pack(side=BOTTOM,fill=X)
                                        hbar.config(command=main_canvas.xview)

        vbar=Scrollbar(main_frame,orient=VERTICAL)
                                        vbar.pack(side=RIGHT,fill=Y)
                                        vbar.config(command=main_canvas.yview)

        main_canvas.config(width=w_main,height=h_main)

        main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

        main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                        mass_H2O.destroy()
                                        work_AC.destroy()
                                        heat_regen.destroy()
```

```
                                                des_amount.destroy()
                                                COP.destroy()
                                                mass_H2O_reclaimed.destroy()
                                                s = " "
                                                mass_H2O = Label(window, bg='white',
text=s.join(("Absorbed water: ", str(round(delta_m_h2o*1000, 2)), "g")), font=("Open
Sans", 15))
                                                mass_H2O.place(x=5, y=(h_frame + 61),
anchor="nw")
                                                mass_H2O_reclaimed = Label(window,
bg='white', text=s.join(("Water reclaimed: ", str(round(0*1000, 2)), "g")), font=("Open
Sans", 15))
                                                mass_H2O_reclaimed.place(x=(w_frame +
100)/2., y=(h_frame + 131), anchor="n")
                                                work_AC = Label(window, bg='white',
text=s.join(("AC work: ", str(round(W_des/1000., 2)), "kJ")), font=("Open Sans", 15))
                                                work_AC.place(x=(w_frame-35)/2.,
y=(h_frame + 61), anchor="n")
                                                des_amount = Label(window, bg='white',
text=s.join(("Required desiccant: ", str(round(m_des*1000., 2)), "g")), font=("Open
Sans", 15))
                                                des_amount.place(x=5, y=(h_frame + 131),
anchor="nw")
                                                Sb.destroy()
                                                Sb =
Scale(window,orient=HORIZONTAL,bg='white',from_=0,to=t_f,command=slider,length
=w_frame,resolution=delta_t)
                                                Sb.set(val_save*delta_t)
                                                Sb.place(relx=0,rely=1,anchor='sw')
                                                Sb_2 =
Scale(window,orient=VERTICAL,bg='white',from_=140,to=(T_HX_preheat_o -
273.15),command=slider_des,length=(h/2. - 70),resolution=1)
                                                Sb_2.set(T_HX_preheat_o - 273.15)
                                                Sb_2.place(x=(w_frame + 30 + w)/2.,y=(h -
20),anchor='s')
                                                sub_frame_1.destroy()
                                                sub_frame_1=Frame(window,width=(w -
w_frame - 30),height=h/2.,bg='white')

        sub_frame_1.place(relx=1,rely=0,anchor='ne')

                                                f = Figure(figsize=((w - w_frame - 30)/100.,
h/200), dpi=100)
                                                a = f.add_subplot(111)
```

```
                                        a.plot([s_1_des, s_2_des, s_3_des, s_g_des,
s_4_des, s_1_des],[T_1_des, T_1_des, T_3_des, T_4_des, T_4_des, T_1_des],
label='Refrigerant', color='orange')
                                        a.plot(s_array,T_array, color='k')

                                        # a tk.DrawingArea
                                        canvas = FigureCanvasTkAgg(f,
master=sub_frame_1)
                                        canvas.show()
                                        canvas.get_tk_widget().pack(side=TOP,
expand=0)

                                        toolbar =
NavigationToolbar2TkAgg(canvas, sub_frame_1)
                                        toolbar.update()
                                        canvas._tkcanvas.pack(side=TOP,
expand=0)

                              Button(window, text='Desiccant + Vapor Compression',
command=des_button, bg='white', font=("Open Sans", 10)).place(x=int(w_main/2.), y=(h
- 25), anchor='s') #Desiccant button

                              def NIPAAm_button():
                                   global count
                                   global count_2
                                   global main_figure
                                   global canvas_image
                                   global main_frame
                                   global main_canvas
                                   global w_frame
                                   global h_frame
                                   global vbar
                                   global hbar
                                   global mass_H2O
                                   global work_AC
                                   global heat_regen
                                   global W_NIPAAm
                                   global Q_regen_NIPAAm
                                   global des_amount
                                   global COP
                                   global t_f
                                   global delta_t
                                   global percent_evap
```

```
					global T_start
					global T_stop
					global GUI_txt_1
					global GUI_txt_2
					global GUI_txt_3
					#global canvas_image
					if count_2 == 3:
						1
					else:
						global val_save
						global Sb
						global Sb_2
						global Sb_3
						global Sb_4
						global sub_frame_1
						global sub_frame_2
						global m_NIPAAm
						if count_2 == 2:
							Sb_2.destroy()
						else:
							1
						count = 0
						count_2 = 3
						count_3 = 1
						main_canvas.destroy()
						vbar.destroy()
						hbar.destroy()
						main_figure = main_figure_5
						main = main_figure[val_save]
						w_main, h_main = main.size
						main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
						w_main, h_main = main.size
						main = ImageTk.PhotoImage(main)
						main.image = main

	main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

	hbar=Scrollbar(main_frame,orient=HORIZONTAL)
						hbar.pack(side=BOTTOM,fill=X)
						hbar.config(command=main_canvas.xview)

	vbar=Scrollbar(main_frame,orient=VERTICAL)
```

```
                                        vbar.pack(side=RIGHT,fill=Y)
                                        vbar.config(command=main_canvas.yview)

        main_canvas.config(width=w_main,height=h_main)

        main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

        main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                        mass_H2O.destroy()
                                        work_AC.destroy()
                                        heat_regen.destroy()
                                        des_amount.destroy()
                                        COP.destroy()
                                        s = " "
                                        mass_H2O = Label(window, bg='white',
text=s.join(("Absorbed water: ", str(round(delta_m_h2o*1000, 2)), "g")), font=("Open
Sans", 15))
                                        mass_H2O.place(x=5, y=(h_frame + 61),
anchor="nw")
                                        work_AC = Label(window, bg='white',
text=s.join(("AC work: ", str(round(W_NIPAAm/1000., 2)), "kJ")), font=("Open Sans",
15))
                                        work_AC.place(x=(w_frame-35)/2.,
y=(h_frame + 61), anchor="n")
                                        des_amount = Label(window, bg='white',
text=s.join(("Required NIPAAm: ", str(round(m_NIPAAm*1000., 2)), "g")),
font=("Open Sans", 15))
                                        des_amount.place(x=5, y=(h_frame + 131),
anchor="nw")
                                        Sb.destroy()
                                        Sb =
Scale(window,orient=HORIZONTAL,bg='white',from_=0,to=t_f,command=slider,length
=w_frame,resolution=delta_t)
                                        Sb.set(val_save*delta_t)
                                        Sb.place(relx=0,rely=1,anchor='sw')
                                        percent_evap = 0
                                        T_start = 20 + 273.15
                                        T_stop = 20 + 273.15
                                        Sb_2 =
Scale(window,orient=VERTICAL,bg='white',from_=100,to=0,command=slider_NIPAA
m,length=(h/2. - 70),resolution=1)
                                        Sb_2.set(0)
```

```
                                                    Sb_2.place(x=(w_frame + 30 + w)/2.,y=(h -
20),anchor='s')
                                                    Sb_3 =
Scale(window,orient=VERTICAL,bg='white',from_=(T_regen_NIPAAm -
273.15),to=0,command=slider_NIPAAm_2,length=(h/2. - 70),resolution=1)
                                                    Sb_3.set(0)
                                                    Sb_3.place(relx=1,y=(h - 20),anchor='se')
                                                    Sb_4 =
Scale(window,orient=VERTICAL,bg='white',from_=(T_regen_NIPAAm -
273.15),to=0,command=slider_NIPAAm_3,length=(h/2. - 70),resolution=1)
                                                    Sb_4.set(0)
                                                    Sb_4.place(x=(w_frame + 30),y=(h -
20),anchor='sw')
                                                    GUI_txt_1 = Label(window, bg='white',
text="Start", font=("Open Sans", 12))
                                                    GUI_txt_1.place(x=(w_frame +
30),rely=1,anchor='sw')
                                                    GUI_txt_2 = Label(window, bg='white',
text="Percent Evaporated", font=("Open Sans", 12))
                                                    GUI_txt_2.place(x=(w_frame + 30 +
w)/2.,rely=1,anchor='s')
                                                    GUI_txt_3 = Label(window, bg='white',
text="Stop", font=("Open Sans", 12))
                                                    GUI_txt_3.place(relx=1,rely=1,anchor='se')
                                                    sub_frame_1.destroy()
                                                    sub_frame_1=Frame(window,width=(w -
w_frame - 30),height=h/2.,bg='white')

        sub_frame_1.place(relx=1,rely=0,anchor='ne')

                                                    f = Figure(figsize=((w - w_frame - 30)/100.,
h/200), dpi=100)
                                                    a = f.add_subplot(111)

                                                    a.plot([s_1_NIPAAm, s_2_NIPAAm,
s_3_NIPAAm, s_g_NIPAAm, s_4_NIPAAm, s_1_NIPAAm],[T_1_NIPAAm,
T_1_NIPAAm, T_3_NIPAAm, T_4_NIPAAm, T_4_NIPAAm, T_1_NIPAAm],
label='Refrigerant', color='orange')
                                                    a.plot(s_array,T_array, color='k')

                                                    # a tk.DrawingArea
                                                    canvas = FigureCanvasTkAgg(f,
master=sub_frame_1)
                                                    canvas.show()
```

```
                                                canvas.get_tk_widget().pack(side=TOP,
expand=0)

                                                toolbar =
NavigationToolbar2TkAgg(canvas, sub_frame_1)
                                                toolbar.update()
                                                canvas._tkcanvas.pack(side=TOP,
expand=0)

                                Button(window, text='NIPAAm + Vapor Compression',
command=NIPAAm_button, bg='white', font=("Open Sans", 10)).place(x=w_main, y=(h
- 25), anchor='se') #NIPAAm button


                                hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                hbar.pack(side=BOTTOM,fill=X)
                                hbar.config(command=main_canvas.xview)
                                vbar=Scrollbar(main_frame,orient=VERTICAL)
                                vbar.pack(side=RIGHT,fill=Y)
                                vbar.config(command=main_canvas.yview)
                                main_canvas.config(width=w_main,height=h_main)
                                main_canvas.config(xscrollcommand=hbar.set,
yscrollcommand=vbar.set)
                                main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                val_save = 0
                                def slider(val):
                                        global val_save
                                        global count
                                        global delta_t
                                        main_canvas.delete("all")
                                        main = main_figure[int(float(val)/delta_t)]
                                        w_main, h_main = main.size
                                        if count == 0:
                                                main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                        else:
                                                1
                                        w_main, h_main = main.size
                                        main = ImageTk.PhotoImage(main)
                                        main.image = main
                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
```

```
                                        val_save = int(float(val)/delta_t)
                                Sb =
Scale(window,orient=HORIZONTAL,bg='white',from_=0,to=t_f,command=slider,length
=w_frame,resolution=delta_t)
                                Sb.place(relx=0,rely=1,anchor='sw')
                                def slider_NIPAAm(val):
                                        global m_NIPAAm
                                        global delta_m_h2o
                                        global h_fg_NIPAAm
                                        global heat_regen
                                        global COP
                                        global mass_H2O_reclaimed
                                        global percent_evap
                                        global T_start
                                        global T_stop
                                        global c_p_NIPAAm_dry
                                        global c_p_NIPAAm_wet
                                        global T_regen_NIPAAm
                                        global COP_AC_NIPAAm
                                        global t_f
                                        heat_regen.destroy()
                                        COP.destroy()
                                        mass_H2O_reclaimed.destroy()
                                        percent_evap = float(val)
                                        regeneration_energy =
m_NIPAAm*omega/360.*((T_regen_NIPAAm - T_stop)*c_p_NIPAAm_wet +
delta_C_NIPAAm*percent_evap*h_fg_NIPAAm/100. + (T_regen_NIPAAm -
T_start)*c_p_NIPAAm_dry/COP_AC_NIPAAm)*t_f
                                        heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((regeneration_energy)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                                        heat_regen.place(x=(w_frame - 30), y=(h_frame +
61), anchor="ne")
                                        COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_AC_cool/(W_NIPAAm + regeneration_energy)),
2)))), font=("Open Sans", 15))
                                        COP.place(x=(w_frame - 30), y=(h_frame + 131),
anchor="ne")
                                        mass_H2O_reclaimed = Label(window, bg='white',
text=s.join(("Water reclaimed: ", str(round(delta_m_h2o*(1 - float(val)/100.)*1000, 2)),
"g")), font=("Open Sans", 15))
                                        mass_H2O_reclaimed.place(x=(w_frame + 100)/2.,
y=(h_frame + 131), anchor="n")
                                def slider_NIPAAm_2(val):
```

```
                    global m_NIPAAm
                    global h_fg_NIPAAm
                    global heat_regen
                    global COP
                    global mass_H2O_reclaimed
                    global percent_evap
                    global T_start
                    global T_stop
                    global c_p_NIPAAm_dry
                    global c_p_NIPAAm_wet
                    global T_regen_NIPAAm
                    global COP_AC_NIPAAm
                    heat_regen.destroy()
                    COP.destroy()
                    T_stop = float(val) + 273.15
                    regeneration_energy =
m_NIPAAm*omega/360.*((T_regen_NIPAAm - T_stop)*c_p_NIPAAm_wet +
delta_C_NIPAAm*percent_evap*h_fg_NIPAAm/100. + (T_regen_NIPAAm -
T_start)*c_p_NIPAAm_dry/COP_AC_NIPAAm)*t_f
                    heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((regeneration_energy)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                    heat_regen.place(x=(w_frame - 30), y=(h_frame +
61), anchor="ne")
                    COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_AC_cool/(W_NIPAAm + regeneration_energy)),
2)))), font=("Open Sans", 15))
                    COP.place(x=(w_frame - 30), y=(h_frame + 131),
anchor="ne")
                def slider_NIPAAm_3(val):
                    global m_NIPAAm
                    global h_fg_NIPAAm
                    global heat_regen
                    global COP
                    global mass_H2O_reclaimed
                    global percent_evap
                    global T_start
                    global T_stop
                    global c_p_NIPAAm_dry
                    global c_p_NIPAAm_wet
                    global T_regen_NIPAAm
                    global COP_AC_NIPAAm
                    heat_regen.destroy()
                    COP.destroy()
```

```
                                        T_start = float(val) + 273.15
                                        regeneration_energy =
m_NIPAAm*omega/360.*((T_regen_NIPAAm - T_stop)*c_p_NIPAAm_wet +
delta_C_NIPAAm*percent_evap*h_fg_NIPAAm/100. + (T_regen_NIPAAm -
T_start)*c_p_NIPAAm_dry/COP_AC_NIPAAm)*t_f
                                        heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((regeneration_energy)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                                        heat_regen.place(x=(w_frame - 30), y=(h_frame +
61), anchor="ne")
                                        COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_AC_cool/(W_NIPAAm + regeneration_energy)),
2)))), font=("Open Sans", 15))
                                        COP.place(x=(w_frame - 30), y=(h_frame + 131),
anchor="ne")
                                def slider_des(val):
                                        global T_HX_preheat_o
                                        global C_p_regen
                                        global heat_regen
                                        global COP
                                        global t_f
                                        heat_regen.destroy()
                                        COP.destroy()
                                        if float(val) == float(round((T_HX_preheat_o -
273.15),0)):
                                                heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((C_p_regen*(0)*t_f)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                                                heat_regen.place(x=(w_frame - 30),
y=(h_frame + 61), anchor="ne")
                                                COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_AC_cool/(W_NIPAAm + C_p_regen*(0)*t_f)), 2)))),
font=("Open Sans", 15))
                                                COP.place(x=(w_frame - 30), y=(h_frame +
131), anchor="ne")
                                        else:
                                                heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((C_p_regen*(float(val) + 273.15 -
T_HX_preheat_o)*t_f)/1000., 2)), "kJ")), font=("Open Sans", 15))
                                                heat_regen.place(x=(w_frame - 30),
y=(h_frame + 61), anchor="ne")
                                                COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_AC_cool/(W_NIPAAm + C_p_regen*(float(val) +
273.15 - T_HX_preheat_o)*t_f)), 2)))), font=("Open Sans", 15))
```

