Analysis of Heat Recovery Techniques Applied to a Xylene Production Plant

Olivia Kuebler, Adrian Marusic, Luke Oluoch, and Jonathan White

November 19, 2018

Abstract

The production of xylene (an important petrochemical in a variety of industries) has been steadily increasing over the past few years. Due to this, maximizing heat recovery in large scale xylene production facilities could have significant economic benefits. After analyzing the system using pinch analysis, the Q_{hmin}, Q_{cmin} *and* T_{pinch} *values of the pilot plant analyzed are 2855.987 KW, 2824.387 KW and 254.3° C when* $\Delta\ T_{min}$ = *10° C. After integrating the network, the savings of doing so would prove to be greatly beneficial to both the company running the process as well as for the environment. By redirecting the energy appropriately, the yearly operational costs and carbon dioxide emissions can be reduced significantly.*

The heat from the streams can also be utilized to run a Carnot engine and produce work that can be used to run other processes. Using carbon dioxide as the working fluid around supercritical conditions, an engine can be used to replace 1521.768 kW that would otherwise be supplied by a cooler to produce 225.746 Joules of work per cycle of the piston cylinder system. Carbon dioxide also proved to be the most appropriate supercritical working fluid for the network as compared to ethane, methane and water. Another process of heat recovery is steam raising, where an independent cold stream of water enters the process and absorbs as much heat as possible to become steam by maximizing the incoming flow rate through an iterative process carried out in Python. Using this process, we can calculate the theoretical value taken out of the system to be 27851 kW. Lastly, the developed networks from each of the different sections were analyzed by eigenvector centrality. Eigenvector centrality is a display of the connections between parts of the network and leads to conclusions of the dependence of the network on certain portions on a scale of 0 to 1. In each of the networks, there are a few important streams that have numerous connections and each of the heat exchanger areas calculated in order to analyze feasibility and vulnerabilities within the system.

Contents

1 Introduction **1**

2 Energy Analysis **1**

3 Environmental Analysis **4**

4 Steam Raising **4**
i Integrating Steam Raising into the Heat Exchange Network . . . 6

5 Carnot Cycle **6**
i Carnot Cycle Construction . . . 6
ii Numerical and Analytical Analysis . . . 8
iii Carnot Engine Specifications . . . 8
iv Supercritical Working Fluid Comparison . . . 9
v Supercritical Working Fluid Integrations . . . 10

6 Eigenvector Centrality **11**
i Original Network . . . 11
ii Steam Raising Network . . . 13
iii Heat Exchanger Energy/Area Analysis . . . 14
iii.1 Original Network . . . 14
iii.2 Steam Raising Network . . . 14

7 Conclusion **15**

8 Bibliography **16**

9 Appendix **17**
i Original Network Flowsheet . . . 17
ii Network Flowsheet with Carnot Engine Integrate . . . 18
iii Steam Raised Network Flowsheet . . . 19
iv Supercritical Water Carnot Cycle . . . 20
v Supercritical Methane Carnot Cycle . . . 21
vi Supercritical Ethane Carnot Cycle . . . 22
vii Steam Raising Code . . . 22
viii Table of Steam Raising Variables . . . 26
ix Supercritical CO2 Carnot Cycle Code . . . 26
x Supercritical Methane Carnot Cycle Code . . . 31
xi Supercritical Ethane Carnot Cycle Code . . . 35
xii Supercritical Water Carnot Cycle Code . . . 39
xiii Pinch Calculations at Tmin of 10 degrees C Code . . . 43
xiv Pinch Calculations at Tmin of 20 degrees C Code . . . 47
xv Cost Analysis Code . . . 52
xvi Eigenvector Centrality for Original Network Code . . . 59
xvii Pinch Calculations for Steam Raising Code . . . 64
xviii Eigenvector Centrality for Steam Raising Network Code . . . 68

1. Introduction

Xylene is a volatile hydrocarbon generated in many chemical plants as a product of the catalytic disproportionation of toluene. This reaction will yield ortho(o), para(p) and meta(m) isomers of xylene. In general, xylene has a wide variety of uses such as a solvent in printing, rubber and leather industries and can also be used in cleaning agents, paint thinners and varnishes. Most p-xylene is processed into into polyethylene terephthalate (PET), a common plastic used in many industries. One example of PET's usage is found in plastic water bottles and other plastic food containers. P-xylene is also used in the manufacture of purified terephthalic acid (PTA), a basic petrochemical used in the textile industry for the production of polyester[1]. Production of this chemical has been growing for the past 15 years at a steady pace[12]. In Asia specifically, polyester producers have gotten into the purified terephthalic acid business to increase cost competitiveness and as a result over 70% of global market of xylene is in Asia. In a recent news article from Houston Business Journal, ExxonMobil Chemical Co.'s Singapore affiliate will buy one of the world's largest aromatics plants from Jurong Aromatics Corporation Pte Ltd.[9].This will allow Exxon to increase their production of paraxylene by 1.4 million additional tonnes per year. President of ExxonMobil Chemical claims, "Our growth in Singapore is driven by the expected increase in global demand for chemical products over the next decade of nearly 45%, or about 4% per year, which is a faster pace than energy demand and economic growth[9]. This data shows there will be a constant increase in the global production of xylene, specifically in Asia, and more plants will need to be built to satisfy the demand. This information gives context to the project as the plants that will be implemented are similar to the plant being analyzed, meaning the energy optimization techniques applied in this paper could be applied to actual systems in the near future.

2. Energy Analysis

The pilot plant for this process is a scaled down model for purification of benzene and xylenes. Through the process there are hot and cold streams, reboilers, and condensers that are used to produce the desired product on a large scale. While an arrangement of these streams and distillation columns work, it is not the optimal solution and has a lot of wasted energy and heat. By properly integrating the system, the wasted heat can be minimized so that the system can be optimized for the process development.

Proper heat exchange and column integration within the network begins with identifying and labeling the hot and cold streams available. At the surface of heat exchange networks, connections between hot and cold streams can be made so the energy of the hot streams can be used to heat the cold streams to their final temperatures. If there is any excess/deficit of heat such that a stream does not reach its target temperature, heaters/coolers can be applied to the system that require outside heat to power them.

For the entirety of process, it is highly preferred that a process be at steady state so that it can continually run. For that to be feasible, however, the mass and energy of running the production has to be balanced such that mass or energy in minus mass or energy out is equal to zero. For the mass balance at each section of the process can be balanced by equation 1 to show that there is no accumulation of any of the chemicals.

$$\frac{\partial m_i}{\partial t} = 0 \tag{1}$$

For the energy balance, the enthalpies of each of the sections of the stream need to be found such that it satisfies equation 2 that states that the difference between the beginning and ending enthalpies are equivalent to the enthalpy of a heat exchanger that transfers energy.

$$H_1 - H_2 = H_{exchanger} \tag{2}$$

When applied to each section of the network given, the mass and energy balances out at all parts allowing for this system to be optimized by pinch analysis.

Pinch analysis is the breakdown of understanding how to properly integrate a network based the temperature differences of hot and cold streams. The integration begins around the pinch temperature which is determined by the point where the hot and cold streams are exactly separated by T_{min} when compared graphically, meaning that there is no available heat for transfer at that exact point. The pinch temperature will decide how the network will be divided and provide a basis for establishing the connections between the streams [6].

The process that was integrated in this paper is shown in Appendix i.

The energy requirements of the pilot system at a T_{min} of 10° after integration are $Q_h = 2856$ kW, $Q_c = 2824$ kW, and $T_{pinch} = 254.3°$ (Appendix xiii). If T_{min} is set to 20°, $Q_h = 2880.4$ kW, $Q_c = 2848.8$ kW, and $T_{pinch} = 259.3°$ (Appendix xiv). For the full scale plant, each of these Q_s are multiplied by 10 to account for the difference in scale. The cost of the full scale plant can be calculated using Equations 3, 4, and 5.

$$CapitalCost = \frac{12.5 * 10^6 *^\circ C * \$}{.05 T_{min}} \tag{3}$$

$$OperationCosts = \frac{\$.15 Q_h t}{kW * hr} \tag{4}$$

$$TotalCost = \frac{12.5 * 10^6 * CC * \$}{.05T_{min}} + \frac{\$.15Q_h t}{kW * hr} \tag{5}$$

The Q_h versus T_{min} is shown in Figure 1 for T_{min} in the range of 10° to 40°.

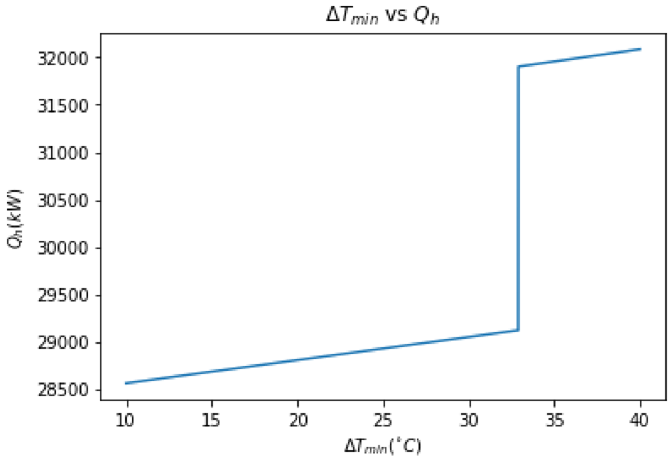


Figure 1: *ΔT_{min} vs T_{min}*

The spike in Q_h is due to a change in the pinch temperature. We can see that Q_h generally remains lowest below a T_{min} of 30°.

Capital cost versus T_{min} is shown in Figure 2 for T_{min} of 10-40°.

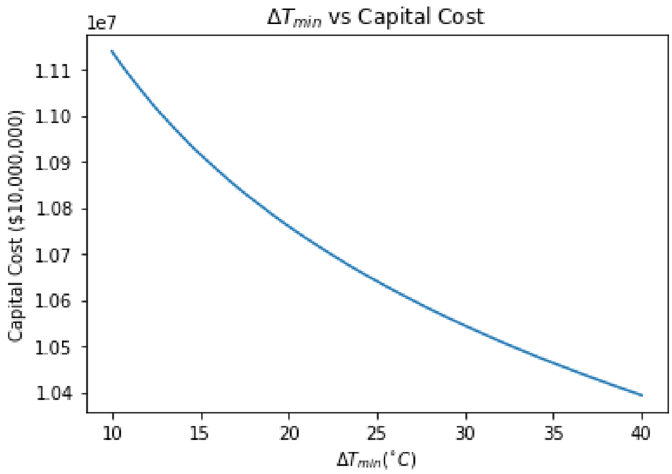


Figure 2: *ΔT_{min} vs Capitol Cost*

This shows that capital cost declines with increasing T_{min}.

Operation cost versus T_{min} for a period of 1 year is shown in Figure 3 over the same T_{min} range.

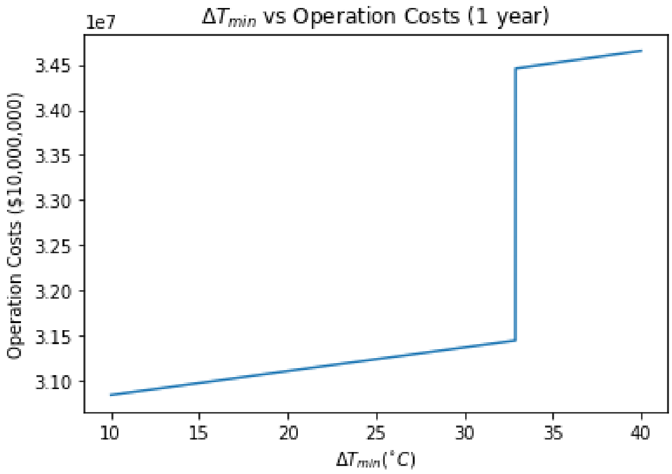


Figure 3: ΔT_{min} *vs Operational Cost*

This graph follows a similar trend to Q_h, since operation cost is a variable of Q_h and time. The ideal T_{min} to minimize operating costs for the long term is 10°.

A 3D graph of Cost vs. Time vs. T_{min} is shown in Figure 4.

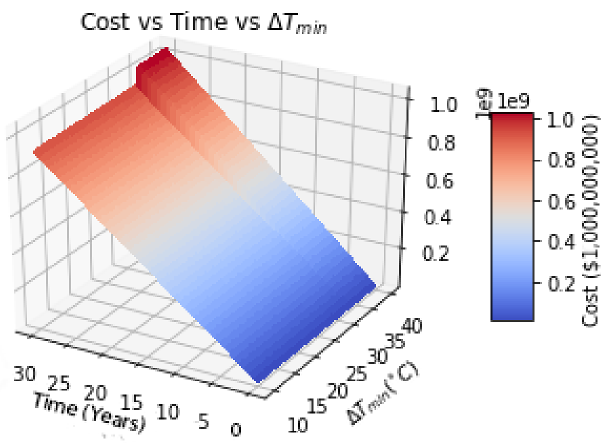


Figure 4: *3D Graph of Cost vs. Time vs.* T_{min}

This shows that while the initial cost is controlled by the capital cost, the final cost of the system will be mostly be controlled by the operating costs since these are a function of time. Because of this, depending on the number of years the plant operates, the ideal ΔT_{min} will change over time.

Figure 5 shows the total cost for one year versus T_{min}.

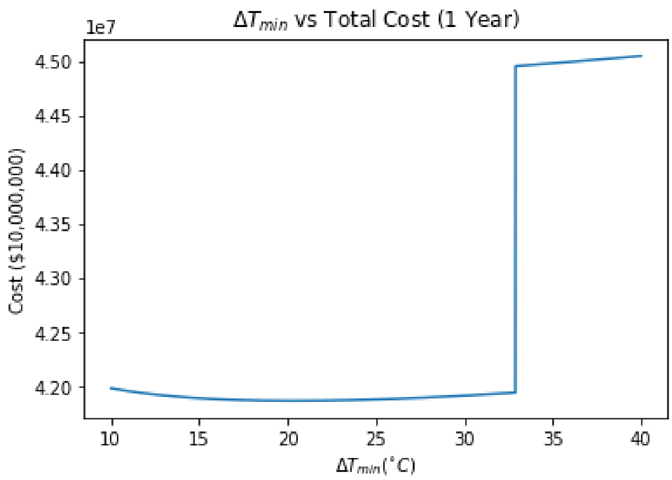


Figure 5: *Costs After One Year*

After 1 year of running the plant, we find that the lowest cost of the system would be \$41,870,000, at a T_{min} of 20.384°.

Due to observations above, a T_{min} of 10° will eventually become the most viable option at some time "t". This t was calculated in Appendix xv, along with the graphs above. From this, it can be seen that the optimal T_{min} will become 10° after 634 days (or 2 years and 34 days if running the system 300 days a year) and beyond. This means that all calculations in the following paper are most economically viable if the system runs for at least 634 days, since all following calculations assume 10°.

3. Environmental Analysis

By utilizing the heat exchange network that was assembled in section 2, there are clear advantages to being more efficient with the energy available. Instead of underutilizing the available energy in the system, by integrating the pinch analysis there are not only cost savings involved which is beneficial to the company but the reduced impact on the environment is helpful on a global scale.

To be able to do the calculations of how much is saved by integrating, there were conversion values about the pricing and energy relations of coal. The information used to find the information was:

1. Coal's energy density is 6.67 kWh/kg.
2. Coal power plants are approximately 30% efficient.
3. The cost of coal is approximately \$60.00/1000 kg coal.
4. 1 kg of coal produces approximately 1.83 kg carbon dioxide.
5. The plant runs 24 hours a day for 300 days a year.

The amount of heat that is required by the integrated network is the sum of the required heat to warm the cold streams as well as the heat necessary to run the reboilers of that system. With the integrated network, the heat needed is simply the Q_h value that is output after integration. Both of those values for heat apply to a full scale plant that is ten times larger than the pilot plant listed in the assignment details. The savings are listed in Table 1.

Table 1: *Savings Made by Integration*

	Unintegrated Plant Design	Integrated Plant Design	Savings
Total Heat Needed (kW)	42,678.00	28,559.87	14,118.13
kWh/year	307,281,600.00	205,631,078.40	101,650,521.60
Coal Needed (kg)	153,564,017.99	102,764,157.12	50,799,860.87
Cost of Coal	\$9,213,841.08	\$6,165,849.43	\$3,047991.65
Carbon Emissions (kg)	281,022,152.92	188,058,407.53	92,963,745.39

4. Steam Raising

In most factories, a significant portion of heat put into large scale processes is output as waste due to the inability of the plant to utilize that heat. Finding a way to use this waste heat would be of financial interest to any large scale manufacturing plant and pinch analysis, discussed in section 2, explains the methodology and techniques used to analyze the process system. While it is difficult to reuse low grade heat (< 100°), high grade heat (>100°) can be recycled in the form of steam. This is achieved by transferring heat between existing hot streams and streams of water or steam. In this situation the water stream is essentially a cooler since it removes heat from the system and therefore the water stream can replace coolers in the network. Due to the constraints of this system, this process can only be used below the pinch and in order to determine the constraints of the water stream, the Grand Composite Curve (GCC) of the system below the pinch can be analyzed in Figure 6.

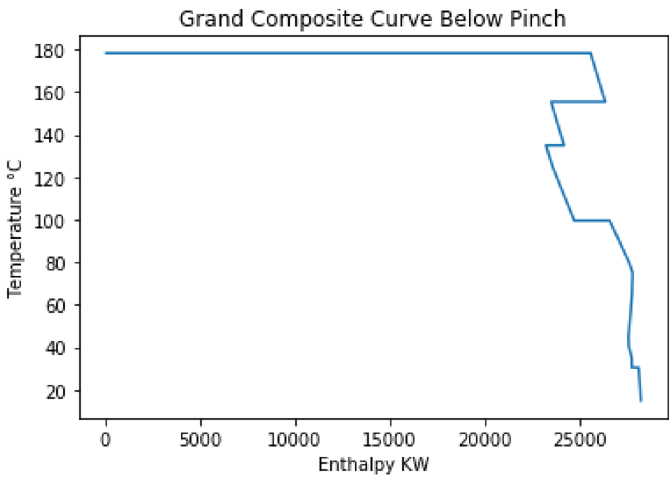


Figure 6: *Grand Composite Curve Below Pinch*

To generate the GCC graph, the interfacial temperatures from the heat cascade must be plotted with their corresponding enthalpies. By using the endpoints of a line segment and Equation 6, the points between any two endpoints can be plotted and a complete graph of line segments can be compiled using

$$\frac{T_2 - T_1}{H_2 - H_1} = \frac{T - T_1}{H - H_1} \tag{6}$$

where (H_1, T_1) and (H_2, T_2) are the end points to any of the GCC line segments and $\frac{T_2-T_1}{H_2-H_1}$ is the slope of every point between the 2 endpoints. Using the slope, the values along the GCC between the endpoints can be calculated. This method is also used to calculate the points between each steam raising line. Values for H1, H2, T1, and T2 for each segment are from the steam raising code in Appendix vii .