```
                                        COP.place(x=(w_frame - 30), y=(h_frame +
131), anchor="ne")
                                def next_fig():
                                        global count
                                        global count_2
                                        global main_figure
                                        global canvas_image
                                        global main_frame
                                        global main_canvas
                                        global w_frame
                                        global h_frame
                                        global vbar
                                        global hbar
                                        #global canvas_image
                                        if count == 1:
                                                1
                                        else:
                                                global val_save
                                                count = count + 1
                                                main_canvas.destroy()
                                                vbar.destroy()
                                                hbar.destroy()
                                                if count_2 == 1:
                                                        main_figure = main_figure_2
                                                elif count_2 == 2:
                                                        main_figure = main_figure_4
                                                elif count_2 == 3:
                                                        main_figure = main_figure_6
                                                else:
                                                        1
                                                main = main_figure[val_save]
                                                w_main, h_main = main.size
                                                #main = main.resize([int(w - h/2. - 25),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                                main = ImageTk.PhotoImage(main)
                                                main.image = main

        main_canvas=Canvas(main_frame,bg='white',width=w_frame,height=h_frame,scr
ollregion=(0,0,w_main,h_main))

        hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                                hbar.pack(side=BOTTOM,fill=X)
                                                hbar.config(command=main_canvas.xview)
```

```
	vbar=Scrollbar(main_frame,orient=VERTICAL)
						vbar.pack(side=RIGHT,fill=Y)
						vbar.config(command=main_canvas.yview)

	main_canvas.config(width=w_frame,height=h_frame)

	main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

	main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
						canvas_image = 
main_canvas.create_image(0,0,image=main, anchor="nw")
				Button(window, text='Next Figure', command=next_fig, 
bg='white', font=("Open Sans", 10)).place(x=w_main, y=(h_main + 25), anchor='ne') 
#next figure button

				def prev_fig():
					global count
					global count_2
					global main_figure
					global main_frame
					global main_canvas
					global canvas_image
					global vbar
					global hbar
					#global canvas_image
					if count == 0:
						1
					else:
						global val_save
						count = count - 1
						main_canvas.destroy()
						vbar.destroy()
						hbar.destroy()
						if count_2 == 1:
							main_figure = main_figure_1
						elif count_2 == 2:
							main_figure = main_figure_3
						elif count_2 == 3:
							main_figure = main_figure_5
						else:
							1
						main = main_figure[val_save]
						w_main, h_main = main.size
```

```
                                        main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                        w_main, h_main = main.size
                                        main = ImageTk.PhotoImage(main)
                                        main.image = main

        main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

        hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                        hbar.pack(side=BOTTOM,fill=X)
                                        hbar.config(command=main_canvas.xview)

        vbar=Scrollbar(main_frame,orient=VERTICAL)
                                        vbar.pack(side=RIGHT,fill=Y)
                                        vbar.config(command=main_canvas.yview)

        main_canvas.config(width=w_main,height=h_main)

        main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

        main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                Button(window, text='Previous Figure',
command=prev_fig, bg='white', font=("Open Sans", 10)).place(x=0, y=(h_main + 25),
anchor='nw') #prev figure button

                                count_3 = 1

                                def Ts_diag():
                                        global main_figure
                                        global main_frame
                                        global main_canvas
                                        global canvas_image
                                        global vbar
                                        global hbar
                                        global count_3
                                        global count_2
                                        global sub_frame_1
                                        global sub_frame_2
                                        #global canvas_image
                                        if count_3 == 1:
                                                1
                                        else:
                                                count_3 = 1
                                                sub_frame_2.destroy
                                                if count_2 == 1:

        sub_frame_1=Frame(window,width=(w - w_frame - 30),height=h/2.,bg='white')

        sub_frame_1.place(relx=1,rely=0,anchor='ne')

                                                        f = Figure(figsize=((w - w_frame -
30)/100., h/200), dpi=100)
                                                        a = f.add_subplot(111)

                                                        a.plot([s_1_AC, s_2_AC, s_3_AC,
s_g_AC, s_4_AC, s_1_AC],[T_1_AC, T_1_AC, T_3_AC, T_4_AC, T_4_AC, T_1_AC],
label='Refrigerant', color='orange')
                                                        a.plot(s_array,T_array, color='k')

                                                        # a tk.DrawingArea
                                                        canvas = FigureCanvasTkAgg(f,
master=sub_frame_1)
                                                        canvas.show()

        canvas.get_tk_widget().pack(side=TOP, expand=0)

                                                        toolbar =
NavigationToolbar2TkAgg(canvas, sub_frame_1)
                                                        toolbar.update()
                                                        canvas._tkcanvas.pack(side=TOP,
expand=0)
                                                elif count_2 == 2:

        sub_frame_1=Frame(window,width=(w - w_frame - 30),height=h/2.,bg='white')

        sub_frame_1.place(relx=1,rely=0,anchor='ne')

                                                        f = Figure(figsize=((w - w_frame -
30)/100., h/200), dpi=100)
                                                        a = f.add_subplot(111)

                                                        a.plot([s_1_des, s_2_des, s_3_des,
s_g_des, s_4_des, s_1_des],[T_1_des, T_1_des, T_3_des, T_4_des, T_4_des, T_1_des],
label='Refrigerant', color='orange')
                                                        a.plot(s_array,T_array, color='k')
                                        # a tk.DrawingArea
                                        canvas = FigureCanvasTkAgg(f,
master=sub_frame_1)
                                        canvas.show()

    canvas.get_tk_widget().pack(side=TOP, expand=0)

                                        toolbar =
NavigationToolbar2TkAgg(canvas, sub_frame_1)
                                        toolbar.update()
                                        canvas._tkcanvas.pack(side=TOP,
expand=0)
                                    elif count_2 == 3:
                                        sub_frame_1.destroy()

    sub_frame_1=Frame(window,width=(w - w_frame - 30),height=h/2.,bg='white')

    sub_frame_1.place(relx=1,rely=0,anchor='ne')

                                        f = Figure(figsize=((w - w_frame -
30)/100., h/200), dpi=100)
                                        a = f.add_subplot(111)

                                        a.plot([s_1_NIPAAm,
s_2_NIPAAm, s_3_NIPAAm, s_g_NIPAAm, s_4_NIPAAm,
s_1_NIPAAm],[T_1_NIPAAm, T_1_NIPAAm, T_3_NIPAAm, T_4_NIPAAm,
T_4_NIPAAm, T_1_NIPAAm], label='Refrigerant', color='orange')
                                        a.plot(s_array,T_array, color='k')

                                        # a tk.DrawingArea
                                        canvas = FigureCanvasTkAgg(f,
master=sub_frame_1)
                                        canvas.show()

    canvas.get_tk_widget().pack(side=TOP, expand=0)

                                        toolbar =
NavigationToolbar2TkAgg(canvas, sub_frame_1)
                                        toolbar.update()
                                        canvas._tkcanvas.pack(side=TOP,
expand=0)
                                    else:
                                        1
				Button(window, text='T-s', command=Ts_diag, bg='white',
font=("Open Sans", 10)).place(x=(w_frame + 30), rely=0.5, anchor='nw') #T-s figure
button

				def Ph_diag():
					global main_figure
					global main_frame
					global main_canvas
					global canvas_image
					global vbar
					global hbar
					global count_3
					global count_2
					global sub_frame_1
					global sub_frame_2
					#global canvas_image
					if count_3 == 2:
						1
					else:
						count_3 = 2
						sub_frame_1.destroy
						if count_2 == 1:

	sub_frame_2=Frame(window,width=(w - w_frame - 30),height=h/2.,bg='white')

	sub_frame_2.place(relx=1,rely=0,anchor='ne')

							f_2 = Figure(figsize=((w - w_frame -
30)/100., h/200), dpi=100)
							a = f_2.add_subplot(111)

							a.plot(h_array,P_array, color='k')
							a.plot([h_1_AC, h_2_AC, h_3_AC,
h_4_AC, h_1_AC],[P_evap_AC, P_evap_AC, P_cond_AC, P_cond_AC, P_evap_AC],
color='orange')
							a.set_yscale("log")
							a.set_xticks([h_1_AC, (h_1_AC +
h_3_AC)/2., h_3_AC], minor=False)


							# a tk.DrawingArea
							canvas_2 =
FigureCanvasTkAgg(f_2, master=sub_frame_2)
							canvas_2.show()
	canvas_2.get_tk_widget().pack(side=TOP, expand=0)

							toolbar_2 = 
NavigationToolbar2TkAgg(canvas_2, sub_frame_2)
							toolbar_2.update()
							canvas_2._tkcanvas.pack(side=TOP, 
expand=0)
						elif count_2 == 2:

	sub_frame_2=Frame(window,width=(w - w_frame - 30),height=h/2.,bg='white')

	sub_frame_2.place(relx=1,rely=0,anchor='ne')

							f_2 = Figure(figsize=((w - w_frame - 
30)/100., h/200), dpi=100)
							a = f_2.add_subplot(111)

							a.plot(h_array,P_array, color='k')
							a.plot([h_1_des, h_2_des, h_3_des, 
h_4_des, h_1_des],[P_evap_des, P_evap_des, P_cond_des, P_cond_des, P_evap_des], 
color='orange')
							a.set_yscale("log")
							a.set_xticks([h_1_des, (h_1_des + 
h_3_des)/2., h_3_des], minor=False)


							# a tk.DrawingArea
							canvas_2 = 
FigureCanvasTkAgg(f_2, master=sub_frame_2)
							canvas_2.show()

	canvas_2.get_tk_widget().pack(side=TOP, expand=0)

							toolbar_2 = 
NavigationToolbar2TkAgg(canvas_2, sub_frame_2)
							toolbar_2.update()
							canvas_2._tkcanvas.pack(side=TOP, 
expand=0)
						elif count_2 == 3:

	sub_frame_2=Frame(window,width=(w - w_frame - 30),height=h/2.,bg='white')

	sub_frame_2.place(relx=1,rely=0,anchor='ne')
```

```
                                                f_2 = Figure(figsize=((w - w_frame -
30)/100., h/200), dpi=100)
                                                a = f_2.add_subplot(111)

                                                a.plot(h_array,P_array, color='k')
                                                a.plot([h_1_NIPAAm,
h_2_NIPAAm, h_3_NIPAAm, h_4_NIPAAm, h_1_NIPAAm],[P_evap_NIPAAm,
P_evap_NIPAAm, P_cond_NIPAAm, P_cond_NIPAAm, P_evap_NIPAAm],
color='orange')
                                                a.set_yscale("log")
                                                a.set_xticks([h_1_NIPAAm,
(h_1_NIPAAm + h_3_NIPAAm)/2., h_3_NIPAAm], minor=False)



                                                # a tk.DrawingArea
                                                canvas_2 =
FigureCanvasTkAgg(f_2, master=sub_frame_2)
                                                canvas_2.show()

     canvas_2.get_tk_widget().pack(side=TOP, expand=0)

                                                toolbar_2 =
NavigationToolbar2TkAgg(canvas_2, sub_frame_2)
                                                toolbar_2.update()
                                                canvas_2._tkcanvas.pack(side=TOP,
expand=0)
                                           else:
                                                1
                              Button(window, text='P-h', command=Ph_diag, bg='white',
font=("Open Sans", 10)).place(relx=1, rely=0.5, anchor='ne') #P-h figure button



                              sub_frame_1=Frame(window,width=(w - w_frame -
30),height=h/2.,bg='white')
                              sub_frame_1.place(relx=1,rely=0,anchor='ne')

                              f = Figure(figsize=((w - w_frame - 30)/100., h/200),
dpi=100)
                              a = f.add_subplot(111)

                              a.plot([s_1_AC, s_2_AC, s_3_AC, s_g_AC, s_4_AC,
s_1_AC],[T_1_AC, T_1_AC, T_3_AC, T_4_AC, T_4_AC, T_1_AC],
label='Refrigerant', color='orange')
```

```
                a.plot(s_array,T_array, color='k')

                # a tk.DrawingArea
                canvas = FigureCanvasTkAgg(f, master=sub_frame_1)
                canvas.show()
                canvas.get_tk_widget().pack(side=TOP, expand=0)

                toolbar = NavigationToolbar2TkAgg(canvas, sub_frame_1)
                toolbar.update()
                canvas._tkcanvas.pack(side=TOP, expand=0)




        elif v.get() == 2:
            window_5_title = Label(window, bg='white', text='Input Values',
font=("Open Sans Bold", 25))
            window_5_title.place(x=w/2., y=0, anchor="n")

            window_5_underline = Label(window, bg='white',
text='__________________________________________________________________________
______________________', font=("Open Sans Bold", 25))
            window_5_underline.place(x=w/2., y=50, anchor="n")
            text_1 = Label(window, bg='white', text='Thermostat Set
Temperature', font=("Open Sans", 15))
            text_1.place(relx=0.5, rely=0.38, anchor="e")
            text_2 = Label(window, bg='white', text='Initial Indoor Air
Humidity Ratio', font=("Open Sans", 15))
            text_2.place(relx=0.5, rely=0.42, anchor="e")
            text_3 = Label(window, bg='white', text='Outside Air
Temperature', font=("Open Sans", 15))
            text_3.place(relx=0.5, rely=0.54, anchor="e")
            text_4 = Label(window, bg='white', text='Outside Air Humidity
Ratio', font=("Open Sans", 15))
            text_4.place(relx=0.5, rely=0.58, anchor="e")
            text_5 = Label(window, bg='white', text='AC Outlet Air
Temperature', font=("Open Sans", 15))
            text_5.place(relx=0.5, rely=0.46, anchor="e")
            text_6 = Label(window, bg='white', text='AC Outlet Air Humidity
Ratio', font=("Open Sans", 15))
            text_6.place(relx=0.5, rely=0.50, anchor="e")
            text_7 = Label(window, bg='white', text='Percent Supply Air from
Outside', font=("Open Sans", 15))
```

```
text_7.place(relx=0.5, rely=0.62, anchor="e")
txt_1 = Entry(window,width=20)
txt_1.insert(INSERT,"294.59444")
txt_1.place(relx=0.5, rely=0.38, anchor=W)
txt_2 = Entry(window,width=20)
txt_2.insert(INSERT,"0.008164972225996216")
txt_2.place(relx=0.5, rely=0.42, anchor=W)
txt_3 = Entry(window,width=20)
txt_3.insert(INSERT,"298.15")
txt_3.place(relx=0.5, rely=0.54, anchor=W)
txt_4 = Entry(window,width=20)
txt_4.insert(INSERT,"0.02009")
txt_4.place(relx=0.5, rely=0.58, anchor=W)
txt_5 = Entry(window,width=20)
txt_5.insert(INSERT,"284.15")
txt_5.place(relx=0.5, rely=0.46, anchor=W)
txt_6 = Entry(window,width=20)
txt_6.insert(INSERT,"0.008164972225996216")
txt_6.place(relx=0.5, rely=0.50, anchor=W)
txt_7 = Entry(window,width=20)
txt_7.insert(INSERT,"50")
txt_7.place(relx=0.5, rely=0.62, anchor=W)

def fifth_window():
	#Evap Cooling
	global count
	global count_2
	global canvas_image
	global main_figure
	global main_frame
	global main_canvas
	global w_frame
	global h_frame
	global vbar
	global hbar
	global val_save
	global T_set
	global x_i
	global T_outside
	global x_outside
	global T_air_o
	global x_air_o
	global Sb
	global mass_H2O
```

```
                                global heat_regen
                                global Q_regen_NIPAAm
                                global h_fg_NIPAAm
                                global delta_m_h2o
                                global des_amount
                                global COP
                                global sub_frame_1
                                global sub_frame_2
                                global t_f
                                global delta_t
                                global T_HX_preheat_o
                                global C_p_regen
                                global Sb_2
                                global line
                                global x_HX_preheat_o
                                global a
                                global canvas
                                global Q_useful
                                global mass_H2O_reclaimed
                                global delta_x_dehum
                                global T_dehum_slope
                                global T_regen_NIPAAm
                                global c_p_NIPAAm_dry
                                global c_p_NIPAAm_wet
                                global m_des
                                global m_NIPAAm
                                global percent_vent
                                [delta_t, t_f, T_h_array_des, x_h_array_des,
T_duct_array_des, x_duct_array_des, T_return_array_des,
x_return_array_des,t_f_NIPAAm, T_h_array_NIPAAm, x_h_array_NIPAAm,
T_duct_array_NIPAAm, x_duct_array_NIPAAm, T_return_array_NIPAAm,
x_return_array_NIPAAm, delta_m_h2o, h2o_des, h2o_NIPAAm, c_p_NIPAAm_dry,
c_p_NIPAAm_wet,
h_fg_NIPAAm,T_HX_preheat_i,T_HX_preheat_o,x_HX_preheat_o,C_p_regen,T_air_A
H,T_air_to_HX,T_air_to_AC,T_air_o,x_AH,x_dehum,x_dehum,x_air_o,Q_useful,m_h2
o_used,omega,m_NIPAAm,m_des,delta_C_NIPAAm] =
house_air_evap_cool(T_set,x_i,T_outside,x_outside,T_air_o,x_air_o,percent_vent)
                                T_regen_NIPAAm = 32 + 273.15
                                delta_x_dehum = (x_AH - x_dehum)
                                T_dehum_slope = (T_air_to_HX -
T_air_AH)/delta_x_dehum
                                final_index_3 = len(T_h_array_des)
                                final_index_5 = len(T_h_array_NIPAAm)
                                main_figure_3 = []
                                main_figure_4 = []
                                main_figure_5 = []
                                main_figure_6 = []
                                img_4 =
Image.open('output\psychrom\psychrom_desiccant_out.png')
                                img_6 =
Image.open('output\psychrom\psychrom_NIPAAm_out.png')
                                for ind in range(final_index_3):
                                        img_3 = Image.open('diag3.png')
                                        draw = ImageDraw.Draw(img_3)
                                        s = " "
                                        seq = ("T = ", str(round(T_h_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                                        T_i_str = s.join(seq)
                                        seq = ("T = ", str(round(T_return_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                                        T_r_str = s.join(seq)
                                        seq = ("T = ", str(round(T_duct_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                                        T_f_str = s.join(seq)
                                        seq = ("x = ",
str(round(x_return_array_des[ind]*1000, 2)), "g/kg")
                                        x_r_str = s.join(seq)
                                        seq = ("x = ", str(round(x_h_array_des[ind]*1000,
2)), "g/kg")
                                        x_i_str = s.join(seq)
                                        seq = ("x = ",
str(round(x_duct_array_des[ind]*1000, 2)), "g/kg")
                                        x_f_str = s.join(seq)
                                        seq = ("T = ", str(round(T_outside - 273.15, 2)),
u'\xb0'"C")
                                        T_out_str = s.join(seq)
                                        seq = ("x = ", str(round(x_outside*1000, 2)),
"g/kg")
                                        x_out_str = s.join(seq)
                                        draw.text((1015,318), T_i_str, font = fnt, fill =
(0,0,0))
                                        draw.text((960,513), T_r_str, font = fnt, fill =
(0,0,0))
                                        draw.text((960,132), T_f_str, font = fnt, fill =
(0,0,0))
                                        draw.text((1015,348), x_i_str, font = fnt, fill =
(0,0,0))
```

```
                                        draw.text((960,543), x_r_str, font = fnt, fill =
(0,0,0))
                                        draw.text((960,162), x_f_str, font = fnt, fill =
(0,0,0))
                                        draw.text((5,21), T_out_str, font = fnt, fill = (0,0,0))
                                        draw.text((5,43), x_out_str, font = fnt, fill = (0,0,0))
                                        draw.text((465,463), T_out_str, font = fnt, fill =
(0,0,0))
                                        draw.text((465,493), x_out_str, font = fnt, fill =
(0,0,0))
                                        main_figure_3.append(img_3)
                                        main_figure_4.append(img_4)
                                for ind in range(final_index_5):
                                        img_5 = Image.open('diag4.png')
                                        draw = ImageDraw.Draw(img_5)
                                        s = " "
                                        seq = ("T = ", str(round(T_h_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                                        T_i_str = s.join(seq)
                                        seq = ("T = ", str(round(T_return_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                                        T_r_str = s.join(seq)
                                        seq = ("T = ", str(round(T_duct_array_des[ind] -
273.15, 2)), u'\xb0'"C")
                                        T_f_str = s.join(seq)
                                        seq = ("x = ",
str(round(x_return_array_des[ind]*1000, 2)), "g/kg")
                                        x_r_str = s.join(seq)
                                        seq = ("x = ", str(round(x_h_array_des[ind]*1000,
2)), "g/kg")
                                        x_i_str = s.join(seq)
                                        seq = ("x = ",
str(round(x_duct_array_des[ind]*1000, 2)), "g/kg")
                                        x_f_str = s.join(seq)
                                        seq = ("T = ", str(round(T_outside - 273.15, 2)),
u'\xb0'"C")
                                        T_out_str = s.join(seq)
                                        seq = ("x = ", str(round(x_outside*1000, 2)),
"g/kg")
                                        x_out_str = s.join(seq)
                                        draw.text((1015,318), T_i_str, font = fnt, fill =
(0,0,0))
                                        draw.text((960,513), T_r_str, font = fnt, fill =
(0,0,0))
```

```
                                        draw.text((960,132), T_f_str, font = fnt, fill =
(0,0,0))
                                        draw.text((1015,348), x_i_str, font = fnt, fill =
(0,0,0))
                                        draw.text((960,543), x_r_str, font = fnt, fill =
(0,0,0))
                                        draw.text((960,162), x_f_str, font = fnt, fill =
(0,0,0))
                                        draw.text((5,21), T_out_str, font = fnt, fill = (0,0,0))
                                        draw.text((5,43), x_out_str, font = fnt, fill = (0,0,0))
                                        draw.text((465,463), T_out_str, font = fnt, fill =
(0,0,0))
                                        draw.text((465,493), x_out_str, font = fnt, fill =
(0,0,0))
                                        main_figure_5.append(img_5)
                                        main_figure_6.append(img_6)
                                count = 0
                                count_2 = 2
                                main_figure = main_figure_3
                                main = main_figure[0]
                                w_main, h_main = main.size
                                main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                w_main, h_main = main.size
                                w_frame = w_main
                                h_frame = h_main
                                s = " "
                                mass_H2O = Label(window, bg='white',
text=s.join(("Absorbed water: ", str(round(delta_m_h2o*1000, 2)), "g")), font=("Open
Sans", 15))
                                mass_H2O.place(x=5, y=(h_frame + 61), anchor="nw")
                                mass_H2O_used = Label(window, bg='white',
text=s.join(("Water consumed: ", str(round(m_h2o_used*1000, 2)), "g")), font=("Open
Sans", 15))
                                mass_H2O_used.place(x=(w_frame-60)/2., y=(h_frame +
61), anchor="n")
                                mass_H2O_reclaimed = Label(window, bg='white',
text=s.join(("Water reclaimed: ", str(round(0*1000, 2)), "g")), font=("Open Sans", 15))
                                mass_H2O_reclaimed.place(x=(w_frame + 100)/2.,
y=(h_frame + 131), anchor="n")
                                des_amount = Label(window, bg='white',
text=s.join(("Required desiccant: ", str(round(m_des*1000., 2)), "g")), font=("Open
Sans", 15))
                                des_amount.place(x=5, y=(h_frame + 131), anchor="nw")
                                heat_regen = Label(window, bg='white', text="",
font=("Open Sans", 15))
                                COP = Label(window, bg='white', text="", font=("Open
Sans", 15))
                                main = ImageTk.PhotoImage(main)
                                main.image = main

        main_frame=Frame(window,width=w_main,height=h_main)
                                main_frame.grid(row=0,column=0)

        main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

                                def des_button():
                                        global count
                                        global count_2
                                        global main_figure
                                        global canvas_image
                                        global main_frame
                                        global main_canvas
                                        global w_frame
                                        global h_frame
                                        global vbar
                                        global hbar
                                        global mass_H2O
                                        global heat_regen
                                        global des_amount
                                        global COP
                                        global t_f
                                        global delta_t
                                        global mass_H2O_reclaimed
                                        global percent_evap
                                        global T_start
                                        global T_stop
                                        global GUI_txt_1
                                        global GUI_txt_2
                                        global GUI_txt_3
                                        global m_des
                                        #global canvas_image
                                        if count_2 == 2:
                                                1
                                        else:
                                                global Sb_2
                                                global Sb_3
                                                global Sb_4
                                                if count_2 == 3:
                                                        Sb_2.destroy()
                                                        Sb_3.destroy()
                                                        Sb_4.destroy()
                                                        GUI_txt_1.destroy()
                                                        GUI_txt_2.destroy()
                                                        GUI_txt_3.destroy()
                                                else:
                                                        1
                                                global val_save
                                                global Sb
                                                global sub_frame_1
                                                global sub_frame_2
                                                global a
                                                global canvas
                                                global line
                                                line, = a.plot([T_HX_preheat_o - 
273.15,T_HX_preheat_o - 273.15],[x_HX_preheat_o,x_HX_preheat_o], color='pink')
                                                canvas.draw()
                                                count = 0
                                                count_2 = 2
                                                main_canvas.destroy()
                                                vbar.destroy()
                                                hbar.destroy()
                                                main_figure = main_figure_3
                                                main = main_figure[val_save]
                                                w_main, h_main = main.size
                                                main = main.resize([int(w - h/2. - 145), 
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                                w_main, h_main = main.size
                                                main = ImageTk.PhotoImage(main)
                                                main.image = main

        main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

        hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                                hbar.pack(side=BOTTOM,fill=X)
                                                hbar.config(command=main_canvas.xview)

        vbar=Scrollbar(main_frame,orient=VERTICAL)
                                                vbar.pack(side=RIGHT,fill=Y)
                                                vbar.config(command=main_canvas.yview)
```

```
        main_canvas.config(width=w_main,height=h_main)

        main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

        main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                        mass_H2O.destroy()
                                        heat_regen.destroy()
                                        des_amount.destroy()
                                        COP.destroy()
                                        mass_H2O_reclaimed.destroy()
                                        s = " "
                                        mass_H2O = Label(window, bg='white',
text=s.join(("Absorbed water: ", str(round(delta_m_h2o*1000, 2)), "g")), font=("Open
Sans", 15))
                                        mass_H2O.place(x=5, y=(h_frame + 61),
anchor="nw")
                                        mass_H2O_reclaimed = Label(window,
bg='white', text=s.join(("Water reclaimed: ", str(round(0*1000, 2)), "g")), font=("Open
Sans", 15))
                                        mass_H2O_reclaimed.place(x=(w_frame +
100)/2., y=(h_frame + 131), anchor="n")
                                        des_amount = Label(window, bg='white',
text=s.join(("Required desiccant: ", str(round(m_des*1000., 2)), "g")), font=("Open
Sans", 15))
                                        des_amount.place(x=5, y=(h_frame + 131),
anchor="nw")
                                        Sb.destroy()
                                        Sb =
Scale(window,orient=HORIZONTAL,bg='white',from_=0,to=t_f,command=slider,length
=w_frame,resolution=delta_t)
                                        Sb.set(val_save*delta_t)
                                        Sb.place(relx=0,rely=1,anchor='sw')
                                        Sb_2 =
Scale(window,orient=VERTICAL,bg='white',from_=140,to=(T_HX_preheat_o -
273.15),command=slider_des,length=(h/2. - 70),resolution=1)
                                        Sb_2.set(T_HX_preheat_o - 273.15)
                                        Sb_2.place(x=(w_frame + 30 + w)/2.,y=(h -
20),anchor='s')
```

```
                                        Button(window, text='Desiccant + Evaporative Cooling',
command=des_button, bg='white', font=("Open Sans", 10)).place(x=0, y=(h - 25),
anchor='sw') #Desiccant button

                                        def NIPAAm_button():
                                                global count
                                                global count_2
                                                global main_figure
                                                global canvas_image
                                                global main_frame
                                                global main_canvas
                                                global w_frame
                                                global h_frame
                                                global vbar
                                                global hbar
                                                global mass_H2O
                                                global heat_regen
                                                global Q_regen_NIPAAm
                                                global des_amount
                                                global COP
                                                global t_f
                                                global delta_t
                                                global a
                                                global canvas
                                                global percent_evap
                                                global T_start
                                                global T_stop
                                                global GUI_txt_1
                                                global GUI_txt_2
                                                global GUI_txt_3
                                                global Sb_3
                                                global Sb_4
                                                global m_NIPAAm
                                                #global canvas_image
                                                if count_2 == 3:
                                                        1
                                                else:
                                                        global val_save
                                                        global Sb
                                                        global Sb_2
                                                        global sub_frame_1
                                                        global sub_frame_2
                                                        if count_2 == 2:
                                                                Sb_2.destroy()
                                                        else:
                                                                1
                                                        count = 0
                                                        count_2 = 3
                                                        line.remove()
                                                        canvas.draw()
                                                        main_canvas.destroy()
                                                        vbar.destroy()
                                                        hbar.destroy()
                                                        main_figure = main_figure_5
                                                        main = main_figure[val_save]
                                                        w_main, h_main = main.size
                                                        main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                                        w_main, h_main = main.size
                                                        main = ImageTk.PhotoImage(main)
                                                        main.image = main

                        main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

                        hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                                        hbar.pack(side=BOTTOM,fill=X)
                                                        hbar.config(command=main_canvas.xview)

                        vbar=Scrollbar(main_frame,orient=VERTICAL)
                                                        vbar.pack(side=RIGHT,fill=Y)
                                                        vbar.config(command=main_canvas.yview)

                        main_canvas.config(width=w_main,height=h_main)

                        main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

                        main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                                        mass_H2O.destroy()
                                                        heat_regen.destroy()
                                                        des_amount.destroy()
                                                        COP.destroy()
                                                        s = " "
                                                        mass_H2O = Label(window, bg='white',
text=s.join(("Absorbed water: ", str(round(delta_m_h2o*1000, 2)), "g")), font=("Open
Sans", 15))
```

```
                                        mass_H2O.place(x=5, y=(h_frame + 61),
anchor="nw")
                                        des_amount = Label(window, bg='white',
text=s.join(("Required NIPAAm: ", str(round(m_NIPAAm*1000., 2)), "g")),
font=("Open Sans", 15))
                                        des_amount.place(x=5, y=(h_frame + 131),
anchor="nw")
                                        Sb.destroy()
                                        Sb =
Scale(window,orient=HORIZONTAL,bg='white',from_=0,to=t_f,command=slider,length
=w_frame,resolution=delta_t)
                                        Sb.set(val_save*delta_t)
                                        Sb.place(relx=0,rely=1,anchor='sw')
                                        percent_evap = 0
                                        T_start = 20 + 273.15
                                        T_stop = 20 + 273.15
                                        Sb_2 =
Scale(window,orient=VERTICAL,bg='white',from_=100,to=0,command=slider_NIPAA
m,length=(h/2. - 70),resolution=1)
                                        Sb_2.set(0)
                                        Sb_2.place(x=(w_frame + 30 + w)/2.,y=(h -
20),anchor='s')
                                        Sb_3 =
Scale(window,orient=VERTICAL,bg='white',from_=(T_regen_NIPAAm -
273.15),to=0,command=slider_NIPAAm_2,length=(h/2. - 70),resolution=1)
                                        Sb_3.set(0)
                                        Sb_3.place(relx=1,y=(h - 20),anchor='se')
                                        Sb_4 =
Scale(window,orient=VERTICAL,bg='white',from_=(T_regen_NIPAAm -
273.15),to=0,command=slider_NIPAAm_3,length=(h/2. - 70),resolution=1)
                                        Sb_4.set(0)
                                        Sb_4.place(x=(w_frame + 30),y=(h -
20),anchor='sw')
                                        GUI_txt_1 = Label(window, bg='white',
text="Start", font=("Open Sans", 12))
                                        GUI_txt_1.place(x=(w_frame +
30),rely=1,anchor='sw')
                                        GUI_txt_2 = Label(window, bg='white',
text="Percent Evaporated", font=("Open Sans", 12))
                                        GUI_txt_2.place(x=(w_frame + 30 +
w)/2.,rely=1,anchor='s')
                                        GUI_txt_3 = Label(window, bg='white',
text="Stop", font=("Open Sans", 12))
                                        GUI_txt_3.place(relx=1,rely=1,anchor='se')
```

```
                                Button(window, text='NIPAAm + Evaporative Cooling',
command=NIPAAm_button, bg='white', font=("Open Sans", 10)).place(x=w_main, y=(h
- 25), anchor='se') #NIPAAm button