The GCC is a graphical representation of the heat cascade analysis, with each temperature associated with a kW value and shows, graphically, the maximum heat that can be taken from the system. Steam lines are one way to find that maximum heat. These lines are graphed on the GCC using specific thermodynamic properties of water and steam[4]. By keeping your steam lines just within the GCC lines, the pinch condition is satisfied while attaining the maximum flow rate ie. the maximum heat transfer. For this system, water at the inlet is 25° (30° shifted) and heated to 120° (125° shifted). Heating the water to 120° will evaporate the water to steam and thus all the stages of heating water need to be considered. The calculations for this require the Equation 7, Equation 8, Equation 9 for the sensible, latent and superheating stages.

$$M_w = \frac{H_{start} - H_{wsat}}{C_{pw}(T_{sat} - T_{start})} \tag{7}$$

$$M_w = \frac{H_{Wsat} - H_{ssat}}{\Delta H_{vap}} \tag{8}$$

$$M_w = \frac{H_{ssat}}{C_{ps}(T_{final} - T_{start})} \tag{9}$$

These 3 equations give 4 unknowns; H_{start}, H_{wsat}, H_{ssat} and MW. With one degree of freedom, If H_{wsat} is guessed, the above 3 equations can be combined into:

$$H_{ssat} = \frac{C_{ps} H_{wsat}(Tfinal - Tsat)}{Hvap + Cps(Tfinal - Tsat)} \tag{10}$$

$$H_{start} = H_{wsat}(1 + C_{pw}\frac{T_{sat} - T_{start}}{\Delta H_{vap}}) - H_{wsat}(C_{pw}C_{ps}\frac{T_{sat} - T_{start}}{\Delta H_{vap}})(\frac{T_{final} - T_{sat}}{\Delta H_{vap} + C_{ps}(T_{final} - T_{sat})}) \tag{11}$$

$$Mw = (\frac{H_{wsat}}{\Delta H_{vap}})(1 - C_{ps}\frac{T_{final} - T_{sat}}{\Delta H_{vap} + C_{ps}(T_{final} - T_{sat})}) \tag{12}$$

By guessing the H_{wsat} value just outside the GCC of the same temperature, a corresponding flow rate is produced. This analysis requires an iterative reduction of H_{wsat} until it is just touching the GCC. Once that is satisfied, look to make sure the "corner points" (GCC line segment endpoints) are all less than the corresponding GCC enthalpies at the same temperature. If any of these criteria are not met, Mw will iterate through at a smaller Hwsat changing the shape of the graph and the value of Mw. Finally, if H_{ssat} is less then or equal to 0, no solution can be found. This process can be verified using computational tools like Microsoft Excel or Python below. Using the GCC data and the initial conditions, the full scale plant can be calculated to have a maximum flow rate of 10.67 Kg/s. The is the maximum heat taken out of the system is equal to H_{start}, 27851 kW. All of this computation was done by a Python code in Appendix vii

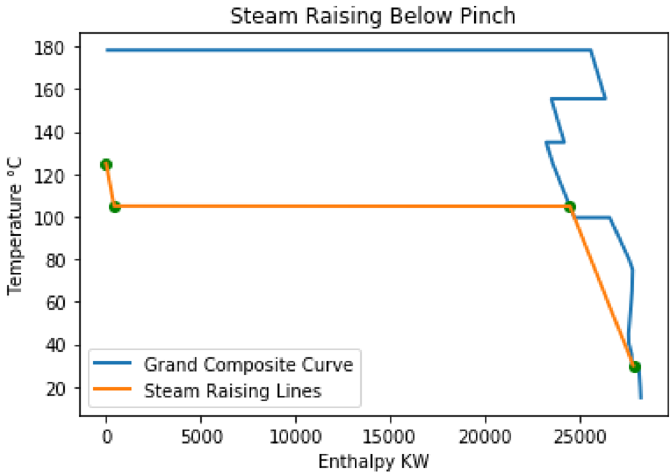


Figure 7: *From right to left:* $(H_{start}, T_{start}), (H_{Wsat}, T_{sat}), (H_{ssat}, T_{sat}), (0, T_{Final})$

i. Integrating Steam Raising into the Heat Exchange Network

A new heat exchanger network was designed for the purpose of utilizing steam raising to get as much heat as possible out of the heat exchange network. This network will have multiple pinches due to the fact that no energy will be left in the system at the original pinch, and no energy will be available at two other points due to the steam raising process taking all available energy. These points are at shifted temperatures of 105°C and 45°C, in addition to the existing pinch at 254.3°C. These pinches were calculated through the python pinch analysis code in Appendix xvii. Due to the existence of three pinches, the network must be changed a great deal to adjust for steam raising and is pictured in Appendix iii. Realistically, the ideal Q_c and Q_h calculated from the code in Appendix xvii cannot be reached through the system. The actual Q_c the network can support in the full scale plant is is 660.39 kW, versus the calculated 441.77 kW. In the same vein, the Q_h that the network can actually support is 28,780 kW, versus the calculated 28,560 kW. Due to this, about 220 kW needs to be added to the system that will be removed by coolers later. However, steam raising uses 27,585 kW that would otherwise be unused, so this small change to Q_c and Q_h is well worth it because so much extra energy that would be wasted by the system is now being used. This steam could then be used to power a turbine if the plant wanted to use this energy to generate electrical energy.

5. Carnot Cycle

i. Carnot Cycle Construction

Another method to utilize the waste heat is to convert to work using the Carnot Engine. The Carnot engine is a basic thermodynamic model of an engine that displays the change in pressure and volume between two specific isotherms. A piston cylinder system within the engine will expand and contract based on the limits that are created by the isotherms. The net result of the movement of the pistons produce work that can be used to move whatever is required by the system.

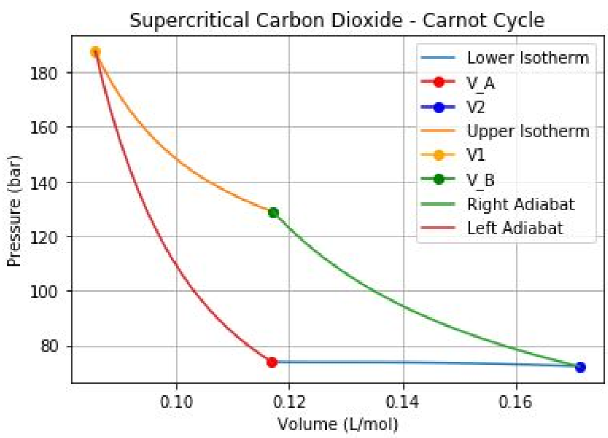


Figure 8: *The Carnot Cycle of Supercritical Carbon Dioxide*

To develop this system, the amount of work the engine produces depends on the fluid that is used to move the piston. For this engine, carbon dioxide near its critical conditions will be used. By staying near the super critical conditions, pressure becomes a weak function of volume and thus the isotherms become flatter. The flatter the isotherms, the greater the difference between them, which increases the potential work that can be extracted from the engine. The critical conditions dictate how some of the parameters of this analysis were established to allow for other parameters of the engine to be changed effecting the outcome of work.

In order to build the Carnot cycle, the volume range of any curve needs to be established. To stay around the critical conditions, the critical volume of the fluid can be calculated by Equation 13 [8], after being given critical pressure and temperature. By initially being given the critical pressure, P_c, and the critical temperature, T_c.

$$V_c = \frac{3RT_c}{8P_c} \tag{13}$$

Once the critical volume is established, the range of the volume can be determined by centering the range over the critical volume. The length of the range is determined by the compression ratio of the fluid. The compression ratio is the relation between the maximum and minimum volumes of the fluid represented by V_1 and V_2, respectively. The compression ratio was set at 2 for the purposes of this project to allow for there to be a reasonable amount of movement of the fluid when the heat is added to the engine.

The next two parameters established were the isotherms that are used to create the heat difference that moves the piston. The lower isotherm was set as the critical temperature of carbon dioxide to ensure that the fluid can remain, at minimum, near the critical conditions. The upper isotherm was determined by selecting a unit that was attached to a hot stream and had a temperature that was reasonably higher than T_c. A cooler on stream 7 had an outgoing temperature of 353.13 K (80°C) which produced a sizeable amount of work. ix

Once the volume range and the temperatures were established, the curves of the isotherms can be created using Equation 14 [8]:

$$P_c = \frac{RT_c}{V_c - b} - \frac{a}{V_c^2} \tag{14}$$

Where V_c is the critical volume, T is the temp of the isotherm and constants a and b are constants that apply to the Van der Waals equation and can be calculated from Equation 15 [8] and Equation 16 [8]:

$$a = \frac{27R^2T_c^2}{64P_c} \tag{15}$$

$$b = \frac{RT_c}{8P_c} \tag{16}$$

After the isotherms are created, the adiabatic curves on the left and right sides of the Carnot cycle need to be established through Equation 17 [8]:

$$V_B - b = (V_2 - b)\frac{T_c}{T_h}^{\frac{C_v}{R}} \tag{17}$$

Where b is the Van der Waal's constant calculated prior, T_c is the the lower isotherm, R is the gas constant, V_b is the volume associated with the upper right corner of the Carnot cycle, and C_v is the heat capacity of the fluid at different temperatures. The range of volumes is then found by iterating T_h in Equation 17 from T_c to T_h. The range of volumes can then be used to calculate a range of pressures using Equation 14 and that will connect the two isotherms by the adiabat.

The left adiabat can be found in a similar way to the right adiabatic curve except that Va is used instead of V_b, V_1 is used instead of V_2, and the T_c/Th is inverted to T_h/T_c and T_c is varied instead of T_h.

C_v can be calculated by Equation 18 [8]:

$$C_v = \frac{R(A + BT + CT^2 + DT^{-2})}{\gamma} \tag{18}$$

The A, B, C, D and gamma constants that are specific to the fluid used in your system. The values to each of those constants can be found in Table 2 [10]:

Table 2: *A,B, C, D and gamma constants for CO_2*

	CO_2
A	5.457
B	1.045*10^{-3}
C	0
D	-1.157*10^{5}
γ	1.29

The C_v value that was used was an average of all the C_v's found iterating temperature from T_c to T_h. The range of the C_v temperatures is not large and therefore taking an average is an accurate way to represent the C_v's for this temperature range.

After all four curves have been established, the Carnot cycle is complete and the work associated by expanding and contracting the piston in the engine can, numerically, be found. The work associated with one cycle of the Carnot engine is the area of the shape formed by connecting the curves. ix

ii. Numerical and Analytical Analysis

To run this engine however, there needs to be an amount of heat that is introduced to create isothermal expansion, and can be taken from the initial network design for our plant that is Figure i. The temperature of the cooler utilized needs to have a T_h value that is large enough to produce an optimal amount of work, but also cannot be too large as the work output becomes negative at some point.x With both of these considerations in mind, cooler 5 from our network fits this criteria for not being excessively large nor being to small to create very little work.

These amounts of work and heat that have been calculated from the network and plot, can also be done through equations, analytically. In Equation 19 [8] and Equation 20 [8], work and Q_H can be found based on some of the chosen parameters of our calculations.

$$Q_H = \int_{V_1}^{V_B} \frac{RT_h}{V-b} dv = RT_h ln \frac{V_B - b}{V_1 - b} \tag{19}$$

$$W = Q_h(1 + \frac{T_c}{T_H}) \tag{20}$$

The implementation of using these equations allow for a good comparison of how much work that can be taken out of the system if the numerical analysis of the system and graph were correct.

Lastly, the efficiency of a Carnot engine is a simple calculation that compares the utilized isotherms with which the system runs. The equation for finding the efficiency is seen as Equation 21 [8]:

$$\eta = 1 - \frac{T_c}{T_h} \tag{21}$$

iii. Carnot Engine Specifications

To determine the rate at which the system has to complete a cycle, an accurate relationship between the work and heat needs to be found. The heat is being inputted to the system at the rate of J/sec and the work that is extracted occurs in J/cycle*mol. The frequency of how fast the piston moves is then found through Equation 22.

$$Hz = \frac{\eta * Q_h}{W * mol} \tag{22}$$

The amount of moles is arbitrarily assigned to allow for an appropriate amount of cycles/sec to occur. In this regard, there cannot be too few moles assigned to the piston since that will cause for a preposterous amount of cycles needed to intake all of the heat. Oppositely, too many moles will call for too large of a system that will be unnecessarily expensive and large that the pistons may reach a point of underutilization of what could be a more efficient process.

The size of the engine can be calculated therefore based on the number of moles selected by the system and the maximum volume needed to run the system V2. The calculations for volume, pressure and work have been on a "per mole" bases thus far and therefore the volume of the engine needed is found through Equation 23.

$$V = mol * V_2 \tag{23}$$

The size of the engine is therefore based on the size of a one piston-cylinder system in an ideal setting. For the purposes of applying it to being able to handle the capacity of an entire plant however, there needs to be some consideration on how to better balance the capacity of absorbing heat and converting it to work.

To improve the reality of getting this engine to appropriately handle the impact of the incoming heat, dispersing the work amongst multiple cylinders better improves the engine and its ability to work. Instead of just working with one piston, the engine could be rearranged to have 6 cylinders in flat arrangement connected to a crankshaft that in turn does the work to run a process. By doing this, there is less of an impact on any one cylinder to have to produce all of the work and improve the longevity of the internal pieces to that engine. On top of this, the amount of supercritical fluid gets divided amongst the cylinders making it easier to maintain, and the volumes of each cylinder still surround the critical volumes to keep it in that state.

Table 3 displays the impact on one cylinder in both a one cylinder engine and 6 cylinder engine using 100 moles of CO_2, the work found numerically from our system and the Q_h that is being inputted into the Carnot engine from our heat exchange network.

Table 3: *Comparisons of Numerical and Analytical values*

	1 Cylinder Engine	6 Cylinder Engine
Frequency (Hz)	23.51	3.92
Size of Cylinder (L)	6.852	1.142

The Carnot cycle that applies to our network can finally be integrated into the system as seen in Appendix ii to use some of the heat that is available.

By adding this engine cycle to the system to the system to replace a cooler, not only is the engine producing work that can be used to run another process but there is a cost savings that is associated with no longer needing to run a cooler to reduce the temperature of the hot stream. In this case, assuming that coal is used to power the electricity needed, the work produced by the engine saves $45,631.50 per year and no longer needing to run the cooler saves $328,537.64 per year for a total savings of $374,169.14 per year.

iv. Supercritical Working Fluid Comparison

The engine that was calculated for in the previous tables utilized supercritical CO_2 for the working fluid within the piston cylinder system. The calculations therefore, were based around the characteristics of the CO_2 such as the critical volume determining the range of the volumes the piston would move or the critical temperature acting as the lower isotherm of the PV diagram.

By varying the fluid within the engine, different parameters may be needed to better absorb the heat from the network and convert it to work, while still remaining around the supercritical conditions that benefit work production. Common supercritical fluids that can be used in this situation are methane, ethane and water. By comparing their properties to those of CO_2 and how these fluids could possibly apply to our network, an optimal fluid can be found that suits the network better than CO_2 would.

Each of the three other fluids have critical conditions that might lead to a better solution for this network. Methane has an extremely low critical temperature (-80°C) and thus using an isotherm of room temperature would create the large difference between the two curves producing work. Ethane has a critical temperature that is very close to CO_2 (305.3K for Ethane, 304.1 for CO_2) but has a critical pressure that is almost half of that of CO_2. The resulting difference in critical volume based on critical temperature and pressure may be able to produce a more efficient Carnot engine. Lastly, water has a high critical temperature which can be applied to few portions of the available network, but exploring greater critical temperatures could also lead to a more efficient Carnot engine.

Table 4 [10] shows a comparison of the critical parameters of the four common supercritical fluids for quick comparison of differences.

Table 4: *Critical Parameters for Selected Fluids*

	CO_2	Methane	Ethane	Water
T_c (K)	304.1	190.6	305.3	647.1
P_c (bar)	73.80	45.99	48.72	220.55
Vc (L/mol)	0.128	0.129	0.195	0.0915

T_c and P_c are from accepted literature and Vc is calculated via Equation 13.

The other individual parameter that needs to be established for each of the fluids is their heat capacity coefficients. The coefficients can be used to calculate a heat capacity that applies to finding a volume range. The coefficients for CO_2 have been listed prior in Table 2, and the coefficients for the other supercritical fluids is listed in Table 5 [10].

Table 5: *Coefficients for Selected Fluids*

	Methane	Ethane	Water
A	1.702	1.131	3.470
B	9.081*10 -3	19.225*10 -3	1.450*10 -3
C	-2.164*10 -6	-5.561*10 -6	0
D	0	0	0.121*10 5
γ	1.31	1.19	1.33

For the purposes of comparison, the critical parameters and the heat capacity constants are the only parameters that will change about the fluids. In order to have a logical argument for discussion on the best critical fluid for our scenario it is fair to keep consistency for outside conditions, beyond that of the fluid, to best see how the fluids compare. The constants that will remain the same between the fluids are the way each of the functions and results are calculated, the compression ratio, the critical volume being exactly between V1 and V2, and the number of moles that the engine will utilize to complete the Carnot cycle.

The first comparison of the fluids that can be looked at is how well each fluid performs under the same difference in temperature difference and same Q_h inlet. By comparing the critical parameters to each other, inferences can be made on which fluids may work best for the network given.

Each of the critical temperatures of the fluids will be used as T_c and T_h will be determined using the same temperature difference as calculated earlier when a CO_2 engine was integrated into the network which is 49.05 K. The reason that the temperature difference is remaining same and not the temperatures themselves is that by using different temperatures other than the critical temperatures for each fluid, the fluid is not going to be around supercritical conditions and therefore the information gathered from that approximation would not make conclusions about the fluid as a supercritical fluid. The same Q_h is being applied to each of the fluids to accurately compare the frequency of cycles needed to produce the calculated work.

The results of this comparison is listed in Table 6. T_h for each fluid was 49.05K higher than T_c, Q_h is 1,521,768 Watts, the number of moles in each system is 40 moles of each fluid. CO_2 results, although previously listed, were added to Table 4 for comparative reasons.

Table 6: *Calculated Values for Selected Fluids*

	Carbon Dioxide	Methane	Ethane	Water
T_c (K)	304.1	190.6	305.3	647.1
T_h (K)	353.1	239.65	354.35	696.15
Efficiency	13.9%	20.5%	13.8%	7.05%
Q_h (J/s)	1,521,768	1,521,768	1,521,768	1,521,768
Work (J/cycle*mol)	224.749	194.590	97.464	347.872
Q_c (J/s)	1,310,405	1,210,302	1,311,121	1,414,546
Hz (1/s)	23.51	40.02	54.03	7.71
Engine Size (L)	6.852	6.891	10.42	4.88

Methane as compared to carbon dioxide at these conditions proves to simply not match up in quality of output in most categories of comparison. Despite the methane based engine being more efficient, the work output with the same temperature difference as carbon dioxide is considerably smaller due to the fact that lower overall temps causes smaller calculated area outputs. This reduction in work, causes a higher required frequency which creates more wear on the engine over time.

Ethane, despite having similar isotherms to carbon dioxide, also produces results that are not as preferable as an engine based on carbon dioxide. The issue mainly stems from the critical volume of ethane being significantly greater than that of carbon dioxide's. The larger the critical volume, the greater the size of the engine, however the isotherms stretch the Carnot graph and ultimately produce less work. This causes required frequency to increase which is less desirable than the original design.

Water, at its high temperatures, produces results that rival the results of carbon dioxide in numerous ways. Most noticeably, the work that is produced by this cycle is greater than carbon dioxide's because the upper isotherm extends to such high temperatures producing a greater area of the Carnot plot. The increased amount of work, decreases the required frequency and the lower critical volume produces a smaller engine size overall. The results of the supercritical water engine make it possible that if it can be successfully integrated into the heat exchange network, it could be more profitable in the long run than the given carbon dioxide engine.

v. Supercritical Working Fluid Integrations

After reviewing the comparative results based on the already integrated supercritical carbon dioxide engine, the engines based on the other supercritical fluids can also be attempted to be integrated into the network. If they can be appropriately integrated to the network, the outputs need to be considered to see if it is profitable to use that type of engine over the given carbon dioxide.

Any attempts to integrate an engine type into the network will first be based on if it is even possible to work it into the system based on the critical temperatures, and then placed on the diagram to find the optimal location for it.

The constants that are kept in this comparison include the calculations, the compression ratio, the critical volume of the fluid being exactly between V1 and V2, and the number of moles of the system which will be set to 40 moles. The results of this comparison is listed in Table 7. CO_2 results, although previously listed, were added to Table 4 for comparative reasons. (App.ix through xii)

Table 7: *Integrated Calculated Values for Selected Fluids*

	Carbon Dioxide	Methane	Ethane	Water
T_c (K)	304.1	190.6	305.3	647.1
T_h (K)	353.1	298.15	353.15	683.15
Effeciency	13.9%	36.1%	13.5%	5.28%
Q_h (J/s)	1,521,768	81,000	1,521,768	976,668
Work (J/cycle*mol)	224.749	-175.060	103.317	274.777
Q_c (J/s)	1,310,405	51,781.3	1,315,576	925,128.8
Hz (1/s)	23.51	-4.17	49.89	4.69
Engine Size (L)	6.852	6.891	10.42	4.88

For methane, the trouble begins with integrating it into our network by the fact that the critical temperature of methane is comparatively low (190.6 K, -82.55°C) and the lowest possible temperature on the network is 20 ° C, located on Stream 2 in Figure i. For consistency between engines, the lower isotherm will be the critical temperature for each of the fluids and the upper isotherm will be the temperature of the stream in the network. The result of having such a dramatic difference in temperature becomes problematic when trying to run a Carnot engine.

Disregarding the energy required to maintain a lower isotherm that is so low in temperature, the calculation method of finding work is limited by the temperature difference. In this instance, the temperature difference is large enough that the resulting work is negative (App.v). Needless to say, a Carnot engine that uses methane as a working fluid is not appropriate, considering the temperatures of the network.

For ethane, due to its similar critical temperature to carbon dioxide, can reasonably be integrated into the system at the same place as the integrated carbon dioxide Carnot engine. Similar to the results found when the fluids were compared at similar conditions, the results of using ethane at this section of the network gives inferior results to that of the carbon dioxide results (App.vi). The greater required size of the engine, higher required frequency, and lower work output prove that carbon dioxide working fluid is more appropriate than the ethane working fluid for this network.

While methane and ethane did not prove to be properly suited for this network, which was consistent with the comparison of the fluids, water showed promising results when attempted to be integrated into the system.

If the original network is redesigned and the Carnot engine with water as the working fluid is integrated at stream two, the calculations can be done to remove heat up until the temperature of the exiting fluid is 410°C. The results from that integration produce the results shown in Table 7 which prove to be better than the carbon dioxide based engine. The work output is increased, the size of the engine decreases, and the frequency decreases. Water is also abundantly accessible to apply to this system so from a cost standpoint of installing one of these engines, the benefits prove to beat the benefits of the carbon dioxide based engine.

When integrating the engine utilizing water as its working fluid, the possible energy production is higher than that of any other fluids tested (App.iv). However, this cannot be integrated into the existing heat exchange network, since the water engine would only function above the pinch, and there is no energy available for this engine to use above the pinch.

Although none of the other explored fluids could be better applied than the given carbon dioxide based engine, the comparison of them proved to show interesting insights about when a supercritical fluid is appropriate for a system. In terms of cost reduction, integrating engines should optimally be done to replace current machinery to create productivity out of waste heat. It is also important to understand that the critical temperature will determine where the engine can be possibly integrated, lower critical temperatures apply to colder systems and higher critical temperatures apply to hotter systems. Lastly, the size of the engine, and the costs associated with it, are determined by the critical volume of the fluid where lower critical volumes reduce size and therefore there is less fluid needed to make the engine work and less capital costs for material and installation.