                                hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                hbar.pack(side=BOTTOM,fill=X)
                                hbar.config(command=main_canvas.xview)
                                vbar=Scrollbar(main_frame,orient=VERTICAL)
                                vbar.pack(side=RIGHT,fill=Y)
                                vbar.config(command=main_canvas.yview)
                                main_canvas.config(width=w_main,height=h_main)
                                main_canvas.config(xscrollcommand=hbar.set,
yscrollcommand=vbar.set)
                                main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                val_save = 0
                                def slider(val):
                                        global val_save
                                        global count
                                        global delta_t
                                        main_canvas.delete("all")
                                        main = main_figure[int(float(val)/delta_t)]
                                        w_main, h_main = main.size
                                        if count == 0:
                                                main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                        else:
                                                1
                                        w_main, h_main = main.size
                                        main = ImageTk.PhotoImage(main)
                                        main.image = main
                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                        val_save = int(float(val)/delta_t)
                                Sb =
Scale(window,orient=HORIZONTAL,bg='white',from_=0,to=t_f,command=slider,length
=w_frame,resolution=delta_t)
                                Sb.place(relx=0,rely=1,anchor='sw')
                                def slider_NIPAAm(val):
                                        global m_NIPAAm
                                        global delta_m_h2o
                                        global h_fg_NIPAAm
                                        global heat_regen
                                        global COP
                                        global mass_H2O_reclaimed
                                        global percent_evap
                                        global T_start
                                        global T_stop
                                        global c_p_NIPAAm_dry
                                        global c_p_NIPAAm_wet
                                        global T_regen_NIPAAm
                                        global COP_AC_NIPAAm
                                        global t_f
                                        heat_regen.destroy()
                                        COP.destroy()
                                        mass_H2O_reclaimed.destroy()
                                        percent_evap = float(val)
                                        regeneration_energy =
m_NIPAAm*omega/360.*((T_regen_NIPAAm - T_stop)*c_p_NIPAAm_wet +
delta_C_NIPAAm*percent_evap*h_fg_NIPAAm/100.)*t_f
                                        heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((regeneration_energy)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                                        heat_regen.place(x=(w_frame - 30), y=(h_frame +
61), anchor="ne")
                                        mass_H2O_reclaimed = Label(window, bg='white',
text=s.join(("Water reclaimed: ", str(round(delta_m_h2o*(1 - float(val)/100.)*1000, 2)),
"g")), font=("Open Sans", 15))
                                        mass_H2O_reclaimed.place(x=(w_frame + 100)/2.,
y=(h_frame + 131), anchor="n")
                                        if regeneration_energy == 0:
                                                COP = Label(window, bg='white',
text="COP: inf", font=("Open Sans", 15))
                                        else:
                                                COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_useful/( regeneration_energy)), 2)))), font=("Open
Sans", 15))
                                        COP.place(x=(w_frame - 30), y=(h_frame + 131),
anchor="ne")
                                def slider_NIPAAm_2(val):
                                        global m_NIPAAm
                                        global h_fg_NIPAAm
                                        global heat_regen
                                        global COP
                                        global mass_H2O_reclaimed
                                        global percent_evap
                                        global T_start
                                        global T_stop
                                        global c_p_NIPAAm_dry
                                        global c_p_NIPAAm_wet
                                        global T_regen_NIPAAm
                                        global COP_AC_NIPAAm
                                        heat_regen.destroy()
                                        COP.destroy()
                                        T_stop = float(val) + 273.15
                                        regeneration_energy =
m_NIPAAm*omega/360.*((T_regen_NIPAAm - T_stop)*c_p_NIPAAm_wet +
delta_C_NIPAAm*percent_evap*h_fg_NIPAAm/100.)*t_f
                                        heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((regeneration_energy)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                                        heat_regen.place(x=(w_frame - 30), y=(h_frame +
61), anchor="ne")
                                        if regeneration_energy == 0:
                                                COP = Label(window, bg='white',
text="COP: inf", font=("Open Sans", 15))
                                        else:
                                                COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_useful/( regeneration_energy)), 2)))), font=("Open
Sans", 15))
                                        COP.place(x=(w_frame - 30), y=(h_frame + 131),
anchor="ne")
                                def slider_NIPAAm_3(val):
                                        global m_NIPAAm
                                        global h_fg_NIPAAm
                                        global heat_regen
                                        global COP
                                        global mass_H2O_reclaimed
                                        global percent_evap
                                        global T_start
                                        global T_stop
                                        global c_p_NIPAAm_dry
                                        global c_p_NIPAAm_wet
                                        global T_regen_NIPAAm
                                        global COP_AC_NIPAAm
                                        heat_regen.destroy()
                                        COP.destroy()
                                        T_start = float(val) + 273.15
                                        regeneration_energy =
m_NIPAAm*omega/360.*((T_regen_NIPAAm - T_stop)*c_p_NIPAAm_wet +
delta_C_NIPAAm*percent_evap*h_fg_NIPAAm/100.)*t_f
                                        heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((regeneration_energy)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                                        heat_regen.place(x=(w_frame - 30), y=(h_frame +
61), anchor="ne")
                                        if regeneration_energy == 0:
                                                COP = Label(window, bg='white',
text="COP: inf", font=("Open Sans", 15))
                                        else:
                                                COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_useful/( regeneration_energy)), 2)))), font=("Open
Sans", 15))
                                        COP.place(x=(w_frame - 30), y=(h_frame + 131),
anchor="ne")
                                def slider_des(val):
                                        global T_HX_preheat_o
                                        global C_p_regen
                                        global heat_regen
                                        global COP
                                        global t_f
                                        global line
                                        global x_HX_preheat_o
                                        global a
                                        global canvas
                                        global delta_x_dehum
                                        global T_dehum_slope
                                        line.remove()
                                        heat_regen.destroy()
                                        COP.destroy()
                                        if float(val) == float(round((T_HX_preheat_o -
273.15),0)):
                                                heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((C_p_regen*(0)*t_f)/1000., 2)), "kJ")),
font=("Open Sans", 15))
                                                heat_regen.place(x=(w_frame - 30),
y=(h_frame + 61), anchor="ne")
                                                COP = Label(window, bg='white',
text="COP: inf", font=("Open Sans", 15))
                                                COP.place(x=(w_frame - 30), y=(h_frame +
131), anchor="ne")
                                        else:
```

```
                                        heat_regen = Label(window, bg='white',
text=s.join(("Regeneration energy: ", str(round((C_p_regen*(float(val) + 273.15 -
T_HX_preheat_o)*t_f)/1000., 2)), "kJ")), font=("Open Sans", 15))
                                        heat_regen.place(x=(w_frame - 30),
y=(h_frame + 61), anchor="ne")
                                        COP = Label(window, bg='white',
text=s.join(("COP: ", str(round((Q_useful/(C_p_regen*(float(val) + 273.15 -
T_HX_preheat_o)*t_f)), 2)))), font=("Open Sans", 15))
                                        COP.place(x=(w_frame - 30), y=(h_frame +
131), anchor="ne")
                                line, = a.plot([T_HX_preheat_o -
273.15,float(val),(float(val) -
T_dehum_slope*delta_x_dehum)],[x_HX_preheat_o,x_HX_preheat_o,(x_HX_preheat_o
+ delta_x_dehum)], color='pink')
                                canvas.draw()
                        def next_fig():
                                global count
                                global count_2
                                global main_figure
                                global canvas_image
                                global main_frame
                                global main_canvas
                                global w_frame
                                global h_frame
                                global vbar
                                global hbar
                                #global canvas_image
                                if count == 1:
                                        1
                                else:
                                        global val_save
                                        count = count + 1
                                        main_canvas.destroy()
                                        vbar.destroy()
                                        hbar.destroy()
                                        if count_2 == 1:
                                                main_figure = main_figure_2
                                        elif count_2 == 2:
                                                main_figure = main_figure_4
                                        elif count_2 == 3:
                                                main_figure = main_figure_6
                                        else:
                                                1
                                        main = main_figure[val_save]
                                        w_main, h_main = main.size
                                        #main = main.resize([int(w - h/2. - 25),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                        main = ImageTk.PhotoImage(main)
                                        main.image = main

        main_canvas=Canvas(main_frame,bg='white',width=w_frame,height=h_frame,scr
ollregion=(0,0,w_main,h_main))

        hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                        hbar.pack(side=BOTTOM,fill=X)
                                        hbar.config(command=main_canvas.xview)

        vbar=Scrollbar(main_frame,orient=VERTICAL)
                                        vbar.pack(side=RIGHT,fill=Y)
                                        vbar.config(command=main_canvas.yview)

        main_canvas.config(width=w_frame,height=h_frame)

        main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

        main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                        canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                Button(window, text='Next Figure', command=next_fig,
bg='white', font=("Open Sans", 10)).place(x=w_main, y=(h_main + 25), anchor='ne')
#next figure button

                                def prev_fig():
                                        global count
                                        global count_2
                                        global main_figure
                                        global main_frame
                                        global main_canvas
                                        global canvas_image
                                        global vbar
                                        global hbar
                                        #global canvas_image
                                        if count == 0:
                                                1
                                        else:
                                                global val_save
                                                count = count - 1
                                                main_canvas.destroy()
```

```
                                                  vbar.destroy()
                                                  hbar.destroy()
                                                  if count_2 == 1:
                                                          main_figure = main_figure_1
                                                  elif count_2 == 2:
                                                          main_figure = main_figure_3
                                                  elif count_2 == 3:
                                                          main_figure = main_figure_5
                                                  else:
                                                          1
                                                  main = main_figure[val_save]
                                                  w_main, h_main = main.size
                                                  main = main.resize([int(w - h/2. - 145),
int((float(h_main)/w_main)*w*2./3.)], Image.ANTIALIAS)
                                                  w_main, h_main = main.size
                                                  main = ImageTk.PhotoImage(main)
                                                  main.image = main

        main_canvas=Canvas(main_frame,bg='white',width=w_main,height=h_main,scro
llregion=(0,0,w_main,h_main))

        hbar=Scrollbar(main_frame,orient=HORIZONTAL)
                                                  hbar.pack(side=BOTTOM,fill=X)
                                                  hbar.config(command=main_canvas.xview)

        vbar=Scrollbar(main_frame,orient=VERTICAL)
                                                  vbar.pack(side=RIGHT,fill=Y)
                                                  vbar.config(command=main_canvas.yview)

        main_canvas.config(width=w_main,height=h_main)

        main_canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)

        main_canvas.pack(side=LEFT,expand=True,fill=BOTH)
                                                  canvas_image =
main_canvas.create_image(0,0,image=main, anchor="nw")
                                  Button(window, text='Previous Figure',
command=prev_fig, bg='white', font=("Open Sans", 10)).place(x=0, y=(h_main + 25),
anchor='nw') #prev figure button
```

```
                                        sub_frame_1=Frame(window,width=(w - w_frame -
30),height=h/2.,bg='white')
                                        sub_frame_1.place(relx=1,rely=0,anchor='ne')

                                        f = Figure(figsize=((w - w_frame - 30)/100., h/200.),
dpi=100, tight_layout=True)
                                        a = f.add_subplot(111)
                                        a.set_xlabel('Temperature (Celsius)')
                                        a.set_ylabel('Humidity Ratio (kg/kg)')
                                        a.plot([T_air_AH - 273.15,T_air_to_HX -
273.15,T_air_to_AC - 273.15,T_air_o - 273.15],[x_AH,x_dehum,x_dehum,x_air_o],
label='Supply Air', color='green')
                                        a.plot([T_HX_preheat_i - 273.15,T_HX_preheat_o -
273.15],[x_HX_preheat_o,x_HX_preheat_o], label='Process Air', color='pink')
                                        T_s_curve = []
                                        for x_val in
np.linspace(x_s(273.15,101325),x_HX_preheat_o*2):
                                                T_s_curve.append(T_s(x_val,101325) - 273.15)

        a.plot(T_s_curve,np.linspace(x_s(273.15,101325),x_HX_preheat_o*2),
label='Saturation Curve', color='black')
                                        a.legend()
                                        a.set_xlim(0,150)
                                        a.set_ylim(0,x_HX_preheat_o*2)



                                        line, = a.plot([T_HX_preheat_o - 273.15,T_HX_preheat_o
- 273.15],[x_HX_preheat_o,x_HX_preheat_o], color='pink')

                                        # a tk.DrawingArea
                                        canvas = FigureCanvasTkAgg(f, master=sub_frame_1)
                                        canvas.show()
                                        canvas.get_tk_widget().pack(side=TOP,anchor='nw')

                                        toolbar = NavigationToolbar2TkAgg(canvas, sub_frame_1)
                                        toolbar.update()
                                        canvas._tkcanvas.pack(side=TOP, expand=0)

                                        Sb_2 =
Scale(window,orient=VERTICAL,bg='white',from_=140,to=(T_HX_preheat_o -
273.15),command=slider_des,length=(h/2. - 70),resolution=1)
                                        Sb_2.set(T_HX_preheat_o - 273.15)
                                        Sb_2.place(x=(w_frame + 30 + w)/2.,y=(h - 20),anchor='s')
```

```
		else:
			1
		#Define function to erase window
		def cont_func():

			if v.get() == 1:
				1
			else:
				global T_set
				global x_i
				global T_outside
				global x_outside
				global T_air_o
				global selection
				global percent_vent
				T_set = float(txt_1.get())
				x_i = float(txt_2.get())
				T_outside = float(txt_3.get())
				x_outside = float(txt_4.get())
				T_air_o = float(txt_5.get())
				percent_vent = float(txt_7.get())
				if selection == "Evap":
					global x_air_o
					x_air_o = float(txt_6.get())
				else:
					1
				def all_children (window) :
					_list = window.winfo_children()

					for item in _list :
						if item.winfo_children() :
							_list.extend(item.winfo_children())

					return _list

				widget_list = all_children(window)
				for item in widget_list:
					item.destroy()
				fifth_window()
				v.set(0)
		Button(window, text='Next', command=cont_func, bg='white',
font=("Open Sans", 20)).place(relx=0.5, rely=0.8, anchor=CENTER) #fourth window
continue button
	#Define function to erase window
```

```
    def cont_func():

        if v.get() == 0:
            1
        else:
            def all_children (window) :
                _list = window.winfo_children()

                for item in _list :
                    if item.winfo_children() :
                        _list.extend(item.winfo_children())

                return _list

            widget_list = all_children(window)
            for item in widget_list:
                item.destroy()
            if v.get() == 2:
                global selection
                selection = "Evap"
            else:
                selection = ""
            fourth_window()
            v.set(0)
    Button(window, text='Next', command=cont_func, bg='white', font=("Open
Sans", 20)).place(relx=0.5, rely=0.8, anchor=CENTER) #third window continue button



#Define function to erase window
def cont_func():
    def all_children (window) :
        _list = window.winfo_children()

        for item in _list :
            if item.winfo_children() :
                _list.extend(item.winfo_children())

        return _list

    widget_list = all_children(window)
    for item in widget_list:
        item.destroy()
    third_window()
```

```
        v.set(0)
Button(window, text='Next', command=cont_func, bg='white', font=("Open Sans",
20)).place(relx=0.5, rely=0.84, anchor=CENTER) #first window continue button


window.mainloop()
```

APPENDIX C

PYTHON CODE FOR “HOUSE_AIR.PY”

```
def
house_air(T_set,x_i,T_outside,x_outside,T_air_calibrate,x_air_calibrate,T_outside_calibr
ate,x_outside_calibrate,T_air_calibrate_o,percent_vent):
	import math
	from CoolProp import CoolProp as CP
	from scipy.optimize import fsolve
	from AC import AC
	from Dehum import Dehum
	from Psyplot import Psyplot
	from HX import HX
	from scipy.interpolate import interp1d
	from RH import RH
	from T_s import T_s

	T_air_i = T_set + 5./9. #initial temperature of air within the house [K]
	x_to_AC = x_i
	V_tot = 271.84 #total conditioned space volume [m^3]
	P_air = 101325 #total pressure within conditioned space [Pa]
	indoor_evap_rate = 0. #no indoor evaporation
	m_dot_supply = 0.7 #defines the total mass flow rate of supply air [kg/s]
	m_dot_vent = percent_vent*m_dot_supply/100. #portion of supply air that comes
from outside [kg/s]
	h_duct_i = 8.33 #duct interior heat transfer coefficient [W/m^2K]
	h_duct_o = 0. #duct exterior heat transfer coefficient [W/m^2K]
	L_duct = 9.14 #length of duct [m]
	D_duct = 0.1016 #diameter of duct [m]
	A_duct = math.pi*D_duct*L_duct #surface area of duct [m^2]
	C_duct = 470*6.404*L_duct #heat capacity of duct [J/K]

	T_regen_NIPAAm = 32 + 273.15 #Regen temperature of NIPAAm, used only to
find the specific heat of water within the NIPAAm; this value is redefined in GUI.py
	c_p_des = 960. #desiccant specific heat
	C_des = 0.4 #absorption capacity of desiccant in kg_water/kg_des
	c_p_NIPAAm = 960. #NIPAAm specific heat
	percent_solid_vol = 0.005 #percent of the room volume that is solid
	V_air = V_tot*(1 - percent_solid_vol) #volume of air within the room
	V_solid = V_tot*percent_solid_vol #volume of solid within the room
	c_p_solid = 903600 #volumetric heat capacity of the solid, J/(m^3*K)

	##The following section models the vapor compression only scenario
	cooling_mode = "VC"
	UAs_house = 0 #sets the heat transfer coefficient for house heat gain to zero
	T_air_initial = T_air_i
	x_initial = x_i
```

```
        sys_config = "AC Only"
        print_query = "no"
        M_a = 0.028964 #molecular mass of air
        M_w = 0.018016 #molecular mass of water
        m_dot_cond = 1.4 #defines the total mass flow rate of air used to cool condenser
[kg/s]
        T_h_array_AC = []
        x_h_array_AC = []
        T_duct_array_AC = []
        x_duct_array_AC = []
        T_return_array_AC = []
        x_return_array_AC = []

        #models the mixing of supply air at the beginning of the process
        x_AH = (m_dot_vent*x_outside + (m_dot_supply -
m_dot_vent)*x_i)/(m_dot_supply)
        T_air_AH = (m_dot_vent*T_outside*(CP.PropsSI('C','T',(T_air_i +
T_outside)/2.,'P',101325,"Air") + x_outside*CP.PropsSI('C','T',(T_air_i +
T_outside)/2.,'Q',1,"Water")) + (m_dot_supply -
m_dot_vent)*T_air_i*(CP.PropsSI('C','T',(T_air_i + T_outside)/2.,'P',101325,"Air") +
x_i*CP.PropsSI('C','T',(T_air_i +
T_outside)/2.,'Q',1,"Water")))/(m_dot_supply*(CP.PropsSI('C','T',(T_air_i +
T_outside)/2.,'P',101325,"Air") + x_AH*CP.PropsSI('C','T',(T_air_i +
T_outside)/2.,'Q',1,"Water")))
        T_air_to_AC = T_air_AH
        x_to_AC = x_AH

        #Calls the vapor compression model
        [T_air_o,x_air_o,m_dot_supply,P_air,W_dot_comp,T_1_AC,T_3_AC,T_4_AC,s
_1_AC,s_2_AC,s_3_AC,s_4_AC,s_g_AC,P_evap_AC,P_cond_AC,h_1_AC,h_2_AC,h_
3_AC,h_4_AC,Q_dot_cool,s_array,T_array,h_array,P_array,dummy] =
AC(T_air_to_AC,x_to_AC,T_outside,x_outside,T_air_calibrate_o,print_query,sys_confi
g,m_dot_supply,m_dot_cond)

        rho_a = CP.PropsSI('D','T',T_air_i,'P',P_air,"Air") #density of dry air [kg/m^3]
        m_w = V_air*rho_a*(1. + x_i)/((1. + x_i*M_a/M_w)*(1. + 1./x_i)) #mass of
water in air [kg]
        m_w_initial = m_w
        rho_tot = CP.PropsSI('D','T',T_air_i,'P',P_air,"Air")*(1 + x_i)/((1 +
x_i*M_a/M_w)) #density of moist air [kg/m^3]
        m_h = V_air*rho_tot #mass of moist air [kg]
        m_a = m_h - m_w #mass of dry air [kg]
        m_a_initial = m_a
        delta_t = 1. #time step [s]
        t_f_AC = 0
        T_duct = T_air_i #initial temperature of the duct [K]
        U_duct = (h_duct_i*h_duct_o)/(h_duct_i + h_duct_o) #heat transfer coefficient of
the duct [W/m^2K]
        T_ss = T_air_i + (T_air_o - T_air_i)*math.exp(-
U_duct*A_duct/(m_dot_supply*((CP.PropsSI('C','T',T_air_i,'P',P_air,"Air") +
CP.PropsSI('C','T',T_air_o,'P',P_air,"Air"))/2. +
x_air_o*(CP.PropsSI('C','T',T_air_i,'Q',1,"Water") +
CP.PropsSI('C','T',T_air_o,'Q',1,"Water"))/2.))) #steady state temperature of the duct [K]
        T_h_array_AC.append(T_air_i)
        x_h_array_AC.append(x_i)
        T_duct_array_AC.append(T_duct)
        x_duct_array_AC.append(x_air_o)
        T_return_array_AC.append(T_air_i)
        x_return_array_AC.append(x_i)
        delta_m_h2o = 0. #initial amount of water vapor that has been condensed from air
[kg]

        #the transient loop for the air within the house
        while T_air_i >= (T_set - 5./9.):
                m_w = m_w + delta_t*(m_dot_supply*(x_air_o - x_initial)  +
indoor_evap_rate) #mass of water in air at new time step
                x_i = m_w/m_a #humidity ratio at new time step
                T_duct = T_duct + delta_t*(T_ss - T_duct)/(h_duct_i*C_duct/((h_duct_i +
h_duct_o)*m_dot_supply*((CP.PropsSI('C','T',T_air_i,'P',P_air,"Air") +
CP.PropsSI('C','T',T_air_o,'P',P_air,"Air"))/2. +
x_air_o*(CP.PropsSI('C','T',T_air_i,'Q',1,"Water") +
CP.PropsSI('C','T',T_air_o,'Q',1,"Water"))/2.))) #temperature of duct at new time step
                T_duct_array_AC.append(T_duct)
                T_air_i =
(m_dot_supply*T_duct*(CP.PropsSI('C','T',T_set,'P',P_air,"Air") +
x_air_o*CP.PropsSI('C','T',T_set,'Q',1,"Water"))*(t_f_AC + delta_t) + (m_a_initial -
m_dot_supply*(t_f_AC +
delta_t))*T_air_initial*(CP.PropsSI('C','T',T_set,'P',P_air,"Air") +
x_initial*CP.PropsSI('C','T',T_set,'Q',1,"Water")) +
V_solid*c_p_solid*T_air_initial)/(m_a*(CP.PropsSI('C','T',T_set,'P',P_air,"Air") +
x_i*CP.PropsSI('C','T',T_set,'Q',1,"Water")) + V_solid*c_p_solid) +
UAs_house*(T_outside - T_air_i)/((m_a*(CP.PropsSI('C','T',T_set,'P',P_air,"Air") +
x_i*CP.PropsSI('C','T',T_set,'Q',1,"Water")) + V_solid*c_p_solid)*delta_t) #average
temperature within the house at new time step
                T_h_array_AC.append(float(T_air_i))
                x_h_array_AC.append(x_i)
                x_duct_array_AC.append(x_air_o)
                T_return_array_AC.append(T_return_array_AC[0])
                x_return_array_AC.append(x_return_array_AC[0])
                delta_m_h2o = delta_m_h2o + m_dot_supply*(x_AH - x_air_o)*delta_t
#amount of water dehumidified within the time step
                t_f_AC = t_f_AC + delta_t #total time after time step
        Q_dot_cool = m_dot_supply*((CP.PropsSI('H','T',T_air_initial,'P',P_air,"Air") +
x_initial*CP.PropsSI('H','T',T_air_initial,'Q',1,"Water")) -
(CP.PropsSI('H','T',T_air_o,'P',P_air,"Air") +
x_air_o*CP.PropsSI('H','T',T_air_o,'Q',1,"Water"))) #rate cooling experienced by the
conditioned space
        W_AC = W_dot_comp*t_f_AC #required compressor power
        Q_AC_cool = Q_dot_cool*t_f_AC #total cooling energy

        ##The following section models the desiccant dehumidification + vapor
compression cooling config
        sys_config = "Desiccant"
        T_air_i = T_air_initial
        T_duct = T_air_initial
        m_w = m_w_initial
        x_i = x_initial
        T_h_array_des = []
        x_h_array_des = []
        T_duct_array_des = []
        x_duct_array_des = []
        T_return_array_des = []
        x_return_array_des = []

        #mixing of the process air
        x_HX_preheat_i = (m_dot_vent*x_initial + (m_dot_supply -
m_dot_vent)*x_outside)/(m_dot_supply)
        T_HX_preheat_i = ((m_dot_supply -
m_dot_vent)*T_outside*(CP.PropsSI('C','T',(T_air_initial +
T_outside)/2.,'P',101325,"Air") + x_outside*CP.PropsSI('C','T',(T_air_initial +
T_outside)/2.,'Q',1,"Water")) +
m_dot_vent*T_air_initial*(CP.PropsSI('C','T',(T_air_initial +
T_outside)/2.,'P',101325,"Air") + x_i*CP.PropsSI('C','T',(T_air_initial +
T_outside)/2.,'Q',1,"Water")))/(m_dot_supply*(CP.PropsSI('C','T',(T_air_initial +
T_outside)/2.,'P',101325,"Air") + x_HX_preheat_i*CP.PropsSI('C','T',(T_air_initial +
T_outside)/2.,'Q',1,"Water")))

        #modeling the dehumidifier for the desiccant system
        [T_air_to_HX] = Dehum(x_AH,T_air_AH,x_air_o,101325)
        x_to_AC = x_air_o

        #modeling the heat reclamation at the heat exchanger
```

```
	[T_air_to_AC,T_HX_preheat_o] = 
HX(m_dot_supply,m_dot_supply,P_air,P_air,T_air_to_HX,T_HX_preheat_i,x_to_AC,x
_HX_preheat_i,"Air","Air",0.99)

	#modeling the cooling at the vapor compression cooler
	C_p_regen = m_dot_supply*(CP.PropsSI('C','T',T_HX_preheat_o,'P',P_air,"Air")
+ x_HX_preheat_i*CP.PropsSI('C','T',T_HX_preheat_o,'Q',1,"Water")) #heat rate of
regeneration air [W/K]
	[T_air_o,x_air_o,m_dot_supply,P_air,W_dot_comp,T_1_des,T_3_des,T_4_des,s
_1_des,s_2_des,s_3_des,s_4_des,s_g_des,P_evap_des,P_cond_des,h_1_des,h_2_des,h_3
_des,h_4_des,Q_dot_cool,dummy1,dummy2,dummy3,dummy4,dummy5] = 
AC(T_air_to_AC,x_to_AC,T_outside,x_outside,T_air_calibrate_o,print_query,sys_confi
g,m_dot_supply,m_dot_cond)
	T_duct_array_des = T_duct_array_AC
	T_h_array_des = T_h_array_AC
	x_h_array_des = x_h_array_AC
	x_duct_array_des = x_duct_array_AC
	T_return_array_des = T_return_array_AC
	x_return_array_des = x_return_array_AC
	t_f_des = t_f_AC

	Psyplot(sys_config,cooling_mode,T_air_AH,T_air_to_HX,T_air_to_AC,T_air_o,
x_AH,x_to_AC,0,x_air_o) #creating psychrometric chart
	W_des = W_dot_comp*t_f_des
	Q_des_cool = Q_dot_cool*t_f_des

	##The following section models the NIPAAm dehumidification + vapor
compression cooling config
	sys_config = "NIPAAm"
	T_air_i = T_air_initial
	T_duct = T_air_initial
	m_w = m_w_initial
	x_i = x_initial
	T_h_array_NIPAAm = []
	x_h_array_NIPAAm = []
	T_duct_array_NIPAAm = []
	x_duct_array_NIPAAm = []
	T_return_array_NIPAAm = []
	x_return_array_NIPAAm = []
	[T_air_to_HX] = Dehum(x_AH,T_air_AH,x_air_o,101325)
	x_HX_preheat_i = (m_dot_vent*x_initial + (m_dot_supply - 
m_dot_vent)*x_outside)/(m_dot_supply)
	x_to_AC = x_air_o
	[T_air_to_AC,T_HX_preheat_o] = 
HX(m_dot_supply,m_dot_supply,P_air,P_air,T_air_to_HX,T_HX_preheat_i,x_to_AC,x
_HX_preheat_i,"Air","Air",0.99)
	[T_air_o,x_air_o,m_dot_supply,P_air,W_dot_comp,T_1_NIPAAm,T_3_NIPAA
m,T_4_NIPAAm,s_1_NIPAAm,s_2_NIPAAm,s_3_NIPAAm,s_4_NIPAAm,s_g_NIPA
Am,P_evap_NIPAAm,P_cond_NIPAAm,h_1_NIPAAm,h_2_NIPAAm,h_3_NIPAAm,h
_4_NIPAAm,Q_dot_cool,dummy1,dummy2,dummy3,dummy4,COP_AC_NIPAAm] = 
AC(T_air_to_AC,x_to_AC,T_outside,x_outside,T_air_calibrate_o,print_query,sys_confi
g,m_dot_supply,m_dot_cond)
	T_duct_array_NIPAAm = T_duct_array_AC
	T_h_array_NIPAAm = T_h_array_AC
	x_h_array_NIPAAm = x_h_array_AC
	x_duct_array_NIPAAm = x_duct_array_AC
	T_return_array_NIPAAm = T_return_array_AC
	x_return_array_NIPAAm = x_return_array_AC
	t_f_NIPAAm = t_f_AC
	Psyplot(sys_config,cooling_mode,T_air_AH,T_air_to_HX,T_air_to_AC,T_air_o,
x_AH,x_to_AC,0,x_air_o)
	W_NIPAAm = W_dot_comp*t_f_NIPAAm

	#determining the absorption capacity of the NIPAAm based on temperature and 
humidity ratio
	C_NIPAAm_matrix = [[0, 0.17, 0.22, 0.25, 0.3, 0.37, 0.47, 0.74, 1.02], #Row for 
21 deg C; columns correspond to inlet air relative humidites of 0, 20, 30, 40, 50, 60, 70, 
80, 90 %RH
	[0, 0.12, 0.18, 0.22, 0.26, 0.31, 0.38, 0.48, 0.90], #Row for 25 deg C
	[0, 0.12, 0.17, 0.20, 0.24, 0.27, 0.31, 0.36, 0.47], #Row for 30
	[0, 0.10, 0.14, 0.16, 0.19, 0.22, 0.25, 0.28, 0.31], #Row for 35
	[0, 0.06, 0.10, 0.12, 0.14, 0.16, 0.19, 0.21, 0.23], #Row for 40
	[0, 0.03, 0.05, 0.06, 0.07, 0.09, 0.10, 0.11, 0.12], #Row for 50
	[0, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]]#Row for 60 deg C;
	RH_array = [0, 20, 30, 40, 50, 60, 70, 80, 90]
	RH_1 = int(math.floor(RH(T_air_AH,T_s(x_AH,P_air))*100/10.) - 1)
	RH_2 = int(math.ceil(RH(T_air_AH,T_s(x_AH,P_air))*100/10.) - 1)
	if RH_1 == -1:
		RH_1 = 0
	else:
		1
	if T_air_AH < 21 + 273.15:
		1
	elif T_air_AH > 21 + 273.15 and T_air_AH < 25 + 273.15:
		C_1 = float(interp1d([RH_array[RH_1], 
RH_array[RH_2]],[C_NIPAAm_matrix[0][RH_1], 
C_NIPAAm_matrix[0][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
```

```
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[1][RH_1],
C_NIPAAm_matrix[1][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([21 + 273.15, 25 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 25 + 273.15 and T_air_AH < 30 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[1][RH_1],
C_NIPAAm_matrix[1][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[2][RH_1],
C_NIPAAm_matrix[2][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([25 + 273.15, 30 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 30 + 273.15 and T_air_AH < 35 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[2][RH_1],
C_NIPAAm_matrix[2][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[3][RH_1],
C_NIPAAm_matrix[3][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([30 + 273.15, 35 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 35 + 273.15 and T_air_AH < 40 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[3][RH_1],
C_NIPAAm_matrix[3][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[4][RH_1],
C_NIPAAm_matrix[4][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([35 + 273.15, 40 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 40 + 273.15 and T_air_AH < 50 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[4][RH_1],
C_NIPAAm_matrix[4][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[5][RH_1],
C_NIPAAm_matrix[5][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([40 + 273.15, 50 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 50 + 273.15 and T_air_AH < 60 + 273.15:
```