6. Eigenvector Centrality

One method used to look at the interconnectedness of heat exchanger networks is eigenvector centrality. Eigenvector centrality is a numerical measure from 0 to 1 of how important a node (or in our case, stream) is to the network as a whole. Eigenvector centrality is more specific than other types of centrality like degree centrality since eigenvector centrality considers a node more useful if it is connected to other well connected nodes. Additionally, the edges (which in our case are heat exchanges) can be weighted, meaning that centrality can be calcualted based on how much heat each stream is transferring.

i. Original Network

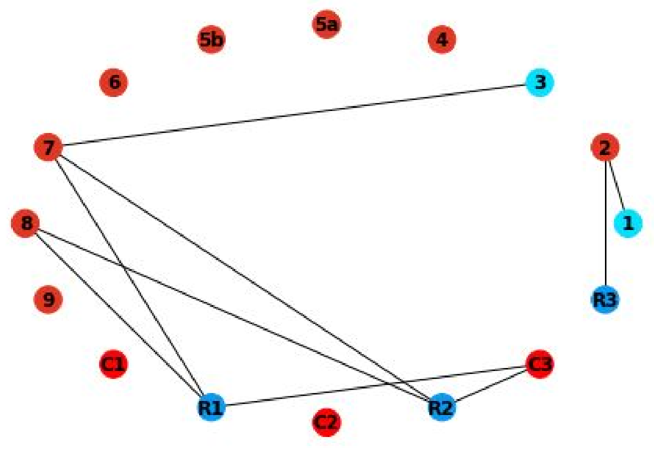


Figure 9: *The original network designed for the plant was the network above. C1-3 represent condensers, and R1-3 represent reboilers.*

The eigenvector centrality values are as follows:

Table 8: *Original Network*

Stream	Centrality
1	$9.05*10^{-6}$
2	$1.28*10^{-5}$
3	0.180
4	$2.03*10^{-16}$
5a	$2.03*10^{-16}$
5b	$2.03*10^{-16}$
6	$2.03*10^{-16}$
7	0.454
8	0.383
9	$2.03*10^{-16}$

Stream	Centrality
Condenser 1	$2.03*10^{-16}$
Condenser 2	$2.03*10^{-16}$
Condenser 3	0.383
Reboiler 1	0.484
Reboiler 2	0.484
Reboiler 3	$9.05*10^{-6}$

From these values, reboilers 1 and 2 are the most connected, followed by stream 8 and condenser 3, and then stream 3. Streams 1 and 2, and reboiler 3 are connected in their own less important network, so they have lower centralities, with stream 2 being the most important of the three. All other streams are unconnected, and therefore very unimportant in the network, so they are given extremely low centrality values. These values are incredibly small, but still non-zero since they are still part of the network as a whole, even though they have no connections. From this, reboilers 1 and 2 are most important for our system as a whole when it comes to their general connectedness, meaning that if they fail in some way, this hurts the rest of the network the most.

The weighted eigenvector centrality based on the amount of heat transfer in each connection is as follows:

Table 9: *Original Network Weighted*

Stream	Centrality
1	0.707
2	0.707
3	$8.61*10^{-6}$
4	$7.61*10^{-19}$
5a	$7.61*10^{-19}$
5b	$7.61*10^{-19}$
6	$7.61*10^{-19}$
7	$4.84*10^{-5}$
8	$2.55*10^{-6}$
9	$7.61*10^{-19}$

Stream	Centrality
Condenser 1	$7.61*10^{-19}$
Condenser 2	$7.61*10^{-19}$
Condenser 3	$1.05*10^{-4}$
Reboiler 1	$2.71*10^{-5}$
Reboiler 2	$1.07*10^{-4}$
Reboiler 3	$1.20*10^{-4}$

From these, we see that streams 1 and 2 are the most important to our overall network, since they transfer so much heat (986.8 kW in the pilot plant, 9868 kW in the full scale) between one another. Reboiler 3 is given a high centrality value since it's interconnected with these two streams. Reboilers 3 and 2, and condenser 3, followed by streams 7, 3, then 8 all have middling centrality values since they are transferring a fair amount of heat, and are interconnected with each other. Finally, all other streams are not connected, giving very low centrality values. From this, we see that if there are problems with streams 1 and 2, this becomes quickly problematic for the network because of how much heat they are transferring. However, these failing would really only affect each other, since they are only connected to each other and reboiler 3. The code used to calculate these values is in Appendix xvi.

Note that everything within this section also applies to the alternate network with the Carnot engine integrates since the Carnot engine will just be replacing an existing cooler.

ii. Steam Raising Network

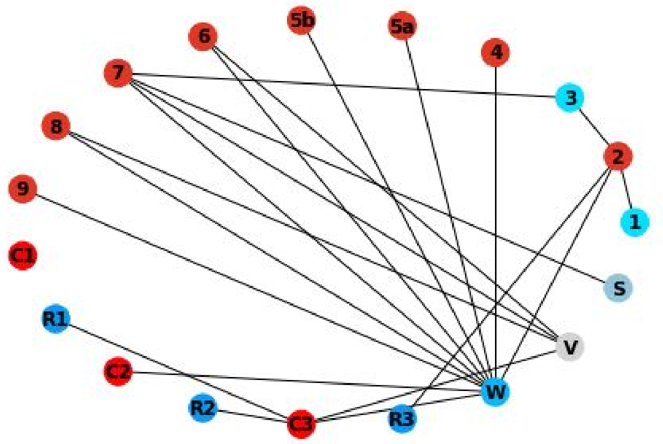


Figure 10: *The redesigned network for steam raising is pictured above. C1-3 are condensers, R1-3 are reboilers, W is the water stream before it is vaporized, S is the steam after being vaporized, and V is the water as it is being vaporized.*

The eigenvector centrality values for this network are below.

Table 10: *Steam Raised Network*

Stream	Centrality
1	0.0670
2	0.243
3	0.153
4	0.164
5a	0.164
5b	0.164
6	0.248
7	0.314
8	0.248
9	0.16

Stream	Centrality
Condenser 1	$2.91*10^{-19}$
Condenser 2	0.164
Condenser 3	0.292
Reboiler 1	0.0804
Reboiler 2	0.0804
Reboiler 3	0.0670
Water	0.596
Steam	0.0864
Vaporization	0.303

From this, we see that water is the most connected stream by far. This is largely due to its high CP value. The vaporization stream also has a high number of connections, due to its high CP value and very high energy requirements (2400 kW in the pilot plant, and 24000 kW in the full scale). Then, condenser 3 is connected to both of these important streams (and reboiler 2), so it is next most important stream. The rest of the streams follow this sort of pattern, with the only notable stream being condenser 1, which has no connections within the network, and as such has a super low centrality value. This shows us that if the water stream fails, a huge number of streams would be affected by this.

The weighted eigenvector centrality based on the amount of heat transfer in each connection is shown in the table below.

Table 11: *Steam Raised Network Weighted*

Stream	Centrality
1	0.0257
2	0.0254
3	0.00126
4	$4.95*10^{-7}$
5a	$4.06*10^{-5}$
5b	$3.67*10^{-6}$
6	0.00713
7	0.0607
8	0.00508
9	$1.02*10^{-5}$

Stream	Centrality
Condenser 1	$2.66*10^{-14}$
Condenser 2	$2.99*10^{-4}$
Condenser 3	0.683
Reboiler 1	0.0313
Reboiler 2	0.0944
Reboiler 3	$4.35*10^{-6}$
Water	0.00311
Steam	0.721
Vaporization	0.00125

From this table, it can be seen that the vaporization stream is most important, with condenser 3 relatively close behind it, and stream 7 behind that. This is due to the vast amounts of heat (for vaporization: 2400 kW in the pilot plant, and 24000 kW in the full scale, and for condenser 3: 2552.7 kW in the pilot plant, and 25527 kW in the full scale) that condenser 3 and the vaporization stream are exchanging. Stream 7 is considered important mostly because of the streams it exchanges heat with. Stream 7 exchanges heat with the vaporization stream, and two streams which connect to stream 2, which has our next biggest heat exchange behind condenser 3 and the vaporization stream. What all of this means is that stream 7, the vaporization stream, and condenser 3 are all important to this network, due to how much heat they are exchanging, and their connectivity, so if any of these fail in some way, the network will be heavily affected.

iii. Heat Exchanger Energy/Area Analysis

Two ways to measure each heat exchanger's significance in the overall network are the amount of energy transferred, and the area of each heat exchanger. It is important to keep each of these in mind, because if important heat exchangers fail for any reason, this could cause the whole system to stop functioning properly since the processes will not be at the desired temperatures.This section analyzes which heat exchanger should be monitored the most closely due to this fact.

iii.1 Original Network

In the original network (Appendix i), the heat exchanger energies and areas needed are in Table 12. Numbers in Table 12 are for the full scale plant.

Table 12: *Powers and Areas for Original Flowsheet*

Heat Exchanger	Power (kW)	Area (m2)
1	4652.252	1624.3
2	5215.8	1819.3
3	431	20.2
Reboiler 1 and Stream 7	665.04	-
Reboiler 2 and Stream 7	746.54	-
Reboiler 1 and Stream 8	24.0	-
Reboiler 2 and Stream 8	104.753	-
Reboiler 3 and Stream 2	1.6726	-
Reboiler 1 and Condenser 3	261	-
Reboiler 2 and Condenser 3	2013.7	-

Areas were calculated using Equations 24 and 25. Note that areas of heat exchange cannot be easily calculated for connections to reboilers or condensers since they do not change temperature. LMTD of the reboiler connections is shown below in a Equation 26 [7] [2]. This equation could be modified for condenser connections, and condenser-reboiler connections to get a heat transfer area. Additionally, note that T^{sat} is needed to calculate this area, and since a number of the streams used for this will be at vapor liquid equilibrium in mixtures of multiple substances, these would need to be categorized before calculating these areas. In addition, the spatial requirements of the facility would need to be taken into account since all of the heat transfer occurring at the reboilers and condensers needs to happen in relatively the same location, complicating things further, thus why calculations for these areas were forgone in this analysis.

$$Q = UA * LMTD \tag{24}$$

$$LMTD = \frac{\Delta T_A - \Delta T_B}{ln(\Delta T_A / \Delta T_B)} \tag{25}$$

$$LMTD = \frac{(T^{sat} - T_{2o}) - (T^{sat} - T_{2i})}{ln\frac{T^{sat} - T_{2o}}{T^{sat} - T_{2i}}} \tag{26}$$

Through this method, we see that the most heat power is transferred through exchangers 1 and 2. This means that these failing would likely have the largest effect on other streams. This is also supported by the weighted edge eigenvector centrality values, since streams 1 and 2, the streams exchanging heat in exchangers 1 and 2, have the highest weighted centrality values. The connection between reboiler 2 and condenser 3 also transfers a great deal of heat, so the same is true if this connection fails. The areas for heat exchangers 1 and 2 will be very large, so this means that they will be harder to maintain since they will take up more overall space in the plant, and there will be a greater area for them to possibly fail. Because these two exchangers also transfer the most heat, precautionary measures should be taken to make these exchangers less likely to fail, or easier to repair, since they are most likely to have issues, and contribute the most to the overall network.

iii.2 Steam Raising Network

In the network where steam raising is integrated into the process (Appendix iii), the heat exchanger energies and areas needed are in Table 13. Numbers in Table 13 are for the full scale plant. Areas for connections to condensers, reboilers, and the vaporization stream are not calculated for the same reasons as above.

Table 13: *Powers and Areas for Steam Raised Flowsheet*

Heat Exchanger	Power (kW)	Area (m^2)
1	4652.252	1624.3
2	109.78	38.64720065
3	2.763	0.9521834871
4	17.582	3.860317573
5	19.012	4.174289484
6	8.47	1.868626706
7	13.439	3.128161073
8	430.474	54.9356993
9	425.1344	22.47118525
10	547.14	48.41905523
11	226	24.83970954
12	345.09	25.55337162
13	12.0	2.453482018
14	70.62	8.274748874
15	38.182	6.484044467
16	3642.3	1274.172275
17	1463.74	514.84686

Heat Exchanger Area (m^2)	Power (kW)
Water Stream and Condenser 2 (Connection 1)	485.793
Water Stream and Condenser 2 (Connection 2)	1185.4
Water Stream and Condenser 3	28.83
Vap. Stream and Stream 6	228.91
Vap. Stream and Stream 7	1960.251
Vap. Stream and Stream 8	163.955
Vap. Stream and Condenser 3	21683.17

Within this network, There are several heat exchangers that are very important to the overall network. These are exchangers 1, 16, and 17, the second connection between water and condenser 2, V (the vaporization stream) connected with stream 7, and V connected with condenser 3, since these all transfer more than 1 MW of power. The connection between V and condenser 3 is especially important since it is transferring 21.683 MW, so if this fails, the impact on the system would be disastrous. This observation is supported by the fact that condenser 3 has the second highest weighted centrality value, meaning that if something goes wrong here, the entire system becomes affected, so extra precaution needs to be taken to ensure that this exchange does not fail. Due to the interconnectedness of this network (Section ii , Figure 10), any of these exchangers encountering problems could be devastating for the system as a whole. The exchangers with the largest heat transfer areas are exchangers 1, 16, and 17. Coincidentally, these exchangers are also some of the most important to the system, so it would be prudent to design the layout of these heat exchangers in such a way where they are not likely to get damaged, or can be fixed easily if they fail.

7. Conclusion

Upon full integration of the system from the given diagram, we were able to recover the maximum amount of heat given the network. After analyzing the system using pinch analysis, the Q_{hmin}, Q_{cmin} and T_{pinch} values of the pilot plant analyzed are 2855.987 KW, 2824.387 KW and 254.3°C when T_{min} =10°C.

An analysis of finding the lowest cost for operating the plant over time was conducted to find the appropriate measures for developing this plant. After 1 year of running the plant, the lowest cost to develop the system is $41,870,000 if the T_{min} is set at 20.384°C. If the system were to be run for a longer period of time, however (at least 634 days), the best T_{min} for the system would become 10°C, since this would produce the lowest possible overall cost.

There are great incentives to implement this heat exchange network from both a business perspective and from an environmental perspective. Based on given information regarding energy facts about coal, the integrated system saves over $3,000,000 per year and reduces over 94,000,000 kilograms of carbon dioxide emissions to the atmosphere per year.

When comparing other common supercritical fluids (methane, ethane and water) to carbon dioxide as working fluids, the results are mainly dependent on the critical conditions of the fluid. Methane and ethane showed to be less effective at producing work than carbon dioxide, but water had promising results before the fluids were attempted to be integrated. After integration, methane could not be appropriately integrated and ethane did not work as well as carbon dioxide in the network. Water could be integrated into the system and produced more work than carbon dioxide, but the arrangement of the network for that to be successful was not feasible overall and thus supercritical carbon dioxide was the best working fluid for our Carnot engine.

Using eigenvector centrality, the relationships between streams can be analyzed, and the most interconnected streams can be identified. This type of analysis can be used to find the most important parts of a system. Additionally, heat exchanger areas and energies were calculated to find the most important exchangers and the amount of area they take up, since exchangers with larger areas are often more vulnerable to leaks or other problems.

8. Bibliography

References

[1] "Mixed Xylenes." *IHS Markit,* Aug. 2018, ihsmarkit.com/products/xylenes-chemical-economics-handbook.html.

[2] "Process Design of Heat Exchanger: Types of Heat Exchangers : Types of Heat Exchanger, Process Design of Shell and Tube Heat Exchanger, Condenser and Reboiler."

[3] Hagberg, Aric A., Schult, Daniel A., and Swart, Pieter J., "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

[4] He, Yuan, et al. "An Algorithm for Optimal Waste Heat Recovery from Chemical Processes." *Computers Chemical Engineering*, vol. 73, 2015, pp. 17–22., doi:10.1016/j.compchemeng.2014.11.003.

[5] Hunter, John D.,. Matplotlib: A 2D Graphics Environment, Computing in Science Engineering, 9, 90-95 (2007),DOI:10.1109/MCSE.2007.55

[6] Kemp, Ian C. *Pinch Analysis and Process Integration: a User Guide on Process Integration for the Efficient Use of Energy.* 2nd ed., vol. 1, Elsevier Science, 2011.

[7] Laskowski, Rafal, et al. "Determining the Optimum Inner Diameter of Condenser Tubes Based on Thermodynamic Objective Functions and an Economic Analysis." Poland, Warsaw.

[8] Latiz, Madeleine R, et al. "Critical CO2 Carnot Cycle for Waste Heat Utilization." New York, Rochester.

[9] Olivia Pusineli. "ExxonMobil Chemical to Buy Major Plant in Singapore." *Bizjournals.com*, Houston Business Journal, 12 May 2017, www.bizjournals.com/houston/news/2017/05/12/exxonmobil-chemical-to-buy-major-plant-in.html.

[10] Smith, J. M., et al.*Introduction to Chemical Engineering Thermodynamics.* 8th ed., McGraw-Hill Education, 2018.

[11] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).

[12] Woods, Laura. "Global Xylene Market Analysis, Growth, Trends Forecast 2018-2023 - ResearchAndMarkets.com."emphBusiness Wire, Business Wire, 28 May 2018, www.businesswire.com/news/home/20180528005115/en/Global-Xylene-Market-Analysis-Growth-Trends-Forecast.