```
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[5][RH_1],
C_NIPAAm_matrix[5][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[6][RH_1],
C_NIPAAm_matrix[6][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([50 + 273.15, 60 + 273.15],[C_1,
C_2])(T_air_AH))
          else:
                    1
          delta_C_NIPAAm = 0.6*C_NIPAAm #relative water content increase during
dehumidification [kg_water/kg_NIPAAm]
          c_p_NIPAAm_wet =
(CP.PropsSI('C','T',T_regen_NIPAAm,'Q',0,"water")*C_NIPAAm + c_p_NIPAAm)
#specific heat of NIPAAm when saturated
          c_p_NIPAAm_dry =
(CP.PropsSI('C','T',T_regen_NIPAAm,'Q',0,"water")*(C_NIPAAm - delta_C_NIPAAm)
+ c_p_NIPAAm) #specific heat of NIPAAm when it is as dried as possible
          h_fg_NIPAAm = (CP.PropsSI('H','T',T_regen_NIPAAm,'Q',1,"Water") -
CP.PropsSI('H','T',T_regen_NIPAAm,'Q',0,"Water")) #heat of evaporation for water
          Q_NIPAAm_cool = Q_dot_cool*t_f_NIPAAm
          Q_useful = 1
          omega = 0.75
          m_NIPAAm = m_dot_supply*(x_AH - x_air_o)*360/(omega*delta_C_NIPAAm)
#necessary mass of NIPAAm
          m_des = m_dot_supply*(x_AH - x_air_o)*360/(omega*C_des) #necessary mass
of desiccant
          return [delta_t, t_f_AC, T_h_array_AC, x_h_array_AC, T_duct_array_AC,
x_duct_array_AC, T_return_array_AC, x_return_array_AC, T_h_array_des,
x_h_array_des, T_duct_array_des, x_duct_array_des, T_return_array_des,
x_return_array_des, T_h_array_NIPAAm, x_h_array_NIPAAm,
T_duct_array_NIPAAm, x_duct_array_NIPAAm, T_return_array_NIPAAm,
x_return_array_NIPAAm, delta_m_h2o, W_AC, W_des, W_NIPAAm,
c_p_NIPAAm_dry, c_p_NIPAAm_wet, h_fg_NIPAAm,
T_1_AC,T_3_AC,T_4_AC,s_1_AC,s_2_AC,s_3_AC,s_4_AC,s_g_AC,P_evap_AC,P_co
nd_AC,h_1_AC,h_2_AC,h_3_AC,h_4_AC,T_1_des,T_3_des,T_4_des,s_1_des,s_2_des,
s_3_des,s_4_des,s_g_des,P_evap_des,P_cond_des,h_1_des,h_2_des,h_3_des,h_4_des,T_
1_NIPAAm,T_3_NIPAAm,T_4_NIPAAm,s_1_NIPAAm,s_2_NIPAAm,s_3_NIPAAm,s
_4_NIPAAm,s_g_NIPAAm,P_evap_NIPAAm,P_cond_NIPAAm,h_1_NIPAAm,h_2_NI
PAAm,h_3_NIPAAm,h_4_NIPAAm,Q_AC_cool,Q_des_cool,Q_NIPAAm_cool,s_array
,T_array,h_array,P_array,T_HX_preheat_o,C_p_regen,Q_useful,omega,m_NIPAAm,CO
P_AC_NIPAAm,m_des,delta_C_NIPAAm]
```

APPENDIX D

PYTHON CODE FOR “HOUSE_AIR_EVAP_COOL.PY”

```
def house_air_evap_cool(T_set,x_i,T_outside,x_outside,T_air_o,x_air_o,percent_vent):
        import math
        from CoolProp import CoolProp as CP
        from scipy.optimize import fsolve
        from AC import AC
        from Dehum import Dehum
        from Psyplot import Psyplot
        from HX import HX
        from x import x
        from scipy.interpolate import interp1d
        from RH import RH
        from T_s import T_s

        P_air = 101325 #total pressure within conditioned space [Pa]
        V_tot = 271.84 #total conditioned space volume [m^3]
        T_regen_NIPAAm = 32 + 273.15 #Regen temperature of NIPAAm, used only to
find the specific heat of water within the NIPAAm; this value is redefined in GUI.py
        c_p_des = 960. #desiccant specific heat
        C_des = 0.4 #absorption capacity of desiccant in kg_water/kg_des
        c_p_NIPAAm = 960. #NIPAAm specific heat
        percent_solid_vol = 0.005 #percent of the room volume that is solid
        V_air = V_tot*(1 - percent_solid_vol) #volume of air within the room
        V_solid = V_tot*percent_solid_vol #volume of solid within the room
        c_p_solid = 903600 #volumetric heat capacity of the solid, J/(m^3*K)
        T_air_i = T_set + 5./9. #initial temperature of air within the house [K]
        indoor_evap_rate = 0. #no indoor evaporation
        UAs_house = 0 #sets the heat transfer coefficient for house heat gain to zero
        h_duct_i = 8.33 #duct interior heat transfer coefficient [W/m^2K]
        h_duct_o = 0. #duct exterior heat transfer coefficient [W/m^2K]
        L_duct = 9.14 #length of duct [m]
        D_duct = 0.1016 #diameter of duct [m]
        A_duct = math.pi*D_duct*L_duct #surface area of duct [m^2]
        C_duct = 470*6.404*L_duct #heat capacity of duct [J/K]
        m_dot_supply = 0.7 #defines the total mass flow rate of supply air [kg/s]
        m_dot_vent = percent_vent*m_dot_supply/100. #portion of supply air that comes
from outside [kg/s]
        cooling_mode = "Evap"
        delta_t = 1 #time step [s]

        ##The following section models the desiccant dehumidification + evaporative
cooling config
        sys_config = "Desiccant"
        print_query = "no"
        M_a = 0.028964 #molecular mass of air
```

```
	M_w = 0.018016 #molecular mass of water
	T_duct = T_air_i #initial temperature of the duct
	T_air_initial = T_air_i
	x_initial = x_i

	#models the mixing of supply air at the beginning of the process
	x_AH = (m_dot_vent*x_outside + (m_dot_supply - 
m_dot_vent)*x_i)/(m_dot_supply)
	T_air_AH = (m_dot_vent*T_outside*(CP.PropsSI('C','T',(T_air_i + 
T_outside)/2.,'P',101325,"Air") + x_outside*CP.PropsSI('C','T',(T_air_i + 
T_outside)/2.,'Q',1,"Water")) + (m_dot_supply - 
m_dot_vent)*T_air_i*(CP.PropsSI('C','T',(T_air_i + T_outside)/2.,'P',101325,"Air") + 
x_i*CP.PropsSI('C','T',(T_air_i + 
T_outside)/2.,'Q',1,"Water")))/(m_dot_supply*(CP.PropsSI('C','T',(T_air_i + 
T_outside)/2.,'P',101325,"Air") + x_AH*CP.PropsSI('C','T',(T_air_i + 
T_outside)/2.,'Q',1,"Water")))
	h_air_o = CP.PropsSI('H','T',T_air_o,'P',101325,"Air") + 
x_air_o*CP.PropsSI('H','T',T_air_o,'Q',1,"Water")

	#mixing of the process air
	x_HX_preheat_i = (m_dot_vent*x_initial + (m_dot_supply - 
m_dot_vent)*x_outside)/(m_dot_supply)
	T_HX_preheat_i = ((m_dot_supply - 
m_dot_vent)*T_outside*(CP.PropsSI('C','T',(T_air_initial + 
T_outside)/2.,'P',101325,"Air") + x_outside*CP.PropsSI('C','T',(T_air_initial + 
T_outside)/2.,'Q',1,"Water")) + 
m_dot_vent*T_air_initial*(CP.PropsSI('C','T',(T_air_initial + 
T_outside)/2.,'P',101325,"Air") + x_i*CP.PropsSI('C','T',(T_air_initial + 
T_outside)/2.,'Q',1,"Water")))/(m_dot_supply*(CP.PropsSI('C','T',(T_air_initial + 
T_outside)/2.,'P',101325,"Air") + x_HX_preheat_i*CP.PropsSI('C','T',(T_air_initial + 
T_outside)/2.,'Q',1,"Water")))

	#iteratively solves for the necessary dehumidifier outlet humidity
	def equations(x_new):
		eq_1 = 
CP.PropsSI('H','T',float(HX(m_dot_supply,m_dot_supply,P_air,P_air,float(Dehum(x_A
H,T_air_AH,float(x_new),101325)[0]),T_HX_preheat_i,float(x_new),x_HX_preheat_i,"
Air","Air",0.80)[0]),'P',101325,"Air") + 
float(x_new)*CP.PropsSI('H','T',(HX(m_dot_supply,m_dot_supply,P_air,P_air,Dehum(x
_AH,T_air_AH,float(x_new),101325)[0],T_HX_preheat_i,float(x_new),x_HX_preheat_i
,"Air","Air",0.80)[0]),'Q',1,"Water") - h_air_o
		return(eq_1)
	[x_dehum] = fsolve(equations, x_air_o - 0.0001)
	#models the heat exchanger
	[T_air_to_HX] = Dehum(x_AH,T_air_AH,x_dehum,101325)
	[T_air_to_AC,T_HX_preheat_o] =
HX(m_dot_supply,m_dot_supply,P_air,P_air,T_air_to_HX,T_HX_preheat_i,x_dehum,x
_HX_preheat_i,"Air","Air",0.80)

	#the following definitions are for the transient modeling
	rho_a = CP.PropsSI('D','T',T_air_i,'P',P_air,"Air") #density of dry air [kg/m^3]
	m_w = V_air*rho_a*(1. + x_i)/((1. + x_i*M_a/M_w)*(1. + 1./x_i)) #mass of
water in air [kg]
	rho_tot = CP.PropsSI('D','T',T_air_i,'P',P_air,"Air")*(1 + x_i)/((1 +
x_i*M_a/M_w)) #density of moist air [kg/m^3]
	m_h = V_air*rho_tot #mass of moist air [kg]
	m_a = m_h - m_w #mass of dry air [kg]
	m_a_initial = m_a
	U_duct = (h_duct_i*h_duct_o)/(h_duct_i + h_duct_o) #heat transfer coefficient of
the duct [W/m^2K]
	T_ss = T_air_i + (T_air_o - T_air_i)*math.exp(-
U_duct*A_duct/(m_dot_supply*((CP.PropsSI('C','T',T_air_i,'P',P_air,"Air") +
CP.PropsSI('C','T',T_air_o,'P',P_air,"Air"))/2. +
x_air_o*(CP.PropsSI('C','T',T_air_i,'Q',1,"Water") +
CP.PropsSI('C','T',T_air_o,'Q',1,"Water"))/2.))) #steady state temperature of the duct [K]
	T_h_array_des = []
	x_h_array_des = []
	T_duct_array_des = []
	x_duct_array_des = []
	T_return_array_des = []
	x_return_array_des = []
	C_p_regen = m_dot_supply*(CP.PropsSI('C','T',T_HX_preheat_o,'P',P_air,"Air")
+ x_HX_preheat_i*CP.PropsSI('C','T',T_HX_preheat_o,'Q',1,"Water")) #heat rate of
regeneration air  [W/K]

	t_f_des = 0. #initializing the total cooling time [s]

	Psyplot(sys_config,cooling_mode,T_air_AH,T_air_to_HX,T_air_to_AC,T_air_o,
x_AH,x_dehum,x_dehum,x_air_o) #creating psychrometric chart
	h2o_des = 0. #initializing the amount of water consumed by the evaporative
cooler [kg]
	delta_m_h2o = 0. #initializing the amount of water absorbed during
dehumidification [kg]
	m_h2o_used = 0.
	T_h_array_des.append(T_air_i)
	x_h_array_des.append(x_i)
	T_duct_array_des.append(T_duct)
	x_duct_array_des.append(x_air_o)
	T_return_array_des.append(T_air_i)
	x_return_array_des.append(x_i)

	#transient model
	while T_air_i >= (T_set - 5./9.):
		m_w = m_w + delta_t*(m_dot_supply*(x_air_o - x_initial)  + 
indoor_evap_rate) #mass of water in air at new time step
		x_i = m_w/m_a #humidity ratio at new time step
		T_duct = T_duct + delta_t*(T_ss - T_duct)/(h_duct_i*C_duct/((h_duct_i + 
h_duct_o)*m_dot_supply*((CP.PropsSI('C','T',T_air_i,'P',P_air,"Air") + 
CP.PropsSI('C','T',T_air_o,'P',P_air,"Air"))/2. + 
x_air_o*(CP.PropsSI('C','T',T_air_i,'Q',1,"Water") + 
CP.PropsSI('C','T',T_air_o,'Q',1,"Water"))/2.))) #temperature of duct at new time step
		T_duct_array_des.append(T_duct)
		T_air_i = 
(m_dot_supply*T_duct*(CP.PropsSI('C','T',T_set,'P',P_air,"Air") + 
x_air_o*CP.PropsSI('C','T',T_set,'Q',1,"Water"))*(t_f_des + delta_t) + (m_a_initial - 
m_dot_supply*(t_f_des + 
delta_t))*T_air_initial*(CP.PropsSI('C','T',T_set,'P',P_air,"Air") + 
x_initial*CP.PropsSI('C','T',T_set,'Q',1,"Water")) + 
V_solid*c_p_solid*T_air_initial)/(m_a*(CP.PropsSI('C','T',T_set,'P',P_air,"Air") + 
x_i*CP.PropsSI('C','T',T_set,'Q',1,"Water")) + V_solid*c_p_solid) #average temperature 
within the house at new time step
		T_h_array_des.append(float(T_air_i))
		x_h_array_des.append(x_i)
		x_duct_array_des.append(x_air_o)
		T_return_array_des.append(T_return_array_des[0])
		x_return_array_des.append(x_return_array_des[0])
		delta_m_h2o = delta_m_h2o + m_dot_supply*(x_AH - x_dehum)*delta_t
		m_h2o_used = m_h2o_used + m_dot_supply*(x_air_o - 
x_dehum)*delta_t
		h2o_des = h2o_des + m_dot_supply*(x_air_o - x_dehum)*delta_t
		#[T_air_to_AC,x_to_AC] = Dehum(T_air_i,x_i,x_to_AC)
		#[T_air_o,x_air_o,m_dot_supply,P_air] = 
AC(T_air_calibrate,x_air_calibrate,T_outside_calibrate,x_outside_calibrate,T_air_calibra
te_o,T_air_to_AC,x_to_AC,T_outside,x_outside,print_query)
		t_f_des = t_f_des + delta_t #total time after time step



	#the overall process for the NIPAAm is the same, the only difference is the 
required regeneration heat, which is modeled in GUI.py
	sys_config = "NIPAAm"
```

```
        T_air_i = T_air_initial
        T_duct = T_air_initial

        T_h_array_NIPAAm = T_h_array_des
        x_h_array_NIPAAm = x_h_array_des
        T_duct_array_NIPAAm = T_duct_array_des
        x_duct_array_NIPAAm = x_duct_array_des
        T_return_array_NIPAAm = T_return_array_des
        x_return_array_NIPAAm = x_return_array_des

        t_f_NIPAAm = t_f_des
        h2o_NIPAAm = h2o_des

        Psyplot(sys_config,cooling_mode,T_air_AH,T_air_to_HX,T_air_to_AC,T_air_o,
x_AH,x_dehum,x_dehum,x_air_o) #creates plot for NIPAAm

        #determining the absorption capacity of the NIPAAm based on temperature and
humidity ratio
        C_NIPAAm_matrix = [[0, 0.17, 0.22, 0.25, 0.3, 0.37, 0.47, 0.74, 1.02], #Row for
21 deg C; columns correspond to inlet air relative humidites of 0, 20, 30, 40, 50, 60, 70,
80, 90 %RH
        [0, 0.12, 0.18, 0.22, 0.26, 0.31, 0.38, 0.48, 0.90], #Row for 25 deg C
        [0, 0.12, 0.17, 0.20, 0.24, 0.27, 0.31, 0.36, 0.47], #Row for 30
        [0, 0.10, 0.14, 0.16, 0.19, 0.22, 0.25, 0.28, 0.31], #Row for 35
        [0, 0.06, 0.10, 0.12, 0.14, 0.16, 0.19, 0.21, 0.23], #Row for 40
        [0, 0.03, 0.05, 0.06, 0.07, 0.09, 0.10, 0.11, 0.12], #Row for 50
        [0, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]]#Row for 60 deg C;
        RH_array = [0, 20, 30, 40, 50, 60, 70, 80, 90]
        RH_1 = int(math.floor(RH(T_air_AH + 273.15,T_s(x_AH,P_air))*100/10.) - 1)
        RH_2 = int(math.ceil(RH(T_air_AH,T_s(x_AH,P_air))*100/10.) - 1)
        if RH_1 == -1:
                RH_1 = 0
        else:
                1
        if T_air_AH < 21 + 273.15:
                1
        elif T_air_AH > 21 + 273.15 and T_air_AH < 25 + 273.15:
                C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[0][RH_1],
C_NIPAAm_matrix[0][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[1][RH_1],
C_NIPAAm_matrix[1][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
```

```
                    C_NIPAAm = float(interp1d([21 + 273.15, 25 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 25 + 273.15 and T_air_AH < 30 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[1][RH_1],
C_NIPAAm_matrix[1][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[2][RH_1],
C_NIPAAm_matrix[2][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([25 + 273.15, 30 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 30 + 273.15 and T_air_AH < 35 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[2][RH_1],
C_NIPAAm_matrix[2][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[3][RH_1],
C_NIPAAm_matrix[3][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([30 + 273.15, 35 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 35 + 273.15 and T_air_AH < 40 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[3][RH_1],
C_NIPAAm_matrix[3][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[4][RH_1],
C_NIPAAm_matrix[4][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([35 + 273.15, 40 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 40 + 273.15 and T_air_AH < 50 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[4][RH_1],
C_NIPAAm_matrix[4][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[5][RH_1],
C_NIPAAm_matrix[5][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([40 + 273.15, 50 + 273.15],[C_1,
C_2])(T_air_AH))
          elif T_air_AH > 50 + 273.15 and T_air_AH < 60 + 273.15:
                    C_1 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[5][RH_1],
C_NIPAAm_matrix[5][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
```

```
                    C_2 = float(interp1d([RH_array[RH_1],
RH_array[RH_2]],[C_NIPAAm_matrix[6][RH_1],
C_NIPAAm_matrix[6][RH_2]])(RH(T_air_AH,T_s(x_AH,P_air))*100))
                    C_NIPAAm = float(interp1d([50 + 273.15, 60 + 273.15],[C_1,
C_2])(T_air_AH))
          else:
                    1
          delta_C_NIPAAm = 0.6*C_NIPAAm #relative water content increase during
dehumidification [kg_water/kg_NIPAAm]
          c_p_NIPAAm_wet =
(CP.PropsSI('C','T',T_regen_NIPAAm,'Q',0,"water")*C_NIPAAm + c_p_NIPAAm)
#specific heat of NIPAAm when saturated
          c_p_NIPAAm_dry =
(CP.PropsSI('C','T',T_regen_NIPAAm,'Q',0,"water")*(C_NIPAAm - delta_C_NIPAAm)
+ c_p_NIPAAm) #specific heat of NIPAAm when it is as dried as possible
          h_fg_NIPAAm = (CP.PropsSI('H','T',T_regen_NIPAAm,'Q',1,"Water") -
CP.PropsSI('H','T',T_regen_NIPAAm,'Q',0,"Water")) #heat of evaporation for water

          Q_useful = m_dot_supply*((CP.PropsSI('H','T',T_air_initial,'P',P_air,"Air") +
x_initial*CP.PropsSI('H','T',T_air_initial,'Q',1,"Water")) -
(CP.PropsSI('H','T',T_air_o,'P',P_air,"Air") +
x_air_o*CP.PropsSI('H','T',T_air_o,'Q',1,"Water")))*t_f_des
          omega = 0.75
          m_NIPAAm = m_dot_supply*(x_AH -
x_dehum)*360./(omega*delta_C_NIPAAm)
          m_des = m_dot_supply*(x_AH - x_dehum)*360./(omega*C_des)


          return [delta_t, t_f_des, T_h_array_des, x_h_array_des, T_duct_array_des,
x_duct_array_des, T_return_array_des, x_return_array_des,t_f_NIPAAm,
T_h_array_NIPAAm, x_h_array_NIPAAm, T_duct_array_NIPAAm,
x_duct_array_NIPAAm, T_return_array_NIPAAm, x_return_array_NIPAAm,
delta_m_h2o, h2o_des, h2o_NIPAAm, c_p_NIPAAm_dry, c_p_NIPAAm_wet,
h_fg_NIPAAm,T_HX_preheat_i,T_HX_preheat_o,x_HX_preheat_i,C_p_regen,T_air_A
H,T_air_to_HX,T_air_to_AC,T_air_o,x_AH,x_dehum,x_dehum,x_air_o,Q_useful,m_h2
o_used,omega,m_NIPAAm,m_des,delta_C_NIPAAm]
```

APPENDIX E

PYTHON CODE FOR “AC.PY”

```
def
AC(T_air_i,w_i,T_outside,w_outside,T_air_o,print_query,sys_config,m_dot_evap,m_dot
_cond,):
	from CoolProp import CoolProp as CP
	import numpy as np
	from PIL import Image, ImageFont, ImageDraw
	import math
	from HX_AC_evap import HX_AC_evap
	from HX_AC_cond import HX_AC_cond
	from x_s import x_s
	from T_s import T_s
	fnt = ImageFont.truetype("C:\Windows\Fonts\ARIALUNI.TTF", 50)

	img = Image.open("psychrometric.png")
	draw = ImageDraw.Draw(img)

	Refrigerant = "R134a"

	##T-s vapor dome
	# This code creates arrays of temperature and specific entropy values to be used
for plotting in the graphical interface (GUI.py)
	T_crit = CP.PropsSI('Tcrit',Refrigerant)
	T_array_1 = np.linspace(193.15,T_crit,1000)
	T_array_2 = []
	for T in T_array_1:
		if T == T_crit:
			1
		else:
			T_array_2.append(T)


	s_array_1 = []
	for T in T_array_1:
		s = CP.PropsSI('S','T',T,'Q',0,Refrigerant)
		s_array_1.append(s)

	s_array_2 = []
	for T in T_array_2:
		s = CP.PropsSI('S','T',T,'Q',1,Refrigerant)
		s_array_2.append(s)

	s_array = s_array_1 + list(np.flipud(s_array_2))
	T_array = list(T_array_1) + list(np.flipud(T_array_2))
```

```
    ##P-h vapor dome
    # This code creates arrays of Pressure and specific enthalpy values to be used for
plotting in the graphical interface (GUI.py)
    P_array_1 = np.linspace(100000,4020000.88,1001)
    P_array_2 = P_array_1


    h_array_1 = []
    for P in P_array_1:
        h = CP.PropsSI('H','P',P,'Q',0,Refrigerant)
        h_array_1.append(h)

    h_array_2 = []
    for P in P_array_2:
        h = CP.PropsSI('H','P',P,'Q',1,Refrigerant)
        h_array_2.append(h)

    P_array = np.concatenate((P_array_1, np.fliplr([P_array_2])[0]), axis=0)
    h_array = np.concatenate((h_array_1, np.fliplr([h_array_2])[0]), axis=0)

    #Setting the air pressure within and outside of the house, as well as the heat
transfer coefficient of the evaporator
    P_air = 101325
    UAs_evap = 1810
    UAs = UAs_evap

    #Determining the humidity ratio of the air exiting the evaporator
    if T_s(w_i, P_air) > T_air_o:
        w_o = x_s(T_air_o, P_air)
    else:
        w_o = w_i

    #Calling the evaporator function
    [T_ref_evap, m_dot_ref, Q_evap] = HX_AC_evap(m_dot_evap, P_air, T_air_i,
w_i, T_air_o, w_o, Refrigerant, UAs)

    #Using information from the evaporator model to define various refrigerant
properties at the evaporator, where state 1 is before the evaporator and state 2 is after the
evaporator
    T_1 = T_ref_evap
    T_2 = T_ref_evap
    s_1 = CP.PropsSI('S','T',T_1,'Q',0,Refrigerant)
    s_2 = CP.PropsSI('S','T',T_2,'Q',1,Refrigerant)
    P_evap = CP.PropsSI('P','T',T_1,'Q',0,Refrigerant)
```

```
        h_1 = CP.PropsSI('H','T',T_1,'Q',0,Refrigerant)
        h_2 = CP.PropsSI('H','T',T_2,'Q',1,Refrigerant)

        #defining the isentropic efficiency of the compressor, as well as the condenser
heat transfer coefficient
        isen_eff = 0.8
        UAs_cond = 3620
        UAs = UAs_cond

        #Calling the condenser function
        [P_cond, h_4, h_3] = HX_AC_cond(m_dot_cond, P_air, T_outside, w_outside,
Q_evap, Refrigerant, UAs, h_2, s_2, isen_eff)

        #Defining the properties at the remaining states, as well as the mass flow rate of
refrigerant, refrigerant COP, and power required
        h_1 = h_4
        m_dot_ref = Q_evap/(h_2 - h_1)
        s_1 = CP.PropsSI('S','P',P_evap,'H',h_4,Refrigerant)
        s_3 = CP.PropsSI('S','P',P_cond,'H',h_3,Refrigerant)
        s_g = CP.PropsSI('S','P',P_cond,'Q',1,Refrigerant)
        s_4 = CP.PropsSI('S','P',P_cond,'H',h_4,Refrigerant)
        T_3 = CP.PropsSI('T','P',P_cond,'H',h_3,Refrigerant)
        T_cond = CP.PropsSI('T','P',P_cond,'Q',1,Refrigerant)
        T_4 = CP.PropsSI('T','P',P_cond,'Q',0,Refrigerant)

        COP_AC = (h_2 - h_1)/(h_3 - h_2)

        W_dot_comp = m_dot_ref*(h_3 - h_2)


        #The remainder of the code plots the process that the air undergoes over a
psychromtric graphic
        T_air_i_F = (T_air_i - 273.15)*9./5. + 32
        T_air_o_F = (T_air_o - 273.15)*9./5. + 32
        w_i_psy = w_i*7000
        w_o_psy = w_o*7000

        if sys_config == "AC Only":
                x_1 = 17.03*T_air_i_F - 204.60 + (7.*T_air_i_F/1500. - 14./25.)*w_i_psy
                y_1 = -(1349./210.)*w_i_psy + 1483
                T_1s = T_s(w_i_psy/7000., 101325)
                T_1s_F = (T_1s - 273.15)*9./5. + 32
                x_1s = 17.03*T_1s_F - 204.60 + (7.*T_1s_F/1500. - 14./25.)*w_i_psy
                x_3 = 17.03*T_air_o_F - 204.60 + (7.*T_air_i_F/1500. -
14./25.)*w_i_psy



                if T_air_o < T_s(w_i,101325):
                        draw.line((x_1, y_1, x_1s, y_1), fill=(255,0,0), width=5)
                        for w in range(int(math.floor(w_o_psy)), int(math.ceil(w_i_psy))):
                                T = T_s(w/7000., 101325)
                                T = (T - 273.15)*9./5. + 32
                                x_1 = 17.03*T - 204.60 + (7.*T/1500. - 14./25.)*w
                                y_1 = -(1349./210.)*w + 1483
                                w = w + 1
                                T = T_s(w/7000., 101325)
                                T = (T - 273.15)*9./5. + 32
                                x_2 = 17.03*T - 204.60 + (7.*T/1500. - 14./25.)*w
                                y_2 = -(1349./210.)*w + 1483
                                draw.line((x_1, y_1, x_2, y_2), fill=(255,0,0), width=5)
                else:
                        draw.line((x_1, y_1, x_3, y_1), fill=(255,0,0), width=5)

                img.save("output\psychrom\psychrom_AC_out.png")
        elif sys_config == "Desiccant":
                1
        elif sys_config == "NIPAAm":
                1
        else:
                1
        return
[T_air_o,w_o,m_dot_evap,P_air,W_dot_comp,T_1,T_3,T_cond,s_1,s_2,s_3,s_4,s_g,P_e
vap,P_cond,h_1,h_2,h_3,h_4,Q_evap,s_array,T_array,h_array,P_array,COP_AC]
```

APPENDIX F

PYTHON CODE FOR “HX_AC_EVAP.PY”

```
def HX_AC_evap(m_dot_air, P_air, T_air_i, w_i, T_air_o, w_o, Refrigerant, UAs):
        from CoolProp import CoolProp as CP
        import math
        from T_s import T_s
        from x_s import x_s
        from scipy.optimize import fsolve

        #defining the heat rate for the air flowing over the evaporator coils
        c_p_air = (CP.PropsSI('C','T',T_air_i,'P',P_air,"Air") +
CP.PropsSI('C','T',T_air_o,'P',P_air,"Air"))/2.
        c_p_water = (CP.PropsSI('C','T',T_air_i,'Q',1,"Water") +
CP.PropsSI('C','T',T_air_o,'Q',1,"Water"))/2.
        C_min = m_dot_air*c_p_air + m_dot_air*c_p_water*(w_i + w_o)/2.
        NTU = UAs/C_min
        eff = 1 - math.exp(-NTU)

        #solving for the evaporator temperature
        if w_i == w_o: #if there is no dehumidification
                Q_p1 = m_dot_air*(T_air_i - T_air_o)*(c_p_air + w_i*c_p_water)
                Q_p2 = 0
                Q_p3 = 0
                Q = Q_p1 + Q_p2 + Q_p3
                Q_max = Q/eff
                T_ref_evap = T_air_i - Q_max/C_min
        else: #if there is some dehumidification
                #defining the air and water vapor properties
                c_p_air_1 = (CP.PropsSI('C','T',T_air_i,'P',P_air,"Air") +
CP.PropsSI('C','T',T_s(w_i, P_air),'P',P_air,"Air"))/2.
                c_p_water_1 = (CP.PropsSI('C','T',T_air_i,'Q',1,"Water") +
CP.PropsSI('C','T',T_s(w_i, P_air),'Q',1,"Water"))/2.
                Q_p1 = m_dot_air*(T_air_i - T_s(w_i, P_air))*(c_p_air_1 +
w_i*c_p_water_1)
                c_p_air_2 = (CP.PropsSI('C','T',T_s(w_i, P_air),'P',P_air,"Air") +
CP.PropsSI('C','T',T_air_o,'P',P_air,"Air"))/2.
                c_p_water_2 = (CP.PropsSI('C','T',T_s(w_i, P_air),'Q',1,"Water") +
CP.PropsSI('C','T',T_air_o,'Q',1,"Water"))/2.
                h_fg = ((CP.PropsSI('H','T',T_s(w_i, P_air),'Q',1,"Water") -
CP.PropsSI('H','T',T_s(w_i, P_air),'Q',0,"Water")) + (CP.PropsSI('H','T',T_s(w_o,
P_air),'Q',1,"Water") - CP.PropsSI('H','T',T_s(w_o, P_air),'Q',0,"Water")))/2.