9. Appendix

i. Original Network Flowsheet

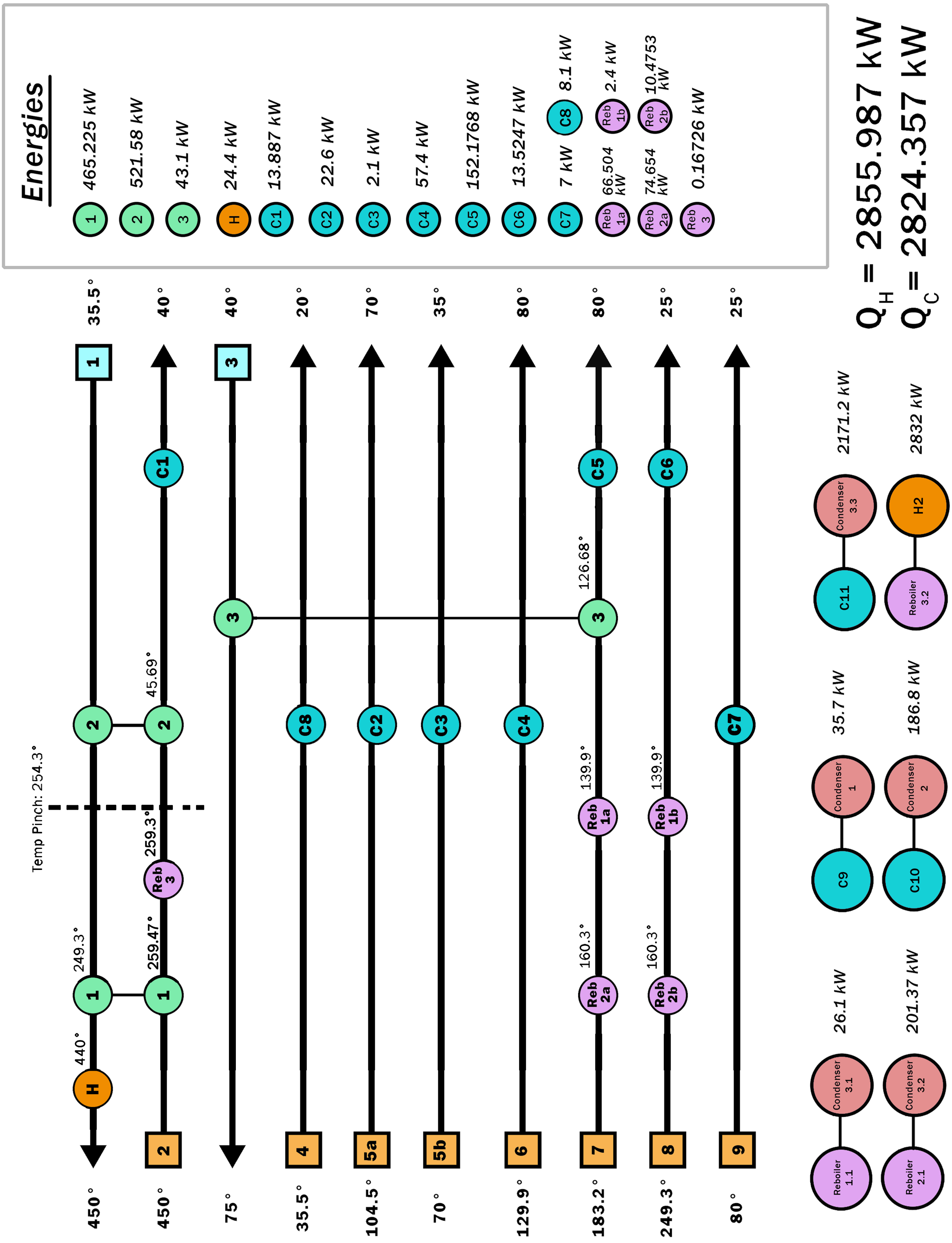


Figure 11: *Design Process Flowsheet*

ii. Network Flowsheet with Carnot Engine Integrate

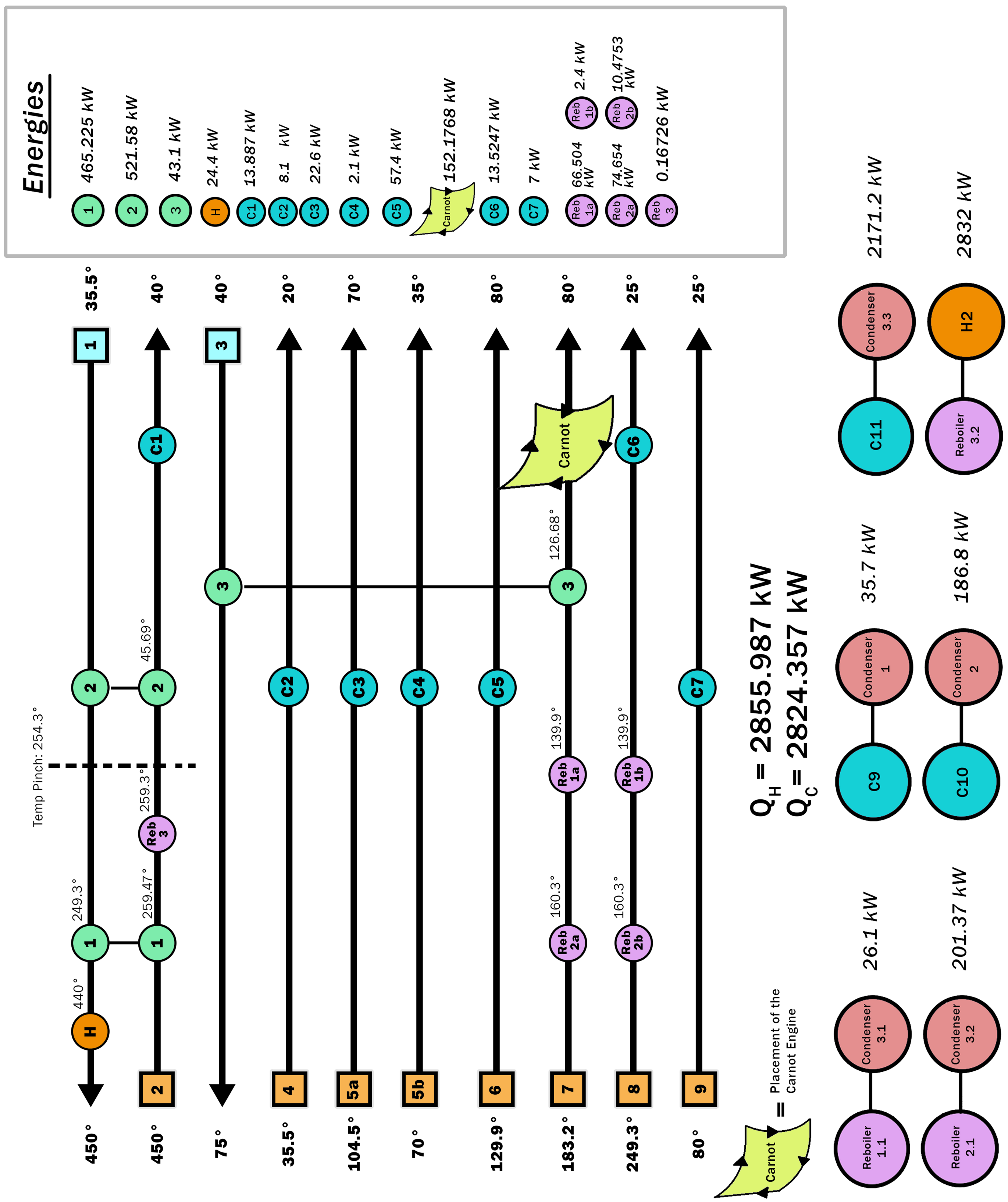


Figure 12: *Carnot Integrated Process Flowsheet*

iii. Steam Raised Network Flowsheet

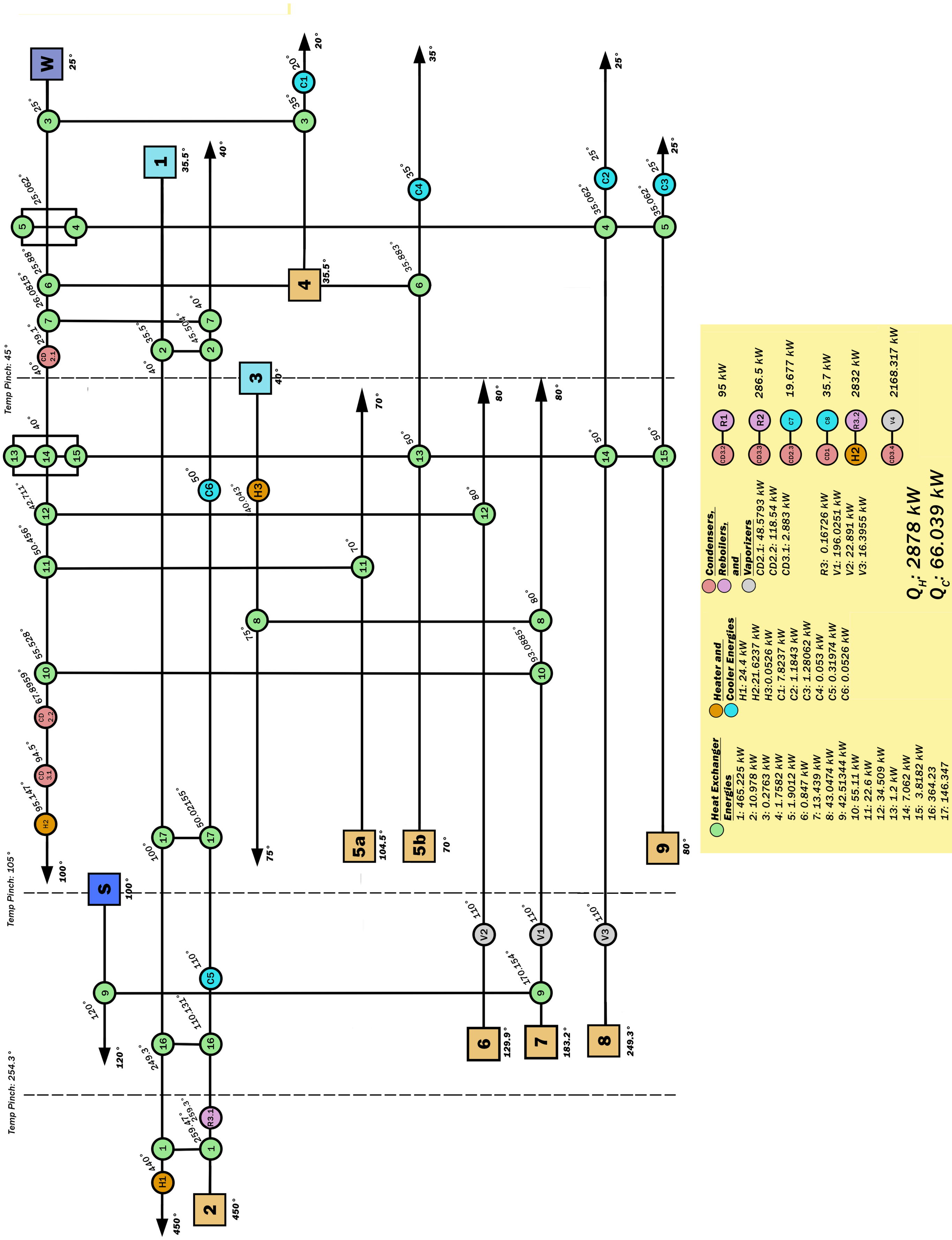
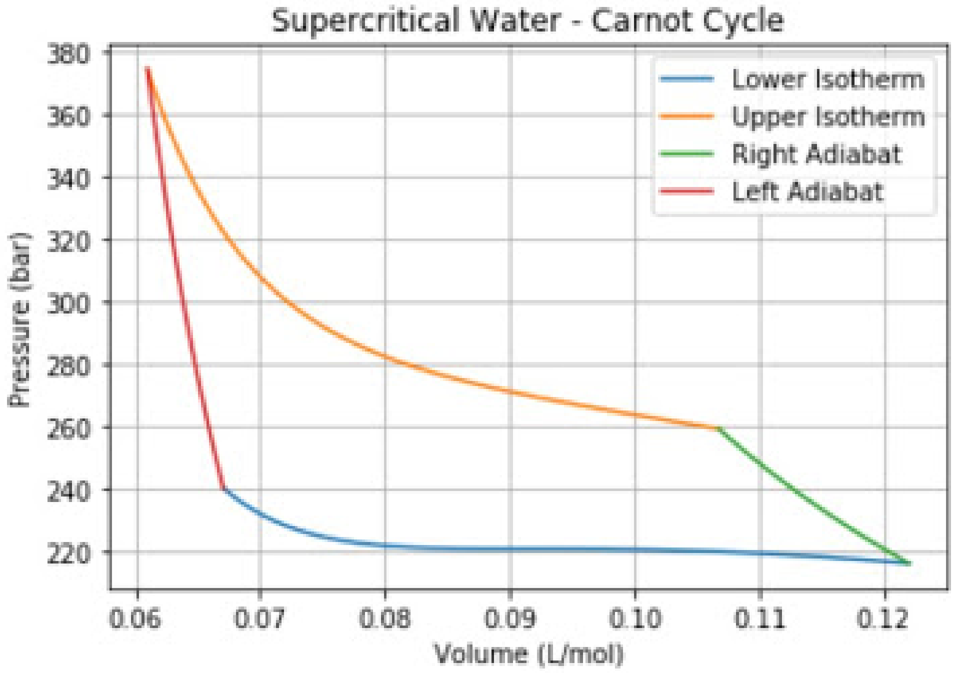


Figure 13: *Steam Raised Process Flowsheet*

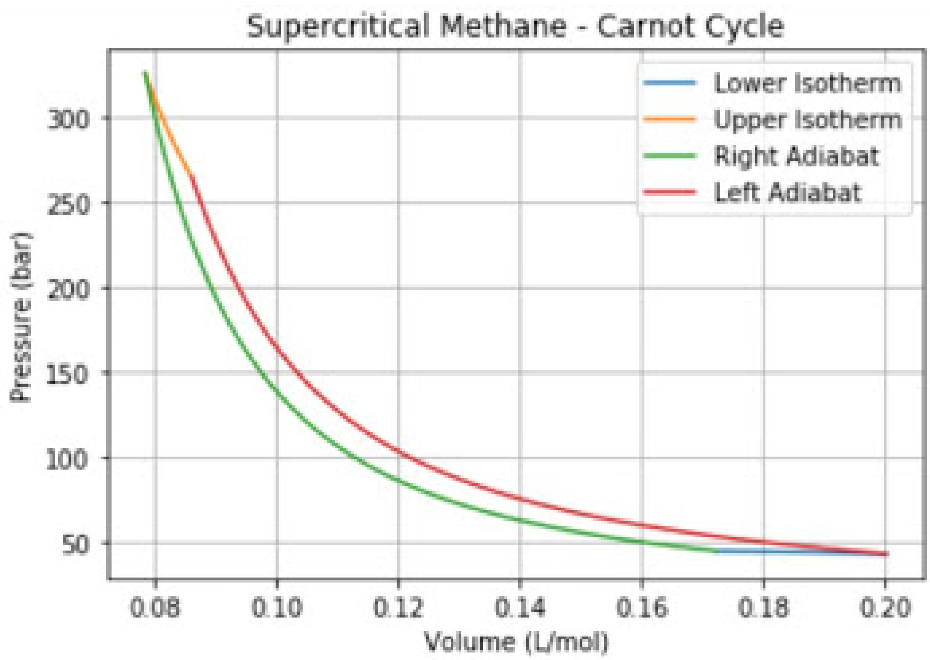
iv. Supercritical Water Carnot Cycle




```
The efficiency of this system is 5.28 %.
The theoretical input energy, Qh, is 5206.802 Joules/sec.
The theoretical work removed from the engine is 4932.038 Joules/sec
The theoretical work of this system is 274.7643 Joules/cycle*mol
The work from the system is 274.7768 Joules/cycle.
The power available from the network is 1521768 Joules/sec
The power lost by the engine is 1441463.9 Joules/sec
The number of moles that the engine contains is 40 moles.
The RPS of the Carnot Engine is 7.306 Cycles/second.
The size of the Carnot Engine is 4.879 L
```

Figure 14: *Steam Raised Process Flowsheet*

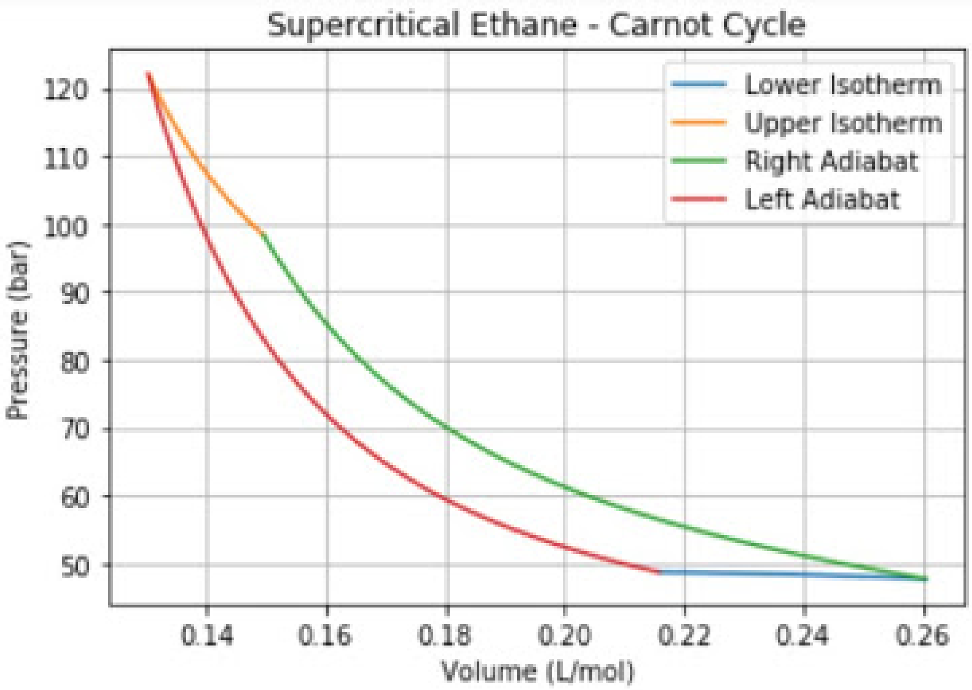
v. Supercritical Methane Carnot Cycle




```
The efficiency of this system is 36.1 %.
The theoretical input energy, Qh, is -485.3017 Joules/sec.
The theoretical work of this system is -175.0602 Joules/cycle*mol
The theoretical work removed from the engine is -310.2415 Joules/sec
The work from the system is -175.0604 Joules/cycle*mol.
The power available from the network is 81000 Joules/sec
The power lost by the engine is 51781.318 Joules/sec
The number of moles that the engine contains is 40 moles.
The RPS of the Carnot Engine is -4.173 Cycles/second.
The size of the Carnot Engine is 6.891 L
```

Figure 15: *Steam Raised Process Flowsheet*

vi. Supercritical Ethane Carnot Cycle




```
The efficiency of this system is 13.5 %.
The theoretical input energy, Qh, is 762.5147 Joules/sec.
The theoretical work of this system is 103.3168 Joules/cycle*mol
The theoretical work removed from the engine is 659.1979 Joules/sec
The work from the system is 103.3172 Joules/cycle.
The power available from the network is 1521768 Joules/sec
The power lost by the engine is 1315576.3 Joules/sec
The number of moles that the engine contains is 40 moles.
The RPS of the Carnot Engine is 49.89 Cycles/second.
The size of the Carnot Engine is 10.42 L
```

Figure 16: *Steam Raised Process Flowsheet*

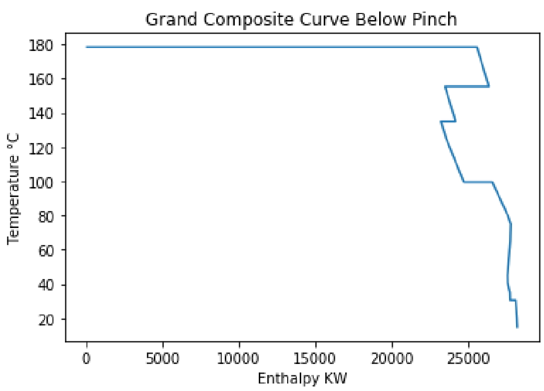
vii. Steam Raising Code

These thermodynamic values were taken from Appendix viii

In [51]:

```
import numpy as np
import matplotlib.pyplot as plt

### I multipy by 10 for all of my enthalpy values so that I can
#test the full scale plant

Qc = 2824.38725*10 #Qc of the real plant
Qh = 2855.9872*10 # Qh of the real plant
#[[30.5, 35.7], [99.5, 186.8], [178.2, 2552.7]]
#[[134.9, -95], [155.3, -286.5], [254.3, -2832]]

temps = [455.0, 445.0, 254.3,254.3,
         244.3, 178.2, 178.2, 155.3,155.3, 134.9,134.9, 124.9,
         99.5, 99.5, 80.0, 75.0, 65.0, 45.0, 40.5, 35.0, 30.5,30.5, 30.0, 20.0, 15.0]

Q = [0,-24.39565742, 0.4083983525000012, -2832, 0.021415750000000067,
     7.921495692060003, 2552.7, 77.36907013433992, -286.5, 68.92266509784002, -95,
     33.785620146, 115.03311043624004, 186.8, 101.0867340483, 19.762532542, -3.292422953,
     -18.486295181999996, 1.3820121535499998, 15.106737546449999, 1.3723750660499998, 35.7,
     0.41377644105, 7.6755288209999994, 2.6129032260000002]

#temps and Q are lists corrisponding to the "corner points" of the GCC
Q = [i * 10 for i in Q] # turnes the Q's into the real plants Q

print(Q)
newq =[]
Sum = Qh

for i in range(len(Q)):
    Sum = Sum + Q[i]
    newq.append(Sum) #creates a list enthaklpuied corresponding from the high temp to low

plt.plot(newq[5::], temps[5::]) # this plots the Grand Composite Curve
plt.title("Grand Composite Curve Below Pinch")
plt.ylabel("Temperature °C")
plt.xlabel("Enthalpy KW")
plt.show()
```

```
[0, -243.95657419999998, 4.083983525000012, -28320, 0.21415750000000067, 79.21495692060003,
25527.0, 773.6907013433993, -2865.0, 689.2266509784002, -950, 337.85620145999997,
1150.3311043624003, 1868.0, 1010.867340483, 197.62532542, -32.92422953, -184.86295181999995,
13.820121535499998, 151.0673754645, 13.723750660499999, 357.0, 4.1377644105, 76.75528820999999,
26.129032260000002]
```



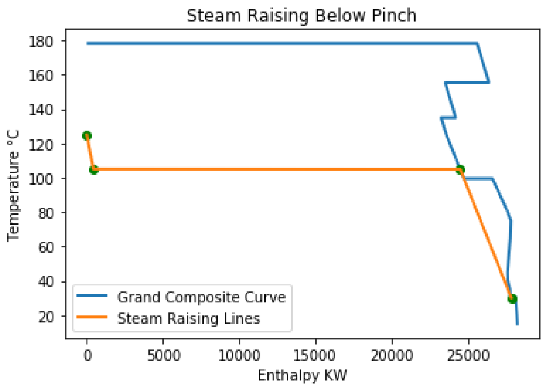
In [53]:

```
Hwsat = 2450*10 #guess from graph
Hvap =2256 #Kj/Kg
Cpw = 4.19 #Kj/Kg*C
Tsat = 105 #C          assumed
Cps = 1.996 #Kj/Kg*C
Tstart = 30 #C
Tfinal = 125 # Final temp of steam
```

```python
Tfinal = 120 # final temp of steam
count = 0

#creating the graph physically
Main_temps = [] #254.3-->30
C = 254.3 #this starts the list below the pinch so we can look at the
#pertinent values of the eqaution
while C>=30:
    Main_temps.append(C)
    C = C-.1 #gves temperature values for the


graphenthalpy = [] #the raw data


for i in range(len(temps)-1):
    slope = (temps[i+1]-temps[i])/(newq[i+1]-newq[i]) #this calculates the slope.
    if slope == 0:
        slope =.000000001
    for j in range(len(Main_temps)):
        if (Main_temps[j] < temps[i] and Main_temps[j]
            > temps[i+1]):
            H  = (1/slope)*(Main_temps[j]-temps[i])
            +newq[i]
            graphenthalpy.append(H)
Main_temps.pop() #removes the last element from the list


#making the optomal steam lines based on equations
graphenthalpysteam = []
graphtempssteam = []
holder = True
while holder == True: # thsi is ideration,
    count = 0

    Hssat = Cps*Hwsat*(Tfinal-Tsat)/(Hvap +Cps*
                                     (Tfinal - Tsat))
    Hstart = Hwsat*(1+Cpw*((Tsat-Tstart)/Hvap))-Hwsat*
    ((Cpw*Cps*(Tsat-Tstart)/Hvap)
    *((Tfinal-Tsat)/(Hvap+Cps*(Tfinal-Tsat))))
    Mw = (Hwsat/Hvap)*(1-Cps*(Tfinal-Tsat)/(
        Hvap + Cps*(Tfinal-Tsat)))
    crit_S_enthalpies = [0,Hssat,Hwsat,Hstart]
    # important x values for slope
    crit_S_temps = [Tfinal,Tsat,Tsat,Tstart]
    # important y values for slope
    graphenthalpysteam = []
    #the graphing enthalpies for steam
    graphtempssteam = []
    #the graphing temps for steam

    for i in range(len(crit_S_temps)-1):
        slope = (crit_S_temps[i+1]-crit_S_temps[i])/
        (crit_S_enthalpies[i+1]-crit_S_enthalpies[i])
        if slope == 0:
            slope = .000000001
            #Deals with when you get an undefined slope aka a horizonal line.
        for j in range(len(Main_temps)):
            if Main_temps[j] < crit_S_temps[i] and
            Main_temps[j] > crit_S_temps[i+1]:
                H  = (1/slope)*(Main_temps[j]-
                        crit_S_temps[i])+crit_S_enthalpies[i]
                graphenthalpysteam.append(H)
                #appends the enthalpy at a specific temoperature
                graphtempssteam.append(Main_temps[j])
                #appends the temperature used to caculate the enthalpy


    if Hwsat > 2449.797*10: #this makes sure that the
                            #value of Hwsat is always
                            #barely touvhing th GCC
        count = count + 1
    #this let me check for any other values where the GCC may cross.
    if graphenthalpysteam[graphtempssteam.index(30.500000000010353)]
    > 2813.85*10:
        count = count + 1
```

```
    if count >= 1 :
        Hwsat = Hwsat-.1
        # this is where we decrement Hwsat

    if count == 0:
        print("the mass flow rate is",Mw)
        # this should print the max flow rate
        # possible in the GCC.
        if Hssat <= 0:
            print("no solution can be found due to Hssat")
        holder = False



plt.plot(newq[5::], temps[5::],linewidth=2, label='Grand Composite Curve')# this takes the upper
half of the system out of the graph
plt.plot(Hstart,Tstart, marker = 'o', color = "green") # important point
plt.plot(Hwsat,Tsat, marker = 'o', color = "green")# important point
plt.plot(0, Tfinal, marker = 'o', color = "Green")# important point
plt.plot(Hssat,Tsat,marker = 'o', color = "Green")# important point
plt.plot(graphenthalpysteam,graphtempssteam,linewidth=2, label='Steam Raising Lines')
plt.title("Steam Raising Below Pinch")
plt.ylabel("Temperature °C")
plt.xlabel("Enthalpy KW")
plt.legend(loc ="best")
plt.show()
#print(Hstart)
```

```
the mass flow rate is 10.670188856754605
```