                #defining the rate of heat transfer required to bring the air to the desired
conditions
                Q_p2 = m_dot_air*(T_s(w_i, P_air) - T_air_o)*(c_p_air_2 +
c_p_water_2*(w_i + w_o)/2.)
```

```
                Q_p3 = m_dot_air*(w_i - w_o)*h_fg
                Q = Q_p1 + Q_p2 + Q_p3
                Q_max = Q/eff

                #iterative solver
                def equations(T_ref_evap):
                        eq_1 = m_dot_air*(c_p_air_2 + ((w_i + 
w_o)/2.)*c_p_water_2)*(T_s(w_i,P_air) - T_air_o) + m_dot_air*(w_i - w_o)*h_fg - 
(UAs - (m_dot_air*(c_p_air_1 + w_i*c_p_water_1)*(T_air_i - 
T_s(w_i,P_air)))/((T_air_i - T_s(w_i,P_air))/math.log((T_air_i - 
T_ref_evap)/(T_s(w_i,P_air) - T_ref_evap))))*((T_s(w_i,P_air) - 
T_air_o)/math.log((T_s(w_i,P_air) - T_ref_evap)/(T_air_o - T_ref_evap)))
                        return(eq_1)
                [T_ref_evap] = fsolve(equations, (T_air_o - 0.1)) #defining the refrigerant 
temperature at the evaporator
        h_fg = CP.PropsSI('H','T',T_ref_evap,'Q',1,Refrigerant) - 
CP.PropsSI('H','T',T_ref_evap,'Q',0,Refrigerant)
        m_dot_ref = Q/h_fg
        return [T_ref_evap, m_dot_ref, Q]
```

APPENDIX G

PYTHON CODE FOR “HX_AC_COND.PY”

```
def HX_AC_cond(m_dot_air, P_air, T_air_i, w_i, Q_evap, Refrigerant, UAs, h_2, s_2,
isen_eff):
        from CoolProp import CoolProp as CP
        import math
        from scipy.optimize import fsolve

        #defining the heat rate for the air flowing over the condenser coils
        c_p_air = CP.PropsSI('C','T',T_air_i,'P',P_air,"Air")
        c_p_water = CP.PropsSI('C','T',T_air_i,'Q',1,"Water")
        C_min = m_dot_air*c_p_air + m_dot_air*c_p_water*w_i

        #iterative solver to find the required condenser temperature
        def equations(T_cond):
                eq_1 = Q_evap*(CP.PropsSI('H','T',T_cond[-1],'Q',1,Refrigerant) -
CP.PropsSI('H','T',T_cond[-1],'Q',0,Refrigerant))/(h_2 - CP.PropsSI('H','T',T_cond[-
1],'Q',0,Refrigerant)) - (1 - math.exp(-(max(0, (UAs - (Q_evap/(h_2 -
CP.PropsSI('H','T',T_cond[-1],'Q',0,Refrigerant)))*((((CP.PropsSI('H','T',T_cond[-
1],'S',s_2,Refrigerant)) - h_2)/isen_eff + h_2) - (CP.PropsSI('H','T',T_cond[-
1],'Q',1,Refrigerant)))/(((CP.PropsSI('T','P',(CP.PropsSI('P','T',T_cond[-
1],'Q',1,Refrigerant)),'S',s_2,Refrigerant)) - T_cond[-
1])/math.log(((CP.PropsSI('T','P',(CP.PropsSI('P','T',T_cond[-
1],'Q',1,Refrigerant)),'S',s_2,Refrigerant)) - T_air_i)/(T_cond[-1] -
T_air_i)))))/C_min)))*C_min*(T_cond[-1] - T_air_i)
                return(eq_1)
        T_4 = fsolve(equations, (T_air_i + 0.1)) #solving for condenser temperature
        T_4 = T_4[-1]

        #solving for the remaining refrigerant  properties
        h_4 = CP.PropsSI('H','T',T_4,'Q',0,Refrigerant)
        P_cond = CP.PropsSI('P','T',T_4,'Q',1,Refrigerant)
        h_3s = CP.PropsSI('H','P',P_cond,'S',s_2,Refrigerant)
        h_3a = (h_3s - h_2)/isen_eff + h_2
        T_3 = CP.PropsSI('T','P',P_cond,'H',h_3a,Refrigerant)

        return [P_cond, h_4, h_3a]
```

APPENDIX H

PYTHON CODE FOR “DEHUM.PY”

```
def Dehum(x_in,T_in,x_out,P_tot):
	from CoolProp import CoolProp as CP
	import time
	from scipy.optimize import fsolve

	h_i = CP.PropsSI('H','T',T_in,'P',P_tot,"Air") +
x_in*CP.PropsSI('H','T',T_in,'Q',1,"Water") #enthalpy of the moist air entering the
dehumidifier

	#iteratively solving for the temperature of the air leaving the dehumidifier, such
that the process is isenthalpic
	def equations(T_out):
		eq_1 = CP.PropsSI('H','T',T_out,'P',P_tot,"Air") +
x_out*CP.PropsSI('H','T',T_out,'Q',1,"Water") - h_i
		return(eq_1)
	[T_out] = fsolve(equations, (T_in + 0.1))
	return [T_out]
```

APPENDIX I

PYTHON CODE FOR “PSYPLOT.PY”

```
def
Psyplot(sys_config,cooling_mode,T_air_AH,T_air_to_HX,T_air_to_AC,T_air_o,x_AH,
x_to_AC,x_dehum,x_air_o):
        from PIL import Image, ImageFont, ImageDraw
        from T_s import T_s
        import math
        fnt = ImageFont.truetype("C:\Windows\Fonts\ARIALUNI.TTF", 80)
        fnt2 = ImageFont.truetype("C:\Windows\Fonts\ARIALUNI.TTF", 50)

        #the following code creates a psychrometric chart based on the system
configuration used (VC = standard vapor compression)
        img = Image.open("psychrometric.png")
        draw = ImageDraw.Draw(img)
        if cooling_mode == "VC":
                #first the temperatures are converted to deg F, and the humidity ratios are
converted to gr/lb
                T_air_i_F = (T_air_to_AC - 273.15)*9./5. + 32
                T_air_o_F = (T_air_o - 273.15)*9./5. + 32
                T_dehum_i_F = (T_air_AH - 273.15)*9./5. + 32
                T_dehum_o_F = (T_air_to_HX - 273.15)*9./5. + 32
                w_0_psy = x_AH*7000
                w_i_psy = x_to_AC*7000
                w_o_psy = x_air_o*7000

                #x and y coordinates are created for the points in the process, based on the
psychrometric graphic over which the lines are plotted. x and y coordinates are created in
units of pixels
                x_1 = 17.03*T_air_i_F - 204.60 + (7.*T_air_i_F/1500. - 14./25.)*w_i_psy
                y_1 = -(1349./210.)*w_i_psy + 1483
                T_1s = T_s(w_i_psy/7000., 101325)
                T_1s_F = (T_1s - 273.15)*9./5. + 32
                x_1s = 17.03*T_1s_F - 204.60 + (7.*T_1s_F/1500. - 14./25.)*w_i_psy
                x_2 = 17.03*T_air_o_F - 204.60 + (7.*T_air_o_F/1500. -
14./25.)*w_i_psy

                if T_air_o < T_s(x_air_o,101325):
                        draw.line((x_1, y_1, x_1s, y_1), fill=(255,0,0), width=5)
                else:
                        draw.line((x_1, y_1, x_2, y_1), fill=(255,0,0), width=5)
                x_0 = 17.03*T_dehum_i_F - 204.60 + (7.*T_dehum_i_F/1500. -
14./25.)*w_0_psy
                y_0 = -(1349./210.)*w_0_psy + 1483
                x_0a = 17.03*T_dehum_o_F - 204.60 + (7.*T_dehum_o_F/1500. -
14./25.)*w_i_psy
```

```
                draw.line((x_0, y_0, x_0a, y_1), fill=(255,0,0), width=5)

                draw.line((x_0a, y_1, x_1, y_1), fill=(255,0,0), width=5)
                if T_air_o < T_s(x_air_o,101325):
                        for w in range(int(math.floor(w_o_psy)), int(math.ceil(w_i_psy))):
                                T = T_s(w/7000., 101325)
                                T = (T - 273.15)*9./5. + 32
                                x_3 = 17.03*T - 204.60 + (7.*T/1500. - 14./25.)*w
                                y_3 = -(1349./210.)*w + 1483
                                w = w + 1
                                T = T_s(w/7000., 101325)
                                T = (T - 273.15)*9./5. + 32
                                x_4 = 17.03*T - 204.60 + (7.*T/1500. - 14./25.)*w
                                y_4 = -(1349./210.)*w + 1483
                                draw.line((x_3, y_3, x_4, y_4), fill=(255,0,0), width=5)
                else:
                        1
                draw.text(((x_0 - 8),(y_0 - 80)), ".", font = fnt, fill = (0,0,0))
                draw.text(((x_0a - 13),(y_1 - 82)), ".", font = fnt, fill = (0,0,0))
                draw.text(((x_1 - 10),(y_1 - 81)), ".", font = fnt, fill = (0,0,0))
                draw.text(((x_2 - 7),(y_1 - 81)), ".", font = fnt, fill = (0,0,0))
                draw.text(((x_0 - 16),(y_0 - 80 + 15)), "1", font = fnt2, fill = (0,0,0))
                draw.text(((x_0a - 17),(y_1 - 82 + 70)), "2", font = fnt2, fill = (0,0,0))
                draw.text(((x_1 - 14),(y_1 - 81 + 70)), "3", font = fnt2, fill = (0,0,0))
                draw.text(((x_2 - 11),(y_1 - 81 + 70)), "4", font = fnt2, fill = (0,0,0))
        else:
                T_air_i_F = (T_air_to_AC - 273.15)*9./5. + 32
                T_air_o_F = (T_air_o - 273.15)*9./5. + 32
                T_dehum_i_F = (T_air_AH - 273.15)*9./5. + 32
                T_dehum_o_F = (T_air_to_HX - 273.15)*9./5. + 32
                w_0_psy = x_AH*7000
                w_i_psy = x_to_AC*7000
                w_o_psy = x_air_o*7000
                w_dehum_psy = x_dehum*7000
                x_1 = 17.03*T_air_i_F - 204.60 + (7.*T_air_i_F/1500. -
14./25.)*w_dehum_psy
                y_1 = -(1349./210.)*w_dehum_psy + 1483
                T_1s = T_s(w_i_psy/7000., 101325)
                T_1s_F = (T_1s - 273.15)*9./5. + 32
                x_1s = 17.03*T_1s_F - 204.60 + (7.*T_1s_F/1500. - 14./25.)*w_i_psy
                x_2 = 17.03*T_air_o_F - 204.60 + (7.*T_air_o_F/1500. -
14./25.)*w_o_psy
                y_2 = -(1349./210.)*w_o_psy + 1483
        draw.line((x_1, y_1, x_2, y_2), fill=(255,0,0), width=5)
        x_0 = 17.03*T_dehum_i_F - 204.60 + (7.*T_dehum_i_F/1500. -
14./25.)*w_0_psy
        y_0 = -(1349./210.)*w_0_psy + 1483
        x_0a = 17.03*T_dehum_o_F - 204.60 + (7.*T_dehum_o_F/1500. -
14./25.)*w_dehum_psy

        if T_dehum_o_F > 120:
            w_120 = (y_0 + ((y_0 - y_1)/(x_0a - x_0))*x_0 - 1483 -
(120*17.03 - 204.60)*((y_0 - y_1)/(x_0a - x_0)))*(-(1349./210.) + ((y_0 - y_1)/(x_0a -
x_0))*(7.*120/1500. - 14./25.))**(-1)
            x_0b = 17.03*120 - 204.60 + (7.*120/1500. - 14./25.)*w_120
            y_0b = -(1349./210.)*w_120 + 1483
            x_1a = 17.03*120 - 204.60 + (7.*120/1500. -
14./25.)*w_dehum_psy
            draw.line((x_0, y_0, x_0b, y_0b), fill=(255,0,0), width=5)

            draw.line((x_1a, y_1, x_1, y_1), fill=(255,0,0), width=5)
        else:
            draw.line((x_0, y_0, x_0a, y_1), fill=(255,0,0), width=5)

            draw.line((x_0a, y_1, x_1, y_1), fill=(255,0,0), width=5)
        if T_air_o < T_s(x_air_o,101325):
            for w in range(int(math.floor(w_o_psy)), int(math.ceil(w_i_psy))):
                T = T_s(w/7000., 101325)
                T = (T - 273.15)*9./5. + 32
                x_3 = 17.03*T - 204.60 + (7.*T/1500. - 14./25.)*w
                y_3 = -(1349./210.)*w + 1483
                w = w + 1
                T = T_s(w/7000., 101325)
                T = (T - 273.15)*9./5. + 32
                x_4 = 17.03*T - 204.60 + (7.*T/1500. - 14./25.)*w
                y_4 = -(1349./210.)*w + 1483
                draw.line((x_3, y_3, x_4, y_4), fill=(255,0,0), width=5)
        else:
            1
        draw.text(((x_0 - 8),(y_0 - 80)), ".", font = fnt, fill = (0,0,0))
        draw.text(((x_0a - 13),(y_1 - 82)), ".", font = fnt, fill = (0,0,0))
        draw.text(((x_1 - 10),(y_1 - 81)), ".", font = fnt, fill = (0,0,0))
        draw.text(((x_2 - 7),(y_2 - 81)), ".", font = fnt, fill = (0,0,0))
        draw.text(((x_0 - 16),(y_0 - 80 + 15)), "1", font = fnt2, fill = (0,0,0))
        draw.text(((x_0a - 17),(y_1 - 82 + 70)), "2", font = fnt2, fill = (0,0,0))
        draw.text(((x_1 - 14),(y_1 - 81 + 70)), "3", font = fnt2, fill = (0,0,0))
        draw.text(((x_2 - 11),(y_2 - 81 + 70)), "4", font = fnt2, fill = (0,0,0))
if sys_config == "Desiccant":
        img.save("output\psychrom\psychrom_desiccant_out.png")
elif sys_config == "NIPAAm":
        img.save("output\psychrom\psychrom_NIPAAm_out.png")
else:
        1
return []
```

APPENDIX J

PYTHON CODE FOR “HX.PY”

```
def HX(m_dot_h, m_dot_c, P_h, P_c, T_h_i, T_c_i, x_h, x_c, Fluid_h, Fluid_c, eff):
        from CoolProp import CoolProp as CP

        #determining the heat rates of the supply (h) and process (c) air
        c_p_h = CP.PropsSI('C','T',T_h_i,'P',P_h,Fluid_h) +
x_h*CP.PropsSI('C','T',T_h_i,'Q',1,"Water")
        c_p_c = CP.PropsSI('C','T',T_c_i,'P',P_c,Fluid_c) +
x_c*CP.PropsSI('C','T',T_c_i,'Q',1,"Water")
        C_max = max(m_dot_h*c_p_h, m_dot_c*c_p_c)
        C_min = min(m_dot_h*c_p_h, m_dot_c*c_p_c)


        Q_max = C_min*(T_h_i - T_c_i) #the maximum available rate of heat transfer
        Q = eff*Q_max #the actual rate of heat transfer

        T_h_o = T_h_i - Q/(m_dot_h*c_p_h)
        T_c_o = T_c_i + Q/(m_dot_c*c_p_c)
        return [T_h_o, T_c_o]
```

APPENDIX K

PYTHON CODE FOR “RH.PY”

```
def RH(T, T_d):
    import math
    m = 17.625 #constant
    T_n = 243.04 #constant
    T = T - 273.15 #actual air temperature
    T_d = T_d - 273.15 #dew point temperature
    RH = math.exp(m*(((T_d)/(T_d + T_n)) - ((T)/(T + T_n)))) #relative humidity
    return RH
```

APPENDIX L

PYTHON CODE FOR “T_S.PY”

```
def T_s(x, P_tot):
        import math
        from scipy.optimize import fsolve
        P_vs = (x*P_tot/0.6219907)/(1 + x/0.6219907) #saturation vapor pressure for
given humidity
        def equation(T):
                Eq_1 = 22064000*math.exp(647.096/T*(-7.85951783*(1 - T/647.096) +
1.84408259*(1 - T/647.096)**1.5 - 11.7866497*(1 - T/647.096)**3 + 22.6807411*(1 -
T/647.096)**3.5 - 15.9618719*(1 - T/647.096)**4 + 1.80122502*(1 - T/647.096)**7.5))
- P_vs
                return(Eq_1)
        T_s = fsolve(equation, 273.15) #saturation temperature
        T_s = T_s[0]
        return T_s
```

APPENDIX M

PYTHON CODE FOR “X.PY”

```
def x(T, RH, P_tot):
        import math
        theta = 1 - T/647.096 #temperature-based variable
        P_vs = 22064000*math.exp(647.096/T*(-7.85951783*theta +
1.84408259*theta**1.5 - 11.7866497*theta**3 + 22.6807411*theta**3.5 -
15.9618719*theta**4 + 1.80122502*theta**7.5)) #Saturation vapor pressure [Pa]
        P_v = P_vs*RH #actual vapor pressure
        x = 0.6219907*P_v/(P_tot - P_v) #humidity ratio
        return x
```

APPENDIX N

PYTHON CODE FOR “X_S.PY”

```
def x_s(T, P_tot):
	import math
	theta = 1 - T/647.096 #temperature based variable
	P_vs = 22064000*math.exp(647.096/T*(-7.85951783*theta +
1.84408259*theta**1.5 - 11.7866497*theta**3 + 22.6807411*theta**3.5 -
15.9618719*theta**4 + 1.80122502*theta**7.5)) #Saturation vapor pressure [Pa]
	x_s = 0.6219907*P_vs/(P_tot - P_vs) #saturation humidity ratio for given
temperature
	return x_s
```