In [4]:

```
# Value Finder for graph
#for i in range(len(Main_temps)):
 #    if Main_temps[i] <=35 and Main_temps[i]>= 25:
 #        print(graphenthalpy[i])
```

In [37]:

```
for i in range(len(graphtempssteam)):
    if graphtempssteam[i] <=29 and graphtempssteam[i]>= 31 :
        print(graphenthalpysteam[i])
```

viii. Table of Steam Raising Variables

Table 14: *Variables Used Within Steam Raising Equations. All values come from Introduction to Chemical Engineering [10]*

Variable	Definition	Units	Value
M_w	Mass Flow Rate	kg/S	Unknown
C_{pw}	Heat Capacity of Water In	kJ/kg*°C	4.19
C_{ps}	Heat Capacity of Steam In	kJ/kg*°C	1.996
H_{start}	Enthalpy of the Inlet Water	kW	Unknown
T_{start}	Temperature of the Inlet Water	°C	20°C
H_{wsat}	Enthalpy of Saturated Water In	kW	Manually Changed
T_{sat}	Water Vaporization Temperature	°C	100°(105°shifted)
H_{ssat}	Enthalpy of Saturated Steam In	kW	unknown
ΔH_{vap}	Vaporization Enthalpy In	kJ/Kg	2265
T_{final}	Final Temperature of Steam In	°C	120°C

ix. Supercritical CO2 Carnot Cycle Code

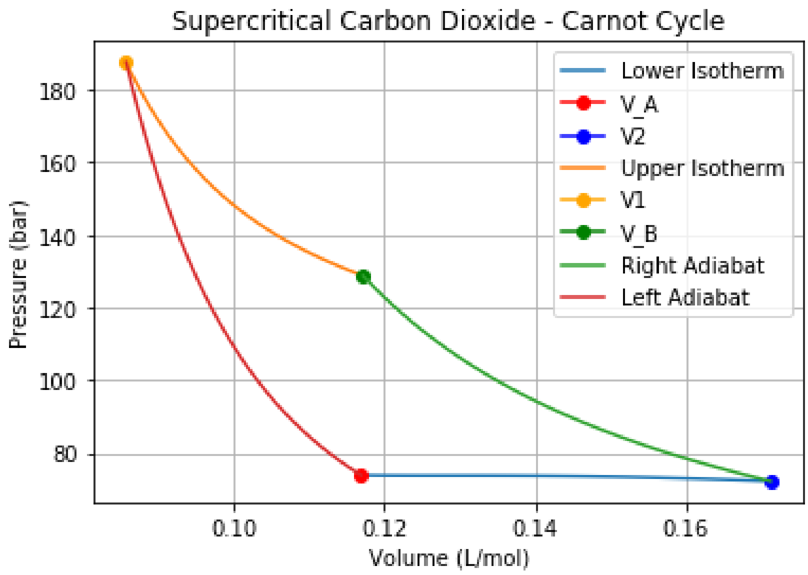
Carnot Cycle - Super Critical CO2

November 19, 2018

```
In [21]: import numpy as np
         import matplotlib.pyplot as plt

In [184]: R = .08314 #L-bar/mol*K
          Tc = 304.1 #K, lower isotherm
          Pc = 73.8 #bar
          Th = 353.15 #80 degrees Celsius, upper isotherm
          r = 8.314 #J/mol*K

          n = 1 - (Tc/Th) #Efficiency of the Carnot engine equation

          a = 27/64*(R**2)*(Tc**2)/Pc #Parameter calculations that apply to the Van der Waal
          b = 1/8*R*Tc/Pc

          Vc = 3/8*R*Tc/Pc #Critical Volume

          CompRatio = 2 #V2/V1
          V1 = .085647
          V2 = V1*CompRatio

          Tspace = np.linspace(Tc, Th, 1000) #array of temperatures that will be used to est
          list(Tspace) #converting the array to a list

          A = 5.457 #Heat Capacity coefficients
          B = 1.045*10**-3
          C = 0
          D = -1.157*10**5
          y = 1.29 #Cp - Cv

          testCv = (R*(A + B*Tspace + C*(Tspace**2) + D*(Tspace**-2)))/y #Calculating Cv
          Cv = np.mean(testCv)

          Var = ((V2-b)*((Tc/Tspace)**(Cv/R))) + b #Volume range of the right adiabat
          Par = ((R*Tspace)/(Var-b))-(a/(Var**2)) #Pressure list based on parameters, temp l

          Val = ((V1-b)*((Th/Tspace)**(Cv/R))) + b #Volume range of the left adiabat
          Pal = ((R*Tspace)/(Val-b))-(a/(Val**2)) #Pressure list based on parameters, temp l
```

```
hVspace = np.linspace(V1, Var[-1], 100) #Volume range for upper isotherm
Ph = ((R*Th)/(hVspace-b))-(a/(hVspace**2)) #Pressure list for the upper isotherm
cVspace = np.linspace(Val[0], V2, 100) #Volume range for lower isotherm
Pc = ((R*Tc)/(cVspace-b))-(a/(cVspace**2)) #Pressure list for lower isotherm

#Plotting all curves, adding specific points

plt.plot(cVspace, Pc, label = 'Lower Isotherm')
plt.plot(cVspace[0], Pc[0], marker = 'o', color = 'red', label = "V_A")
plt.plot(cVspace[-1], Pc[-1], marker = 'o', color = 'blue', label = "V2")
plt.plot(hVspace, Ph, label = 'Upper Isotherm')
plt.plot(hVspace[0], Ph[0], marker = 'o', color = 'orange', label = "V1")
plt.plot(hVspace[-1], Ph[-1], marker = 'o', color = 'green', label = "V_B")
plt.plot(Var, Par, label= 'Right Adiabat')
plt.plot(Val, Pal, label = 'Left Adiabat')
plt.grid(True)
plt.xlabel('Volume (L/mol)')
plt.ylabel('Pressure (bar)')
plt.title("Supercritical Carbon Dioxide - Carnot Cycle")
plt.legend(loc = 'best')
plt.show()

#Finding the work associated with the plot of the graph

War = -np.trapz(Par, Var)
Wth = np.trapz(Ph, hVspace)
Wal = -np.trapz(Pal, Val)
Wtc = np.trapz(Pc, cVspace)

#Using equations given to solve for and report findings for this supercritical flu

print('The efficiency of this system is {:.3}'.format(n*100), '%.')

Qh = r*Th*np.log((Var[-1]-b)/(V1-b)) #Qh for one cycle
print('The theoretical input energy, Qh, is {:.7}'.format(Qh), "Joules/sec.")

TheoWORK = Qh*(1-(Tc/Th))
print('The theoretical work of this system is {:.7}'.format(TheoWORK), "Joules/cycl

TheoQc = Qh*(1-n)
print('The theoretical work removed from the engine is {:.7}'.format(TheoQc), "Joul

WORK = ((War+Wth)-(Wal+Wtc))*100
print('The work from the system is {:.7}'.format(WORK), "Joules/cycle.")

Qhnetwork = 1521768 #From cooler 5, W #Full scale size, 10x greater than the numbe
print('The power available from the network is',(Qhnetwork), 'Joules/sec')
```

```
Qcnetwork = Qhnetwork*(1-n)
print('The power lost by the engine is {:.8}'.format(Qcnetwork), 'Joules/sec')

Moles = 40
print('The number of moles that the engine contains is',(Moles), 'moles.')

Hz = (n*Qhnetwork)/(WORK*Moles)
print('The RPS of the Carnot Engine is {:.4}'.format(Hz), "Cycles/second.")

V_Engine = V2*Moles
print('The size of the Carnot Engine is {:.4}'.format(V_Engine), "L")

Work_Cost_Savings = WORK*Hz*Moles*300*24*(1/1000)*(1/6.67)*(60/1000)*(1/.3)
print('The amount of money saved by utilizing this engine is ${:.6}'.format(Work_Co

Cooler_Cost_Savings = (Qhnetwork/1000)*24*300*(1/6.67)*(60/1000)*(1/.3)
print('The amount of money saved by no longer needing to run the cooler is ${:.7}'.
```




```
The efficiency of this system is 13.9 %.
The theoretical input energy, Qh, is 1618.123 Joules/sec.
The theoretical work of this system is 224.7457 Joules/cycle*mol
The theoretical work removed from the engine is 1393.377 Joules/sec
The work from the system is 224.7488 Joules/cycle.
```

```
The power available from the network is 1521768 Joules/sec
The power lost by the engine is 1310405.3 Joules/sec
The number of moles that the engine contains is 40 moles.
The RPS of the Carnot Engine is 23.51 Cycles/second.
The size of the Carnot Engine is 6.852 L
The amount of money saved by utilizing this engine is $45631.5
The amount of money saved by no longer needing to run the cooler is $328537.6
```

x. Supercritical Methane Carnot Cycle Code

Carnot Cycle - Super Critical Methane

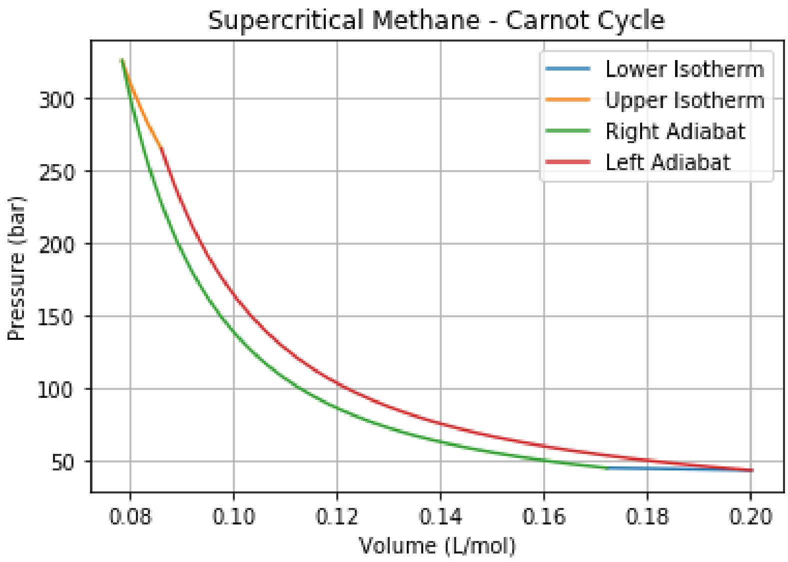
November 19, 2018

In [34]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [44]:
```
R = .08314 #L-bar/mol*K
Tc = 190.6 #K, lower isotherms
Pc = 45.99 #bar, critical pressure
Th = 298.15 #20 degrees celsius, best attempt to integrate Methane to the network
r = 8.314 #J/mol*K

n = 1 - (Tc/Th) #efficiency of the Carnot Engine

a = 27/64*(R**2)*(Tc**2)/Pc #Parameter calculations that apply to the Van der Waal
b = 1/8*R*Tc/Pc

Vc = 3/8*R*Tc/Pc #Critical Volume

CompRatio = 2 #V2/V1
V1 = .086141
V2 = V1*CompRatio

Tspace = np.linspace(Tc, Th, 1000) #array of temperatures that will be used to est
list(Tspace)  #converting the array to a list

A = 1.702 #Heat Capacity coefficient
B = 9.081*10**-3
C = -2.164*10**-6
D = 0
y = 1.31

testCv = (R*(A + B*Tspace + C*(Tspace**2) + D*(Tspace**-2)))/y #Calculating Cv
Cv = np.mean(testCv)

Var = ((V2-b)*((Tc/Tspace)**(Cv/R))) + b #Volume range of the right adiabat
Par = ((R*Tspace)/(Var-b))-(a/(Var**2)) #Pressure list based on parameters, temp l

Val = ((V1-b)*((Th/Tspace)**(Cv/R))) + b #Volume range of the left adiabat
Pal = ((R*Tspace)/(Val-b))-(a/(Val**2)) #Pressure list based on parameters, temp l
```

```
hVspace = np.linspace(V1, Var[-1], 100) #Volume range for upper isotherm
Ph = ((R*Th)/(hVspace-b))-(a/(hVspace**2))  #Pressure list for the upper isotherm
cVspace = np.linspace(Val[0], V2, 100)  #Volume range for lower isotherm
Pc = ((R*Tc)/(cVspace-b))-(a/(cVspace**2)) #Pressure list for lower isotherm

#Plotting all curves

plt.plot(cVspace, Pc, label = 'Lower Isotherm')
plt.plot(hVspace, Ph, label = 'Upper Isotherm')
plt.plot(Var, Par, label = 'Right Adiabat')
plt.plot(Val, Pal, label = 'Left Adiabat')
plt.grid(True)
plt.xlabel('Volume (L/mol)')
plt.ylabel('Pressure (bar)')
plt.title("Supercritical Methane - Carnot Cycle")
plt.legend(loc = 'best')
plt.show()

#Finding the work associated with the plot of the graph

War = -np.trapz(Par, Var)
Wth = np.trapz(Ph, hVspace)
Wal = -np.trapz(Pal, Val)
Wtc = np.trapz(Pc, cVspace)

#Using equations given to solve for and report findings for this supercritical flu

print('The efficiency of this system is {:.3}'.format(n*100), '%.')

Qh = r*Th*np.log((Var[-1]-b)/(V1-b)) #Qh for one cycle
print('The theoretical input energy, Qh, is {:.7}'.format(Qh), "Joules/sec.")

TheoWORK = Qh*(1-(Tc/Th))
print('The theoretical work of this system is {:.7}'.format(TheoWORK), "Joules/cycl

TheoQc = Qh*(1-n)
print('The theoretical work removed from the engine is {:.7}'.format(TheoQc), "Joul

WORK = ((War+Wth)-(Wal+Wtc))*100
print('The work from the system is {:.7}'.format(WORK), "Joules/cycle*mol.")

Qhnetwork = 81000
print('The power available from the network is',(Qhnetwork), 'Joules/sec')

Qcnetwork = Qhnetwork*(1-n)
print('The power lost by the engine is {:.8}'.format(Qcnetwork), 'Joules/sec')
```

```
Moles = 40
print('The number of moles that the engine contains is',(Moles), 'moles.')

Hz = (n*Qhnetwork)/(WORK*Moles)
print('The RPS of the Carnot Engine is {:.4}'.format(Hz), "Cycles/second.")

V_Engine = V2*Moles
print('The size of the Carnot Engine is {:.4}'.format(V_Engine), "L")
```




```
The efficiency of this system is 36.1 %.
The theoretical input energy, Qh, is -485.3017 Joules/sec.
The theoretical work of this system is -175.0602 Joules/cycle*mol
The theoretical work removed from the engine is -310.2415 Joules/sec
The work from the system is -175.0604 Joules/cycle*mol.
The power available from the network is 81000 Joules/sec
The power lost by the engine is 51781.318 Joules/sec
The number of moles that the engine contains is 40 moles.
The RPS of the Carnot Engine is -4.173 Cycles/second.
The size of the Carnot Engine is 6.891 L
```

xi. Supercritical Ethane Carnot Cycle Code

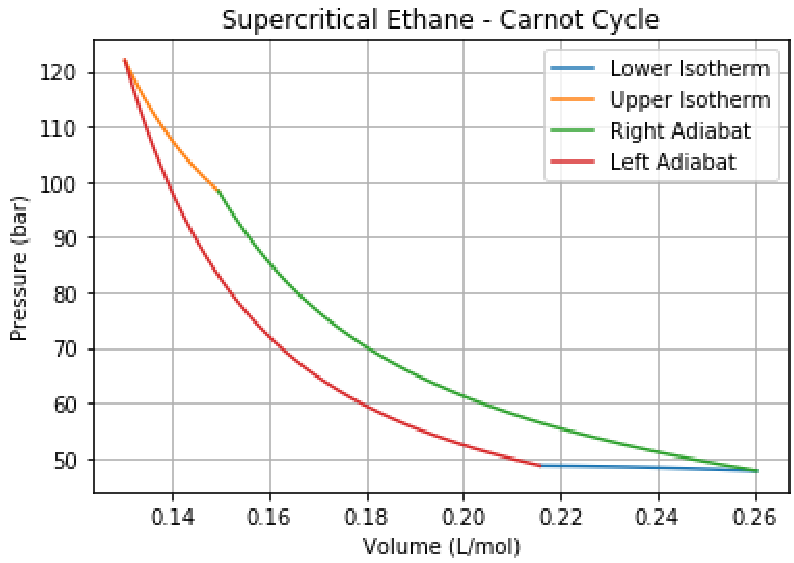
Carnot Cycle - Super Critical Ethane

November 19, 2018

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt

In [23]: R = .08314 #L-bar/mol*K
         Tc = 305.3 #K, critical temperature
         Pc = 48.72 #bar
         Th = 353.15 #80 degrees Celsius, upper isotherm

         n = 1 - (Tc/Th) #Efficiency of the Carnot engine equation

         a = 27/64*(R**2)*(Tc**2)/Pc #Parameter calculations that apply to the Van der Waal'
         b = 1/8*R*Tc/Pc

         Vc = 3/8*R*Tc/Pc #Critical Volume

         CompRatio = 2 #V2/V1
         V1 = .130248
         V2 = V1*CompRatio

         Tspace = np.linspace(Tc, Th, 1000) #array of temperatures that will be used to esta
         list(Tspace) #converting the array to a list

         A = 1.131 #Heat Capacity coefficients
         B = 19.225*10**-3
         C = -5.561*10**-6
         D = 0
         y = 1.19

         testCv = (R*(A + B*Tspace + C*(Tspace**2) + D*(Tspace**-2)))/y
         Cv = np.mean(testCv) #Calculating Cv

         Var = ((V2-b)*((Tc/Tspace)**(Cv/R))) + b #Volume range of the right adiabat
         Par = ((R*Tspace)/(Var-b))-(a/(Var**2))  #Pressure list based on parameters, temp l

         Val = ((V1-b)*((Th/Tspace)**(Cv/R))) + b #Volume range of the left adiabat
         Pal = ((R*Tspace)/(Val-b))-(a/(Val**2)) #Pressure list based on parameters, temp li
```

```
hVspace = np.linspace(V1, Var[-1], 100) #Volume range for upper isotherm
Ph = ((R*Th)/(hVspace-b))-(a/(hVspace**2)) #Pressure list for the upper isotherm
cVspace = np.linspace(Val[0], V2, 100) #Volume range for lower isotherm
Pc = ((R*Tc)/(cVspace-b))-(a/(cVspace**2)) #Pressure list for lower isotherm

#Plotting all curves

plt.plot(cVspace, Pc, label = 'Lower Isotherm')
plt.plot(hVspace, Ph, label = 'Upper Isotherm')
plt.plot(Var, Par, label = 'Right Adiabat')
plt.plot(Val, Pal, label = 'Left Adiabat')
plt.grid(True)
plt.xlabel('Volume (L/mol)')
plt.ylabel('Pressure (bar)')
plt.title("Supercritical Ethane - Carnot Cycle")
plt.legend(loc = 'best')
plt.show()

#Finding the work associated with the plot of the graph

War = -np.trapz(Par, Var)
Wth = np.trapz(Ph, hVspace)
Wal = -np.trapz(Pal, Val)
Wtc = np.trapz(Pc, cVspace)

#Using equations given to solve for and report findings for this supercritical flui

print('The efficiency of this system is {:.3}'.format(n*100), '%.')

Qh = r*Th*np.log((Var[-1]-b)/(V1-b)) #Qh for one cycle
print('The theoretical input energy, Qh, is {:.7}'.format(Qh), "Joules/sec.")

TheoWORK = Qh*(1-(Tc/Th))
print('The theoretical work of this system is {:.7}'.format(TheoWORK), "Joules/cycl

TheoQc = Qh*(1-n)
print('The theoretical work removed from the engine is {:.7}'.format(TheoQc), "Joul

WORK = ((War+Wth)-(Wal+Wtc))*100
print('The work from the system is {:.7}'.format(WORK), "Joules/cycle.")

Qhnetwork = 1521768 #From cooler 5, W #Full scale size, 10x greater than the number
print('The power available from the network is',(Qhnetwork), 'Joules/sec')

Qcnetwork = Qhnetwork*(1-n)
print('The power lost by the engine is {:.8}'.format(Qcnetwork), 'Joules/sec')

Moles = 40
```

```
print('The number of moles that the engine contains is',(Moles), 'moles.')

Hz = (n*Qhnetwork)/(WORK*Moles)
print('The RPS of the Carnot Engine is {:.4}'.format(Hz), "Cycles/second.")

V_Engine = V2*Moles
print('The size of the Carnot Engine is {:.4}'.format(V_Engine), "L")
```




```
The efficiency of this system is 13.5 %.
The theoretical input energy, Qh, is 762.5147 Joules/sec.
The theoretical work of this system is 103.3168 Joules/cycle*mol
The theoretical work removed from the engine is 659.1979 Joules/sec
The work from the system is 103.3172 Joules/cycle.
The power available from the network is 1521768 Joules/sec
The power lost by the engine is 1315576.3 Joules/sec
The number of moles that the engine contains is 40 moles.
The RPS of the Carnot Engine is 49.89 Cycles/second.
The size of the Carnot Engine is 10.42 L
```

xii. Supercritical Water Carnot Cycle Code

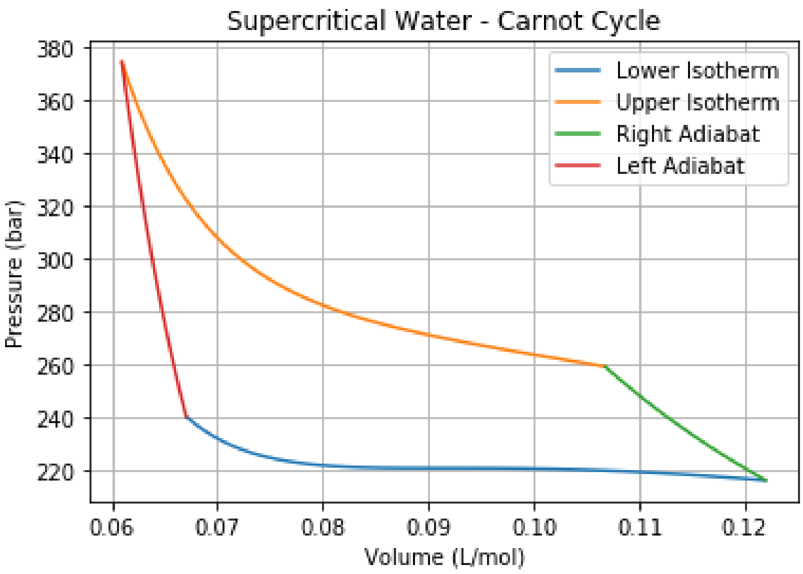
Carnot Cycle - Super Critical H2O

November 19, 2018

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [14]: R = .08314 #L-bar/mol*K
         Tc = 647.1 #K, critical temperature
         Pc = 220.55 #bar, critical pressure
         Th = 683.15 #410 degrees Celsius
         r = 8.314 #J/mol*K

         n = 1 - (Tc/Th) #Efficiency of the Carnot engine equation

         a = 27/64*(R**2)*(Tc**2)/Pc #Parameter calculations that apply to the Van der Waal's
         b = 1/8*R*Tc/Pc

         Vc = 3/8*R*Tc/Pc #Critical Volume

         CompRatio = 2 #V2/V1
         V1 = .060984
         V2 = V1*CompRatio

         Tspace = np.linspace(Tc, Th, 1000) #array of temperatures that will be used to estal
         list(Tspace) #converting the array to a list

         A = 3.470 #Heat Capacity coefficients
         B = 1.450*10**-3
         C = 0
         D = 0.121*10**5
         y = 1.33

         testCv = (R*(A + B*Tspace + C*(Tspace**2) + D*(Tspace**-2)))/y #Calculating Cv
         Cv = np.mean(testCv)

         Var = ((V2-b)*((Tc/Tspace)**(Cv/R))) + b #Volume range of the right adiabat
         Par = ((R*Tspace)/(Var-b))-(a/(Var**2)) #Pressure list based on parameters, temp lis

         Val = ((V1-b)*((Th/Tspace)**(Cv/R))) + b #Volume range of the left adiabat
         Pal = ((R*Tspace)/(Val-b))-(a/(Val**2)) #Pressure list based on parameters, temp lis
```

```
hVspace = np.linspace(V1, Var[-1], 100) #Need to keep this array at a length of 100
Ph = ((R*Th)/(hVspace-b))-(a/(hVspace**2))  #Pressure list for the upper isotherm
cVspace = np.linspace(Val[0], V2, 100) #Volume range for lower isotherm
Pc = ((R*Tc)/(cVspace-b))-(a/(cVspace**2)) #Pressure list for lower isotherm

#Plotting all curves

plt.plot(cVspace, Pc, label = 'Lower Isotherm')
plt.plot(hVspace, Ph, label = 'Upper Isotherm')
plt.plot(Var, Par, label = 'Right Adiabat')
plt.plot(Val, Pal, label = 'Left Adiabat')
plt.grid(True)
plt.xlabel('Volume (L/mol)')
plt.ylabel('Pressure (bar)')
plt.title("Supercritical Water - Carnot Cycle")
plt.legend(loc = 'best')
plt.show()

#Finding the work associated with the plot of the graph

War = -np.trapz(Par, Var)
Wth = np.trapz(Ph, hVspace)
Wal = -np.trapz(Pal, Val)
Wtc = np.trapz(Pc, cVspace)

#Using equations given to solve for and report findings for this supercritical flui

print('The efficiency of this system is {:.3}'.format(n*100), '%.')

Qh = r*Th*np.log((Var[-1]-b)/(V1-b)) #Qh for one cycle
print('The theoretical input energy, Qh, is {:.7}'.format(Qh), "Joules/sec.")

TheoQc = Qh*(1-n)
print('The theoretical work removed from the engine is {:.7}'.format(TheoQc), "Joule

TheoWORK = Qh*(1-(Tc/Th))
print('The theoretical work of this system is {:.7}'.format(TheoWORK), "Joules/cycle

WORK = ((War+Wth)-(Wal+Wtc))*100
print('The work from the system is {:.7}'.format(WORK), "Joules/cycle.")

Qhnetwork = 976668 #From cooler 5, W #Full scale size, 10x greater than the numbers
print('The power available from the network is',(Qhnetwork), 'Joules/sec')

Qcnetwork = Qhnetwork*(1-n)
print('The power lost by the engine is {:.8}'.format(Qcnetwork), 'Joules/sec')
```

```python
Moles = 40
print('The number of moles that the engine contains is',(Moles), 'moles.')

Hz = (n*Qhnetwork)/(WORK*Moles)
print('The RPS of the Carnot Engine is {:.4}'.format(Hz), "Cycles/second.")

V_Engine = V2*Moles
print('The size of the Carnot Engine is {:.4}'.format(V_Engine), "L")
```




```
The efficiency of this system is 5.28 %.
The theoretical input energy, Qh, is 5206.802 Joules/sec.
The theoretical work removed from the engine is 4932.038 Joules/sec
The theoretical work of this system is 274.7643 Joules/cycle*mol
The work from the system is 274.7768 Joules/cycle.
The power available from the network is 976668 Joules/sec
The power lost by the engine is 925128.98 Joules/sec
The number of moles that the engine contains is 40 moles.
The RPS of the Carnot Engine is 4.689 Cycles/second.
The size of the Carnot Engine is 4.879 L
```

xiii. Pinch Calculations at Tmin of 10 degrees C Code

In [518]:

```
#COLUMN INTEGRATION

CP = [2.439565742, 2.441707317, 1.231428571, 0.5225806452, 0.6550724638, 0.06, 1.150300601, 3.25872
093, 0.1176995096, 0.1272727273, 0,0, 0, 0, 0, 0] #input CP values (0 for columns)

Ts = [35.5, 450, 40, 35.5, 104.5, 70, 129.9, 183.2, 249.3, 80, 35.5, 129.9, 104.5, 150.3, 183.2, 24
9.3] #input temperatures at inlets

Tt = [450, 40, 75, 20, 70, 35, 80, 80, 25, 25, 35.5, 129.9, 104.5, 150.3, 183.2, 249.3] #input
temperatures at outlets

Qcol = [0,0,0,0,0,0,0,0,0,0, 35.7, -95, 186.8, -286.5, 2552.7, -2832] #input 0 if not a column, -Qre
b for reboiler, or Qcond for condenser

Tmin = 10 #input Tmin

o = list(range(len(CP))) #list of length of the number of streams
Tss = list(range(len(Ts))) #list that will become shifted inlet temperatures
Tts = list(range(len(Tt))) #list that will become shifted outlet temperatures

coldnum = [] #list used to identify cold streams
hotnum = [] #list used to identify hot streams
disnumC = [] #list used to identify condensers
disnumR = [] #list used to identify reboilers
columntempsC = [] #used to identify where the condenders fit into the heat cascade
columntempsR = [] #used to identify where the reboilers fit into the heat cascade

for i in o: #function creating modified temps and identifying hot vs cold streams
    if Ts[i] > Tt[i]:
        Tss[i] = Ts[i]-(Tmin*.5)
        Tts[i] = Tt[i]-(Tmin*.5)
        hotnum.append(i)

    if Ts[i] < Tt[i]:
        Tss[i] = Ts[i]+(Tmin*.5)
        Tts[i] = Tt[i]+(Tmin*.5)
        coldnum.append(i)

    if Ts[i] == Tt[i]: #identifies columns/shifts temps
        if Qcol[i] > 0:
            Tss[i] = Ts[i]-(Tmin*.5)
            Tts[i] = Tt[i]-(Tmin*.5)
            disnumC.append(i)
            columntempsC.append([Tts[i], Qcol[i]])
        if Qcol[i] < 0:
            Tss[i] = Ts[i]+(Tmin*.5)
            Tts[i] = Tt[i]+(Tmin*.5)
            disnumR.append(i)
            columntempsR.append([Tts[i], Qcol[i]])
```

In [519]:

```
Tall = [] #list of all shifted temperatures
for i in o:
    Tall.append(Tss[i])
    Tall.append(Tts[i])

Tstar = list(set(Tall)) #list of all temperatures, without repeats, in descending order
Tstar.sort(reverse=True)
print(Tstar) #HEAT CASCADE TEMPS

print()

CPtot = [] #will be list of net CPs for each temp interval
lenT = list(range(len(Tstar) - 1)) #list for iterating through temperature intervals
p=0
lentempc = list(range(len(coldnum))) #iterates over cold streams
lentemph = list(range(len(hotnum))) #iterates over hot streams

#function to calculate net CPs
for i in lenT: #iterate through temp intervals
```

```
    p=0
    for j in o: #iterate through streams
        for k in lentempc:
            if j == coldnum[k]: #if cold, increase temp in integer increments to see
                                #if it's in the selected temp range
                x = Tss[j]
                while Tts[j] >= x:
                    if -.005 +x <Tstar[i] -.1 < x +.005:
                        p = p-CP[j]
                    x = x+.1
        for k in lentemph: #same for hot streams, but decreasing temp
            if j == hotnum[k]:
                x = Tss[j]
                while Tts[j] <= x:
                    if -.005 + x<Tstar[i] -.1 <x+.005:
                        p = p+CP[j]
                    x = x-.1
    CPtot.append(p)
print(CPtot) #NET CP VALUES
```

```
[455.0, 445.0, 254.3, 244.3, 178.2, 155.3, 134.9, 124.9, 99.5, 80.0, 75.0, 65.0, 45.0, 40.5, 35.0, 30.5, 30.0, 20.0, 15.0]

[-2.439565742, 0.0021415750000000067, 0.0021415750000000067, 0.11984108460000001, 3.3785620146, 3.3785620146, 3.3785620146, 4.5288626156000005, 5.1839350794, 3.9525065084, -0.3292422953, -0.9243147590999998, 0.3071138119, 2.7466795539, 0.30497223689999997, 0.8275528821, 0.7675528821, 0.5225806452]
```

In [520]:

```
H = [] #list of delta H values for intervals
Hval = list(range((len(Tstar) - 1))) #list to iterate through temperature intervals

for i in Hval:
    H.append((Tstar[i]-Tstar[i+1])*(CPtot[i])) #calculates delta H

print(H) #delta Hs before column integration
print()

count = 0
for i in range(len(Tstar)): #adds in column values
    for j in range(len(columntempsR)):
        if Tstar[i] == columntempsR[j][0]:
            H.insert(count + i, columntempsR[j][1])
            count = count+1
    for j in range(len(columntempsC)):
        if Tstar[i] == columntempsC[j][0]:
            H.insert(count + i, columntempsC[j][1])
            count = count+1

print(H) #delta Hs after column integration
```

```
[-24.39565742, 0.4083983525000012, 0.021415750000000067, 7.921495692060003, 77.36907013433992, 68.92266509784002, 33.785620146, 115.03311043624004, 101.0867340483, 19.762532542, -3.292422953, -18.486295181999996, 1.3820121535499998, 15.106737546449999, 1.3723750660499998, 0.41377644105, 7.6755288209999994, 2.6129032260000002]

[-24.39565742, 0.4083983525000012, -2832, 0.021415750000000067, 7.921495692060003, 2552.7, 77.36907013433992, -286.5, 68.92266509784002, -95, 33.785620146, 115.03311043624004, 186.8, 101.0867340483, 19.762532542, -3.292422953, -18.486295181999996, 1.3820121535499998, 15.106737546449999, 1.3723750660499998, 35.7, 0.41377644105, 7.6755288209999994, 2.6129032260000002]
```

In [521]:

```
Hlen = list(range(len(H))) #list of delta H values for intervals

Hsum = []
n = 0
for i in Hlen:
    n = n+H[i]
    Hsum.append(n)
Qh = -min(Hsum) #Qh is -largest negative number when adding through H
```

```
print(Qh) #Qh
Hnewsum = list(range(len(Hsum))) #***HEAT CASCADE Hs***

for i in Hlen:
    Hnewsum[i] = Hsum[i] + Qh
print()
print(Hnewsum) #Heat Cascade after Qh
```

```
2855.9872590675

[2831.5916016475, 2832.0, 0.0, 0.0214157499999601, 7.942911442059994, 2560.64291144206,
2638.0119815763996, 2351.5119815763996, 2420.43464667424, 2325.43464667424, 2359.22026682024,
2474.25337725648, 2661.05337725648, 2762.1401113047796, 2781.90264384678, 2778.61022089378,
2760.12392571178, 2761.50593786533, 2776.61267541178, 2777.9850504778296, 2813.68505047783,
2814.0988269188797, 2821.77435573988, 2824.38725896588]
```

In [522]:

```
Qc = sum(H)+Qh #Qc is ending value of cacade after adding Qc
print(Qc)

pinch = 0
count = 0
recheckR = len(columntempsR)*[0]
recheckC = len(columntempsC)*[0]

Q = Qh
for i in Hlen: #finds pinch by checking for zero value in cascade
    Q = Q + H[i]
    if -.01 < Q < .01:
        pinch = Tstar[i+1-count]

    for j in range(len(columntempsR)):
        if Tstar[i+1-count] == columntempsR[j][0]:
            if recheckR[j] == 0:
                count = count + 1
                recheckR[j] = 1
            else:
                recheckR[j] = 0
    for j in range(len(columntempsC)):
        if Tstar[i+1-count] == columntempsC[j][0]:
            if recheckC[j] == 0:
                count = count + 1
                recheckC[j] = 1
            else:
                recheckC[j] = 0

print(pinch) #pinch temperature
```

```
2824.38725896588
254.3
```

In [523]:

```
#COLUMN INTEGRATION (Tmin of 20)

CP = [2.439565742, 2.441707317, 1.231428571, 0.5225806452, 0.6550724638, 0.06, 1.150300601, 3.25872
093, 0.1176995096, 0.1272727273, 0,0, 0, 0, 0, 0] #input CP values (0 for columns)

Ts = [35.5, 450, 40, 35.5, 104.5, 70, 129.9, 183.2, 249.3, 80, 35.5, 129.9, 104.5, 150.3, 183.2, 24
9.3] #input temperatures at inlets

Tt = [450, 40, 75, 20, 70, 35, 80, 80, 25, 25, 35.5, 129.9, 104.5, 150.3, 183.2, 249.3] #input
temperatures at outlets

Qcol = [0,0,0,0,0,0,0,0,0,0, 35.7, -95, 186.8, -286.5, 2552.7, -2832] #input 0 if not a column, -Qre
b for reboiler, or Qcond for condenser

Tmin = 20 #input Tmin

o = list(range(len(CP))) #list of length of the number of streams
Tss = list(range(len(Ts))) #list that will become shifted inlet temperatures
Tts = list(range(len(Tt))) #list that will become shifted outlet temperatures
```

xiv. Pinch Calculations at Tmin of 20 degrees C Code

```
print(Qh) #Qh
Hnewsum = list(range(len(Hsum))) #***HEAT CASCADE Hs***

for i in Hlen:
    Hnewsum[i] = Hsum[i] + Qh
print()
print(Hnewsum) #Heat Cascade after Qh
```

```
2855.9872590675

[2831.5916016475, 2832.0, 0.0, 0.0214157499999601, 7.942911442059994, 2560.64291144206,
2638.0119815763996, 2351.5119815763996, 2420.43464667424, 2325.43464667424, 2359.22026682024,
2474.25337725648, 2661.05337725648, 2762.1401113047796, 2781.90264384678, 2778.61022089378,
2760.12392571178, 2761.50593786533, 2776.61267541178, 2777.9850504778296, 2813.68505047783,
2814.0988269188797, 2821.77435573988, 2824.38725896588]
```

In [522]:

```
Qc = sum(H)+Qh #Qc is ending value of cacade after adding Qc
print(Qc)

pinch = 0
count = 0
recheckR = len(columntempsR)*[0]
recheckC = len(columntempsC)*[0]

Q = Qh
for i in Hlen: #finds pinch by checking for zero value in cascade
    Q = Q + H[i]
    if -.01 < Q < .01:
        pinch = Tstar[i+1-count]

    for j in range(len(columntempsR)):
        if Tstar[i+1-count] == columntempsR[j][0]:
            if recheckR[j] == 0:
                count = count + 1
                recheckR[j] = 1
            else:
                recheckR[j] = 0
    for j in range(len(columntempsC)):
        if Tstar[i+1-count] == columntempsC[j][0]:
            if recheckC[j] == 0:
                count = count + 1
                recheckC[j] = 1
            else:
                recheckC[j] = 0

print(pinch) #pinch temperature
```

```
2824.38725896588
254.3
```

In [523]:

```
#COLUMN INTEGRATION (Tmin of 20)

CP = [2.439565742, 2.441707317, 1.231428571, 0.5225806452, 0.6550724638, 0.06, 1.150300601, 3.25872
093, 0.1176995096, 0.1272727273, 0,0, 0, 0, 0, 0] #input CP values (0 for columns)

Ts = [35.5, 450, 40, 35.5, 104.5, 70, 129.9, 183.2, 249.3, 80, 35.5, 129.9, 104.5, 150.3, 183.2, 24
9.3] #input temperatures at inlets

Tt = [450, 40, 75, 20, 70, 35, 80, 80, 25, 25, 35.5, 129.9, 104.5, 150.3, 183.2, 249.3] #input
temperatures at outlets

Qcol = [0,0,0,0,0,0,0,0,0,0, 35.7, -95, 186.8, -286.5, 2552.7, -2832] #input 0 if not a column, -Qre
b for reboiler, or Qcond for condenser

Tmin = 20 #input Tmin

o = list(range(len(CP))) #list of length of the number of streams
Tss = list(range(len(Ts))) #list that will become shifted inlet temperatures
Tts = list(range(len(Tt))) #list that will become shifted outlet temperatures
```

```
coldnum = [] #list used to identify cold streams
hotnum = [] #list used to identify hot streams
disnumC = [] #list used to identify condensers
disnumR = [] #list used to identify reboilers
columntempsC = [] #used to identify where the condenders fit into the heat cascade
columntempsR = [] #used to identify where the reboilers fit into the heat cascade

for i in o: #function creating modified temps and identifying hot vs cold streams
    if Ts[i] > Tt[i]:
        Tss[i] = Ts[i]-(Tmin*.5)
        Tts[i] = Tt[i]-(Tmin*.5)
        hotnum.append(i)

    if Ts[i] < Tt[i]:
        Tss[i] = Ts[i]+(Tmin*.5)
        Tts[i] = Tt[i]+(Tmin*.5)
        coldnum.append(i)

    if Ts[i] == Tt[i]: #identifies columns/shifts temps
        if Qcol[i] > 0:
            Tss[i] = Ts[i]-(Tmin*.5)
            Tts[i] = Tt[i]-(Tmin*.5)
            disnumC.append(i)
            columntempsC.append([Tts[i], Qcol[i]])
        if Qcol[i] < 0:
            Tss[i] = Ts[i]+(Tmin*.5)
            Tts[i] = Tt[i]+(Tmin*.5)
            disnumR.append(i)
            columntempsR.append([Tts[i], Qcol[i]])
```

In [524]:

```
Tall = [] #list of all shifted temperatures
for i in o:
    Tall.append(Tss[i])
    Tall.append(Tts[i])

Tstar = list(set(Tall)) #list of all temperatures, without repeats, in descending order
Tstar.sort(reverse=True)
print(Tstar) #HEAT CASCADE TEMPS

print()

CPtot = [] #will be list of net CPs for each temp interval
lenT = list(range(len(Tstar) - 1)) #list for iterating through temperature intervals
p=0
lentempc = list(range(len(coldnum))) #iterates over cold streams
lentemph = list(range(len(hotnum))) #iterates over hot streams

#function to calculate net CPs
for i in lenT: #iterate through temp intervals
    p=0
    for j in o: #iterate through streams
        for k in lentempc:
            if j == coldnum[k]: #if cold, increase temp in integer increments to see
                                #if it's in the selected temp range
                x = Tss[j]
                while Tts[j] >= x:
                    if -.005 +x <Tstar[i] -.1 < x +.005:
                        p = p-CP[j]
                    x = x+.1
        for k in lentemph: #same for hot streams, but decreasing temp
            if j == hotnum[k]:
                x = Tss[j]
                while Tts[j] <= x:
                    if -.005 + x<Tstar[i] -.1 <x+.005:
                        p = p+CP[j]
                    x = x-.1
    CPtot.append(p)
print(CPtot) #NET CP VALUES
```

```
[460.0, 440.0, 259.3, 239.3, 173.2, 160.3, 139.9, 119.9, 94.5, 85.0, 70.0, 60.0, 50.0, 45.5, 30.0,
25.5, 25.0, 15.0, 10.0]

[-2.439565742, 0.0021415750000000067, 0.0021415750000000067, 0.11984108460000001, 3.3785620146, 3.
```

```
3785620146, 3.3785620146, 4.5288626156000005, 5.1839350794, 3.9525065084, -0.3292422953, -
0.9243147590999998, 0.3071138119, 2.7466795539, 0.30497223689999997, 0.8275528821, 0.7675528821, 0.
5225806452]
```

In [525]:

```
H = [] #list of delta H values for intervals
Hval = list(range((len(Tstar) - 1))) #list to iterate through temperature intervals

for i in Hval:
    H.append((Tstar[i]-Tstar[i+1])*(CPtot[i])) #calculates delta H

print(H) #delta Hs before column integration
print()

count = 0
for i in range(len(Tstar)): #adds in column values
    for j in range(len(columntempsR)):
        if Tstar[i] == columntempsR[j][0]:
            H.insert(count + i, columntempsR[j][1])
            count = count+1
    for j in range(len(columntempsC)):
        if Tstar[i] == columntempsC[j][0]:
            H.insert(count + i, columntempsC[j][1])
            count = count+1

print(H) #delta Hs after column integration
```

```
[-48.79131484, 0.38698260250000116, 0.04283150000000013, 7.921495692060003, 43.58344998833992,
68.92266509784002, 67.571240292, 115.03311043624004, 49.247383254300004, 59.287597626, -
3.292422953, -9.243147590999998, 1.3820121535499998, 42.57353308545, 1.3723750660499998,
0.41377644105, 7.6755288209999994, 2.6129032260000002]

[-48.79131484, 0.38698260250000116, -2832, 0.04283150000000013, 7.921495692060003, 2552.7,
43.58344998833992, -286.5, 68.92266509784002, -95, 67.571240292, 115.03311043624004, 186.8,
49.247383254300004, 59.287597626, -3.292422953, -9.243147590999998, 1.3820121535499998,
42.57353308545, 1.3723750660499998, 35.7, 0.41377644105, 7.6755288209999994, 2.6129032260000002]
```

In [526]:

```
Hlen = list(range(len(H))) #list of delta H values for intervals

Hsum = []
n = 0
for i in Hlen:
    n = n+H[i]
    Hsum.append(n)
Qh = -min(Hsum) #Qh is -largest negative number when adding through H

print(Qh) #Qh
Hnewsum = list(range(len(Hsum))) #***HEAT CASCADE Hs***

for i in Hlen:
    Hnewsum[i] = Hsum[i] + Qh
print()
print(Hnewsum) #Heat Cascade after Qh
```

```
2880.4043322375

[2831.6130173975002, 2832.0, 0.0, 0.0428314999999202, 7.9643271920599545, 2560.66432719206,
2604.2477771803997, 2317.7477771803997, 2386.6704422782395, 2291.6704422782395, 2359.24168257024,
2474.2747930064797, 2661.07479300648, 2710.32217626078, 2769.6097738867797, 2766.31735093378,
2757.0742033427796, 2758.4562154963296, 2801.0297485817796, 2802.4021236478297, 2838.10212364783,
2838.51590008888, 2846.19142890988, 2848.8043321358796]
```

In [527]:

```
Qc = sum(H)+Qh #Qc is ending value of cacade after adding Qc
print(Qc)

pinch = 0
count = 0
recheckR = len(columntempsR)*[0]
```

```
recheckC = len(columntempsC)*[0]

Q = Qh
for i in Hlen: #finds pinch by checking for zero value in cascade
    Q = Q + H[i]
    if -.01 < Q < .01:
        pinch = Tstar[i+1-count]

    for j in range(len(columntempsR)):
        if Tstar[i+1-count] == columntempsR[j][0]:
            if recheckR[j] == 0:
                count = count + 1
                recheckR[j] = 1
            else:
                recheckR[j] = 0
    for j in range(len(columntempsC)):
        if Tstar[i+1-count] == columntempsC[j][0]:
            if recheckC[j] == 0:
                count = count + 1
                recheckC[j] = 1
            else:
                recheckC[j] = 0

print(pinch) #pinch temperature
```

```
2848.8043321358796
259.3
```

In [485]:

```
def HXColInt(T): #uses program from before, but have Tmin as an input, and spits out
                #Qh, Qc, and Tpinch for each value put in
    CP = [24.39565742, 24.41707317, 12.31428571, 5.225806452, 6.550724638, 0.6, 11.50300601,
          32.5872093, 1.176995096, 1.272727273, 0, 0, 0, 0, 0, 0] #input CP values (0 for columns)

    Ts = [35.5, 450, 40, 35.5, 104.5, 70, 129.9, 183.2, 249.3, 80, 35.5, 129.9, 104.5, 150.3,
          183.2, 249.3] #input temperatures at inlets

    Tt = [450, 40, 75, 20, 70, 35, 80, 80, 25, 25, 35.5, 129.9, 104.5, 150.3, 183.2, 249.3]
            #input temperatures at outlets

    Qcol = [0,0,0,0,0,0,0,0,0,0, 357, -950, 1868, -2865, 25527, -28320]
            #input 0 if not a column, -Qreb for reboiler, or Qcond for condenser

    Tmin = T #input Tmin

    o = list(range(len(CP))) #list of length of the number of streams
    Tss = list(range(len(Ts))) #list that will become shifted inlet temperatures
    Tts = list(range(len(Tt))) #list that will become shifted outlet temperatures

    coldnum = [] #list used to identify cold streams
    hotnum = [] #list used to identify hot streams
    disnumC = [] #list used to identify condensers
    disnumR = [] #list used to identify reboilers
    columntempsC = [] #used to identify where the condenders fit into the heat cascade
    columntempsR = [] #used to identify where the reboilers fit into the heat cascade

    for i in o: #function creating modified temps and identifying hot vs cold streams
        if Ts[i] > Tt[i]:
            Tss[i] = Ts[i]-(Tmin*.5)
            Tts[i] = Tt[i]-(Tmin*.5)
            hotnum.append(i)

        if Ts[i] < Tt[i]:
            Tss[i] = Ts[i]+(Tmin*.5)
            Tts[i] = Tt[i]+(Tmin*.5)
            coldnum.append(i)

        if Ts[i] == Tt[i]: #identifies columns/shifts temps
            if Qcol[i] > 0:
                Tss[i] = Ts[i]-(Tmin*.5)
                Tts[i] = Tt[i]-(Tmin*.5)
                disnumC.append(i)
                columntempsC.append([Tts[i], Qcol[i]])
            if Qcol[i] < 0:
                Tss[i] = Ts[i]+(Tmin*.5)
```

xv. Cost Analysis Code

```
recheckC = len(columntempsC)*[0]

Q = Qh
for i in Hlen: #finds pinch by checking for zero value in cascade
    Q = Q + H[i]
    if -.01 < Q < .01:
        pinch = Tstar[i+1-count]

    for j in range(len(columntempsR)):
        if Tstar[i+1-count] == columntempsR[j][0]:
            if recheckR[j] == 0:
                count = count + 1
                recheckR[j] = 1
            else:
                recheckR[j] = 0
    for j in range(len(columntempsC)):
        if Tstar[i+1-count] == columntempsC[j][0]:
            if recheckC[j] == 0:
                count = count + 1
                recheckC[j] = 1
            else:
                recheckC[j] = 0

print(pinch) #pinch temperature
```

```
2848.8043321358796
259.3
```

In [485]:

```
def HXColInt(T): #uses program from before, but have Tmin as an input, and spits out
                #Qh, Qc, and Tpinch for each value put in
    CP = [24.39565742, 24.41707317, 12.31428571, 5.225806452, 6.550724638, 0.6, 11.50300601,
          32.5872093, 1.176995096, 1.272727273, 0, 0, 0, 0, 0, 0] #input CP values (0 for columns)

    Ts = [35.5, 450, 40, 35.5, 104.5, 70, 129.9, 183.2, 249.3, 80, 35.5, 129.9, 104.5, 150.3,
          183.2, 249.3] #input temperatures at inlets

    Tt = [450, 40, 75, 20, 70, 35, 80, 80, 25, 25, 35.5, 129.9, 104.5, 150.3, 183.2, 249.3]
            #input temperatures at outlets

    Qcol = [0,0,0,0,0,0,0,0,0,0, 357, -950, 1868, -2865, 25527, -28320]
            #input 0 if not a column, -Qreb for reboiler, or Qcond for condenser

    Tmin = T #input Tmin

    o = list(range(len(CP))) #list of length of the number of streams
    Tss = list(range(len(Ts))) #list that will become shifted inlet temperatures
    Tts = list(range(len(Tt))) #list that will become shifted outlet temperatures

    coldnum = [] #list used to identify cold streams
    hotnum = [] #list used to identify hot streams
    disnumC = [] #list used to identify condensers
    disnumR = [] #list used to identify reboilers
    columntempsC = [] #used to identify where the condenders fit into the heat cascade
    columntempsR = [] #used to identify where the reboilers fit into the heat cascade

    for i in o: #function creating modified temps and identifying hot vs cold streams
        if Ts[i] > Tt[i]:
            Tss[i] = Ts[i]-(Tmin*.5)
            Tts[i] = Tt[i]-(Tmin*.5)
            hotnum.append(i)

        if Ts[i] < Tt[i]:
            Tss[i] = Ts[i]+(Tmin*.5)
            Tts[i] = Tt[i]+(Tmin*.5)
            coldnum.append(i)

        if Ts[i] == Tt[i]: #identifies columns/shifts temps
            if Qcol[i] > 0:
                Tss[i] = Ts[i]-(Tmin*.5)
                Tts[i] = Tt[i]-(Tmin*.5)
                disnumC.append(i)
                columntempsC.append([Tts[i], Qcol[i]])
            if Qcol[i] < 0:
                Tss[i] = Ts[i]+(Tmin*.5)
```

```
            Tss[i] = Ts[i]+(Tmin*.5)
            Tts[i] = Tt[i]+(Tmin*.5)
            disnumR.append(i)
            columntempsR.append([Tts[i], Qcol[i]])

Tall = [] #list of all shifted temperatures
for i in o:
    Tall.append(Tss[i])
    Tall.append(Tts[i])

Tstar = list(set(Tall)) #list of all temperatures, without repeats, in descending order
Tstar.sort(reverse=True) #***HEAT CASCADE TEMPS***

CPtot = [] #will be list of net CPs for each temp interval
lenT = list(range(len(Tstar) - 1)) #list for iterating through temperature intervals
p=0
lentempc = list(range(len(coldnum))) #iterates over cold streams
lentemph = list(range(len(hotnum))) #iterates over hot streams

#function to calculate net CPs
for i in lenT: #iterate through temp intervals
    p=0
    for j in o: #iterate through streams
        for k in lentempc:
            if j == coldnum[k]: #if cold, checks if terget temp is in range
                if Tss[j] <Tstar[i] -.01 < Tts[j]:
                    p = p-CP[j]
        for k in lentemph: #same for hot streams w/ target temp
            if j == hotnum[k]:
                if Tts[j] <Tstar[i] -.01 < Tss[j]:
                    p = p+CP[j]
    CPtot.append(p)

H = [] #list of delta H values for intervals
Hval = list(range((len(Tstar) - 1))) #list to iterate through temperature intervals

for i in Hval:
    H.append((Tstar[i]-Tstar[i+1])*(CPtot[i])) #calculates delta H

count = 0
for i in range(len(Tstar)): #adds in column values
    for j in range(len(columntempsR)):
        if Tstar[i] == columntempsR[j][0]:
            H.insert(count + i, columntempsR[j][1])
            count = count+1
    for j in range(len(columntempsC)):
        if Tstar[i] == columntempsC[j][0]:
            H.insert(count + i, columntempsC[j][1])
            count = count+1

Hlen = list(range(len(H))) #list of delta H values for intervals

Hsum = []
n = 0
for i in Hlen:
    n = n+H[i]
    Hsum.append(n)
Qh = -min(Hsum) #Qh is -largest negative number when adding through H
QH.append(Qh)  #Adds Qh to list
Hnewsum = list(range(len(Hsum))) #***HEAT CASCADE Hs***

for i in Hlen:
    Hnewsum[i] = Hsum[i] + Qh

Qc = sum(H)+Qh #Qc is ending value of cacade after adding Qc
QC.append(Qc) #Adds Qc to list
pinch = 0
count = 0
recheckR = len(columntempsR)*[0]
recheckC = len(columntempsC)*[0]

Q = Qh
for i in Hlen: #finds pinch by checking for zero value in cascade
    Q = Q + H[i]
    if -.01 < Q < .01:
        pinch = Tstar[i+1-count]

    for i in range(len(columntempsR)):
```

```
        for j in range(len(columntempsR)):
            if Tstar[i+1-count] == columntempsR[j][0]:
                if recheckR[j] == 0:
                    count = count + 1
                    recheckR[j] = 1
                else:
                    recheckR[j] = 0
        for j in range(len(columntempsC)):
            if Tstar[i+1-count] == columntempsC[j][0]:
                if recheckC[j] == 0:
                    count = count + 1
                    recheckC[j] = 1
                else:
                    recheckC[j] = 0
    Tpinch.append(pinch) #adds Tpinch to list
    return
```

In [489]:

```
QH = [] #this code runs through Tmin values from 10 to 40 in intervals of .001 and the program
        #just above to get Qh, Qc, and Tpinch for each Tmin
QC = []
Tpinch = []

Tmins = list(range(30001))
for i in Tmins:
    Tmins[i] = Tmins[i]/1000
    Tmins[i] = Tmins[i]+10

for i in list(range(30001)): #takes a while to run due to number of iterations
    HXColInt(Tmins[i])
```

In [491]:

```
hrs = 300*24 #setting hrs equal to hours in a year
print(hrs)
```

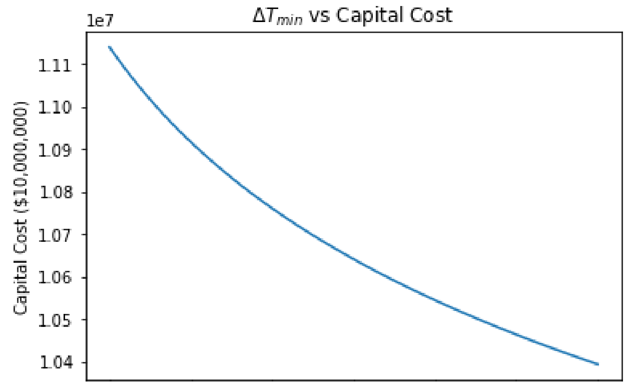
```
7200
```

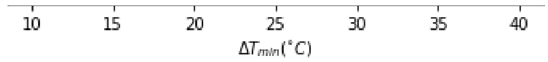
In [492]:

```
cost = [] #calculating all three costs in terms of Tmin
Ccost = []
Tcost = []
for i in list(range(30001)):
    cost.append(QH[i]*hrs*.15)
    Ccost.append((12.5*10**6)/(Tmins[i]**.05))
    Tcost.append(cost[i]+Ccost[i])
```

In [493]:

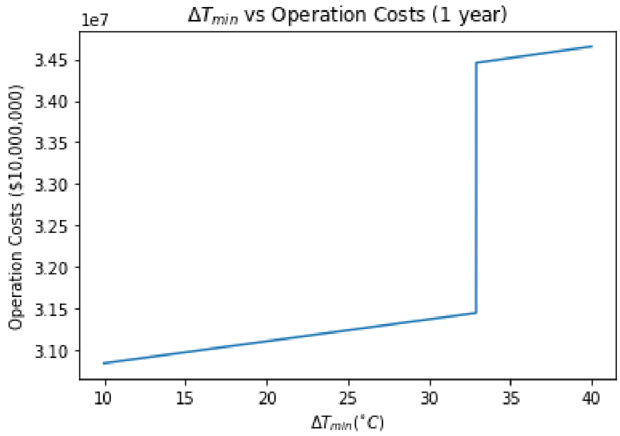
```
import matplotlib.pyplot as plt #Graphing Tmin vs Capital cost
plt.plot(Tmins, Ccost)
plt.title('$\Delta T_{min}$ vs Capital Cost')
plt.xlabel('$\Delta T_{min} (^{\circ}C)$')
plt.ylabel('Capital Cost ($10,000,000)')
plt.show()
```





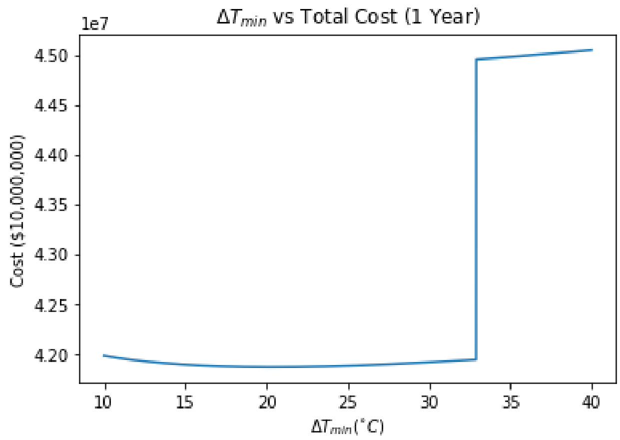
In [494]:

```
import matplotlib.pyplot as plt #graphing Tmin vs operation costs
plt.plot(Tmins, cost)
plt.title('$\Delta T_{min}$ vs Operation Costs (1 year)')
plt.xlabel('$\Delta T_{min} (^{\circ}C)$')
plt.ylabel('Operation Costs ($10,000,000)')
plt.show()
```



In [495]:

```
import matplotlib.pyplot as plt #graphing Tmin vs total cost for 1 year
plt.plot(Tmins, Tcost)
plt.title('$\Delta T_{min}$ vs Total Cost (1 Year)')
plt.xlabel('$\Delta T_{min} (^{\circ}C)$')
plt.ylabel('Cost ($10,000,000)')
plt.show()
```



In [357]:

```
minimum = min(Tcost) #calculating minimum cost for 1 year, and the Tmin needed to achieve this
for i in list(range(30001)):
    if minimum == Tcost[i]:
        print(Tcost[i])
        print(Tmins[i])
```

```
41869410.86540445
20.384
```

In [471]:

```
hours = list(range(360*5)) #calculation of the number of days until the best Tmin value is 10
degrees
for i in hours:
    hours[i] = i*24
for i in range(len(hours)):
```

```
for i in range(len(hours)):
    num = (QH[0]*.15*hours[i]) + ((12.5*10**6)/(Tmins[0]**.05))
    num2 = (QH[1]*.15*hours[i]) + ((12.5*10**6)/(Tmins[1]**.05))
    if num<num2:
        days = i
        break

print(i)
```

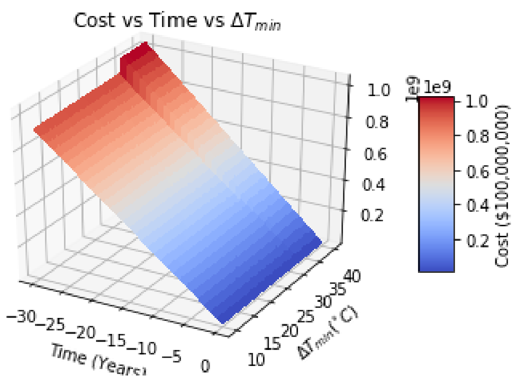
634

In [496]:

```
hours = [] #calculating cost in terms of Tmin and time
for i in range(31):
    hours.append(i*7200)
cost = list(range(31))
Tcost = list(range(31))
for i in cost:
    cost[i] = list(range(len(hours)))
    Tcost[i] = list(range(len(hours)))
Ccost = []
for i in list(range(31)):
    for j in range(31):
        cost[i][j] = (QH[i]*hours[j]*.15)
    Ccost.append((12.5*10**6)/(Tmins[i]**.05))
    for j in range(31):
        Tcost[i][j] = (cost[i][j]+Ccost[i])
```

In [475]:

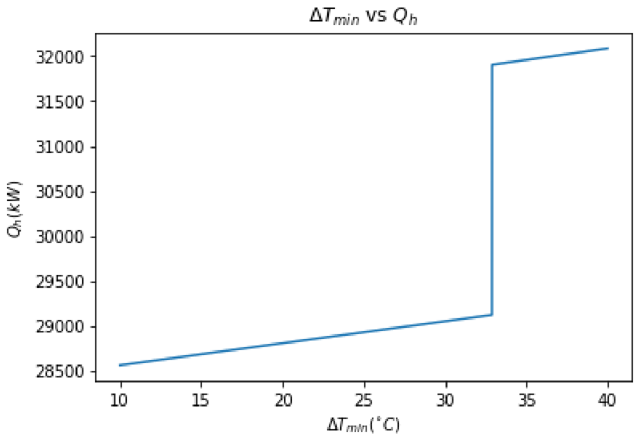
```
import numpy as np #using the relations between cost, Tmin, and time, makes a 3D plot to show
these relationships
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

xthree1 = np.array(Tmins)
ythree1 = np.array(hours)
Qthree = np.array(QH)
xthree1, ythree1 = np.meshgrid(xthree1, ythree1)
zthree1 = ((12.5*10**6)/(xthree1)) + (Qthree*ythree1*.15)

fig1 = plt.figure() #here we create our 3d coordinate system
ax1 = fig1.gca(projection='3d')

three1 = ax1.plot_surface(-ythree1/7200, xthree1, zthree1, cmap='coolwarm', linewidth=0, antialiase
d = False) #creates our 3d plot as a surface plot.
fig1.colorbar(three1, shrink=0.5, aspect=5, label = 'Cost ($100,000,000)') #creates a bar to displa
y what z values each color cooresponds to
plt.xlabel('Time (Years)')
plt.ylabel('$\Delta T_{min} (^{\circ}C)$')
plt.title('Cost vs Time vs $\Delta T_{min}$')
plt.show()
```



In [453]:

```
plt.plot(Tmins, QH) #Plots Tmin vs Qh
plt.title('$\Delta T_{min}$ vs $Q_h$')
```

```
plt.xlabel('$\Delta T_{min} (^{\circ}C)$')
plt.ylabel('$Q_h (kW)$')
plt.show()
```

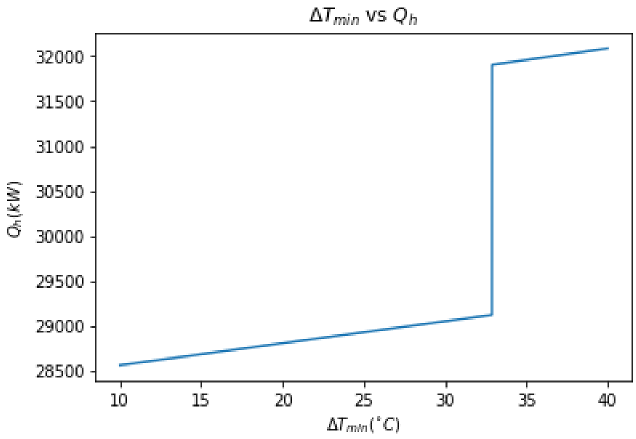


In [360]:

```
import networkx as nx #don't use!
G = nx.Graph()

G.add_node('1', node_color= '#f20d0d')
G.add_node('2', node_color= '#f2690d')
G.add_node('3', node_color= '#f2ba0d')
G.add_node('4', node_color= '#d9f20d')
G.add_node('5a', node_color= '#5af20d')
G.add_node('5b', node_color= '#0df23f')
G.add_node('6', node_color= '#0df279')
G.add_node('7', node_color= '#0df2b3')
G.add_node('8', node_color= '#0df2ec')
G.add_node('9', node_color= '#0d79f2')
G.add_node('C1', node_color= '#0d28f2')
G.add_node('R1', node_color= '#2c0df2')
G.add_node('C2', node_color= '#710df2')
G.add_node('R2', node_color= '#d90df2')
G.add_node('C3', node_color= '#f20d98')
G.add_node('R3', node_color= '#f20d52')

G.add_edge('1', '2', weight=465.2252+521.58)
G.add_edge('2', 'R3', weight=.16726)
G.add_edge('3', '7', weight=43.1)
G.add_edge('7', 'R1', weight=74.654)
G.add_edge('7', 'R2', weight=66.504)
G.add_edge('8', 'R1', weight=10.4753)
G.add_edge('8', 'R2', weight=2.4)
G.add_edge('R1', 'C3', weight=26.1)
G.add_edge('R2', 'C3', weight=201.37)

color_map = []
for node in G:
    if node == '1':
        color_map.append('#f20d0d')
    if node == '2':
        color_map.append('#f2690d')
    if node == '3':
        color_map.append('#f2ba0d')
    if node == '4':
        color_map.append('#ecee0a')
    if node == '5a':
        color_map.append('#5af20d')
    if node == '5b':
        color_map.append('#0df23f')
    if node == '6':
        color_map.append('#0df279')
    if node == '7':
        color_map.append('#0df2b3')
    if node == '8':
        color_map.append('#0df2ec')
    if node == '9':
        color_map.append('#0d79f2')
    if node == 'C1':
```

xvi. Eigenvector Centrality for Original Network Code

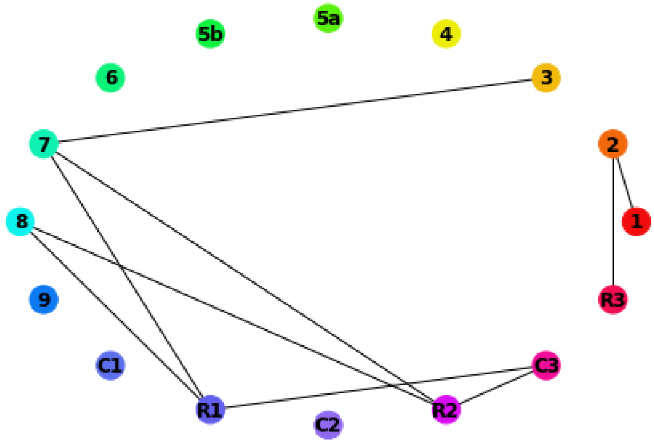
```
plt.xlabel('$\Delta T_{min} (^{\circ}C)$')
plt.ylabel('$Q_h (kW)$')
plt.show()
```



In [360]:

```
import networkx as nx #don't use!
G = nx.Graph()

G.add_node('1', node_color= '#f20d0d')
G.add_node('2', node_color= '#f2690d')
G.add_node('3', node_color= '#f2ba0d')
G.add_node('4', node_color= '#d9f20d')
G.add_node('5a', node_color= '#5af20d')
G.add_node('5b', node_color= '#0df23f')
G.add_node('6', node_color= '#0df279')
G.add_node('7', node_color= '#0df2b3')
G.add_node('8', node_color= '#0df2ec')
G.add_node('9', node_color= '#0d79f2')
G.add_node('C1', node_color= '#0d28f2')
G.add_node('R1', node_color= '#2c0df2')
G.add_node('C2', node_color= '#710df2')
G.add_node('R2', node_color= '#d90df2')
G.add_node('C3', node_color= '#f20d98')
G.add_node('R3', node_color= '#f20d52')

G.add_edge('1', '2', weight=465.2252+521.58)
G.add_edge('2', 'R3', weight=.16726)
G.add_edge('3', '7', weight=43.1)
G.add_edge('7', 'R1', weight=74.654)
G.add_edge('7', 'R2', weight=66.504)
G.add_edge('8', 'R1', weight=10.4753)
G.add_edge('8', 'R2', weight=2.4)
G.add_edge('R1', 'C3', weight=26.1)
G.add_edge('R2', 'C3', weight=201.37)

color_map = []
for node in G:
    if node == '1':
        color_map.append('#f20d0d')
    if node == '2':
        color_map.append('#f2690d')
    if node == '3':
        color_map.append('#f2ba0d')
    if node == '4':
        color_map.append('#ecee0a')
    if node == '5a':
        color_map.append('#5af20d')
    if node == '5b':
        color_map.append('#0df23f')
    if node == '6':
        color_map.append('#0df279')
    if node == '7':
        color_map.append('#0df2b3')
    if node == '8':
        color_map.append('#0df2ec')
    if node == '9':
        color_map.append('#0d79f2')
    if node == 'C1':
```

```
        color_map.append('#5e6fee')
    if node == 'R1':
        color_map.append('#635eee')
    if node == 'C2':
        color_map.append('#906aee')
    if node == 'R2':
        color_map.append('#d90df2')
    if node == 'C3':
        color_map.append('#f20d98')
    if node == 'R3':
        color_map.append('#f20d52')


import matplotlib.pyplot as plt
nx.draw_circular(G,node_color = color_map,with_labels = True, font_weight='bold')
plt.show()

EC = nx.eigenvector_centrality(G)
print(EC)
```

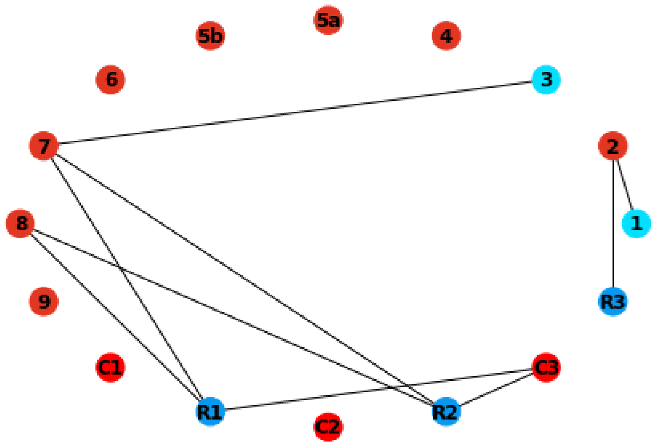



```
{'1': 9.046110461023596e-06, '2': 1.27931321007047e-05, '3': 0.18000813939677104, '4': 2.030074636709603e-16, '5a': 2.030074636709603e-16, '5b': 2.030074636709603e-16, '6': 2.030074636709603e-16, '7': 0.45440134941833504, '8': 0.38309229533177, '9': 2.030074636709603e-16, 'C1': 2.030074636709603e-16, 'R1': 0.483527181027733, 'C2': 2.030074636709603e-16, 'R2': 0.483527181027733, 'C3': 0.38309229533177, 'R3': 9.046110461023596e-06}
```

In [498]:

```
import networkx as nx #creates graph
G = nx.Graph()

G.add_node('1')
G.add_node('2')
G.add_node('3')
G.add_node('4')
G.add_node('5a')
G.add_node('5b')
G.add_node('6')
G.add_node('7')
G.add_node('8')
G.add_node('9')
G.add_node('C1')
G.add_node('R1')
G.add_node('C2')
G.add_node('R2')
G.add_node('C3')
G.add_node('R3')

G.add_edge('1', '2', weight=465.2252+521.58) #creates edges and edge weights
G.add_edge('2', 'R3', weight=.16726)
G.add_edge('3', '7', weight=43.1)
G.add_edge('7', 'R1', weight=74.654)
G.add_edge('7', 'R2', weight=66.504)
G.add_edge('8', 'R1', weight=10.4753)
G.add_edge('8', 'R2', weight=2.4)
G.add_edge('R1', 'C3', weight=26.1)
```

```python
G.add_edge('R2', 'C3', weight=201.37)

color_map = [] #colors the nodes
for node in G:
    if node == '1':
        color_map.append('#05dfff')
    if node == '2':
        color_map.append('#e13622')
    if node == '3':
        color_map.append('#05dfff')
    if node == '4':
        color_map.append('#e13622')
    if node == '5a':
        color_map.append('#e13622')
    if node == '5b':
        color_map.append('#e13622')
    if node == '6':
        color_map.append('#e13622')
    if node == '7':
        color_map.append('#e13622')
    if node == '8':
        color_map.append('#e13622')
    if node == '9':
        color_map.append('#e13622')
    if node == 'C1':
        color_map.append('#fd0000')
    if node == 'R1':
        color_map.append('#0998f0')
    if node == 'C2':
        color_map.append('#fd0000')
    if node == 'R2':
        color_map.append('#0998f0')
    if node == 'C3':
        color_map.append('#fd0000')
    if node == 'R3':
        color_map.append('#0998f0')


nx.draw_circular(G,node_color = color_map,with_labels = True, font_weight='bold') #draws our graph
plt.show()

EC = nx.eigenvector_centrality(G) #prints unweighted centralities
print(EC)
print()

ECW = nx.eigenvector_centrality(G, max_iter=100, tol=1e-04, weight = 'weight') #prints weighted cen
tralities
print(ECW)
```




```
{'1': 9.046110461023596e-06, '2': 1.27931321007047e-05, '3': 0.18000813939677104, '4':
2.030074636709603e-16, '5a': 2.030074636709603e-16, '5b': 2.030074636709603e-16, '6':
2.030074636709603e-16, '7': 0.45440134941833504, '8': 0.38309229533177, '9': 2.030074636709603e-16
, 'C1': 2.030074636709603e-16, 'R1': 0.483527181027733, 'C2': 2.030074636709603e-16, 'R2':
0.483527181027733, 'C3': 0.38309229533177, 'R3': 9.046110461023596e-06}

{'1': 0.7071659513667207, '2': 0.707047578120793, '3': 8.609888467230922e-06, '4':
7.610624739699129e-19, '5a': 7.610624739699129e-19, '5b': 7.610624739699129e-19, '6':
```

```
7.610624739699129e-19, '7': 4.48381644426463e-05, '8': 2.554859928484192e-06, '9':
7.610624739699129e-19, 'C1': 7.610624739699129e-19, 'R1': 2.7112364662527182e-05, 'C2':
7.610624739699129e-19, 'R2': 0.00010650607344006016, 'C3': 0.00010471126102060963, 'R3': 0.00011986
213391011565}
```

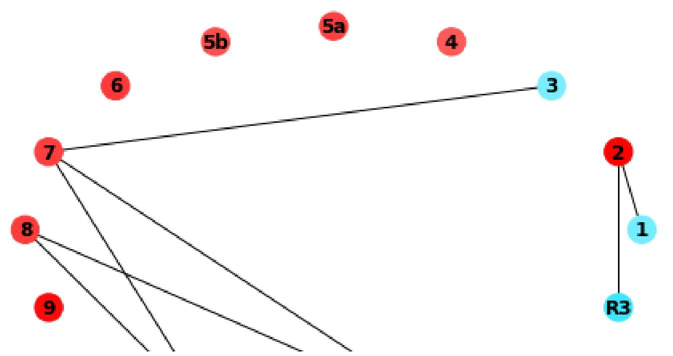
In [362]:

```python
print(['%s %0.4f'%(node,EC[node]) for node in EC]) #Prints centrality and weighted centrality to 4 decimal places
print()
print(['%s %0.4f'%(node,ECW[node]) for node in ECW])
```

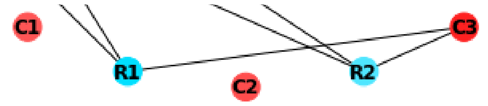
```
['1 0.0000', '2 0.0000', '3 0.1800', '4 0.0000', '5a 0.0000', '5b 0.0000', '6 0.0000', '7 0.4544',
'8 0.3831', '9 0.0000', 'C1 0.0000', 'R1 0.4835', 'C2 0.0000', 'R2 0.4835', 'C3 0.3831', 'R3 0.000
0']

['1 0.7072', '2 0.7070', '3 0.0000', '4 0.0000', '5a 0.0000', '5b 0.0000', '6 0.0000', '7 0.0000',
'8 0.0000', '9 0.0000', 'C1 0.0000', 'R1 0.0000', 'C2 0.0000', 'R2 0.0001', 'C3 0.0001', 'R3 0.000
1']
```

In [257]:

```python
color_map = [] #not used!
for node in G:
    if node == '1':
        color_map.append('#75edff')
    if node == '2':
        color_map.append('#ff0000')
    if node == '3':
        color_map.append('#79eeff')
    if node == '4':
        color_map.append('#ff5858')
    if node == '5a':
        color_map.append('#ff4040')
    if node == '5b':
        color_map.append('#ff4e4e')
    if node == '6':
        color_map.append('#ff3939')
    if node == '7':
        color_map.append('#ff3e3e')
    if node == '8':
        color_map.append('#ff3c3c')
    if node == '9':
        color_map.append('#ff0a0a')
    if node == 'C1':
        color_map.append('#ff5555')
    if node == 'R1':
        color_map.append('#00dfff')
    if node == 'C2':
        color_map.append('#ff4d4d')
    if node == 'R2':
        color_map.append('#4fe6ff')
    if node == 'C3':
        color_map.append('#ff2121')
    if node == 'R3':
        color_map.append('#3ee5ff')

nx.draw_circular(G,node_color = color_map,with_labels = True, font_weight='bold')
plt.show()
```



xvii. Pinch Calculations for Steam Raising Code



In [499]:

```
#STEAM RAISING NETWORK
#Very similar to earlier code, but not with 2 more streams and a
#"reboiler" which is our vaporization stream
CP = [2.439565742, 2.441707317, 1.231428571, 0.5225806452, 0.6550724638, 0.06, 1.150300601,
      3.25872093, 0.1176995096, 0.1272727273, 0, 0, 0, 0, 0, 0, 4.184*1.064965, 1.996*1.064965,
      ((4.184+1.996)/2)*1.064965] #input CP values (0 for columns)

Ts = [35.5, 450, 40, 35.5, 104.5, 70, 129.9, 183.2, 249.3, 80, 35.5, 129.9, 104.5, 150.3,
      183.2, 249.3, 25, 100, 100] #input temperatures at inlets

Tt = [450, 40, 75, 20, 70, 35, 80, 80, 25, 25, 35.5, 129.9, 104.5, 150.3, 183.2, 249.3,
      100, 120, 100] #input temperatures at outlets

Qcol = [0,0,0,0,0,0,0,0,0,0, 35.7, -95, 186.8, -286.5, 2552.7, -2832, 0, 0, -2257*1.064965]
                     #input 0 if not a column, -Qreb for reboiler, or Qcond for condenser

Tmin = 10 #input Tmin

o = list(range(len(CP))) #list of length of the number of streams
Tss = list(range(len(Ts))) #list that will become shifted inlet temperatures
Tts = list(range(len(Tt))) #list that will become shifted outlet temperatures

coldnum = [] #list used to identify cold streams
hotnum = [] #list used to identify hot streams
disnumC = [] #list used to identify condensers
disnumR = [] #list used to identify reboilers
columntempsC = [] #used to identify where the condenders fit into the heat cascade
columntempsR = [] #used to identify where the reboilers fit into the heat cascade

for i in o: #function creating modified temps and identifying hot vs cold streams
    if Ts[i] > Tt[i]:
        Tss[i] = Ts[i]-(Tmin*.5)
        Tts[i] = Tt[i]-(Tmin*.5)
        hotnum.append(i)

    if Ts[i] < Tt[i]:
        Tss[i] = Ts[i]+(Tmin*.5)
        Tts[i] = Tt[i]+(Tmin*.5)
        coldnum.append(i)

    if Ts[i] == Tt[i]: #identifies columns/shifts temps
        if Qcol[i] > 0:
            Tss[i] = Ts[i]-(Tmin*.5)
            Tts[i] = Tt[i]-(Tmin*.5)
            disnumC.append(i)
            columntempsC.append([Tts[i], Qcol[i]])
        if Qcol[i] < 0:
            Tss[i] = Ts[i]+(Tmin*.5)
            Tts[i] = Tt[i]+(Tmin*.5)
            disnumR.append(i)
            columntempsR.append([Tts[i], Qcol[i]])
```

In [500]:

```
Tall = [] #list of all shifted temperatures
for i in o:
    Tall.append(Tss[i])
    Tall.append(Tts[i])


Tstar = list(set(Tall)) #list of all temperatures, without repeats, in descending order
Tstar.sort(reverse=True) #***HEAT CASCADE TEMPS***
print(Tstar)
print()

CPtot = [] #will be list of net CPs for each temp interval
lenT = list(range(len(Tstar) - 1)) #list for iterating through temperature intervals
n=0
```

```
p=0
lentempc = list(range(len(coldnum))) #iterates over cold streams
lentemph = list(range(len(hotnum))) #iterates over hot streams

#function to calculate net CPs
for i in lenT: #iterate through temp intervals
    p=0
    for j in o: #iterate through streams
        for k in lentempc:
            if j == coldnum[k]: #if cold, increase temp in integer increments to see
                                #if it's in the selected temp range
                x = Tss[j]
                while Tts[j] >= x:
                    if -.005 +x <Tstar[i] -.1 < x +.005: #WILL FAIL IF TEMPS ARE NOT INTEGERS
                        p = p-CP[j]
                    x = x+.1
        for k in lentemph: #same for hot streams, but decreasing temp
            if j == hotnum[k]:
                x = Tss[j]
                while Tts[j] <= x:
                    if -.005 + x<Tstar[i] -.1 <x+.005:
                        p = p+CP[j]
                    x = x-.1
    CPtot.append(p)
print(CPtot)
```

```
[455.0, 445.0, 254.3, 244.3, 178.2, 155.3, 134.9, 125.0, 124.9, 105.0, 99.5, 80.0, 75.0, 65.0, 45.
0, 40.5, 35.0, 30.5, 30.0, 20.0, 15.0]

[-2.439565742, 0.0021415750000000067, 0.0021415750000000067, 0.11984108460000001, 3.3785620146, 3.
3785620146, 3.3785620146, 2.4031924756000005, 2.4031924756000005, 0.0730490556000003,
0.7281215194000001, -0.5033070516000002, -4.7850558553, -5.3801283191, -4.1486997481, -
1.7091340061000002, -4.1508413231, -3.6282606779, 0.7675528821, 0.5225806452]
```

In [501]:

```
H = [] #list of delta H values for intervals
Hval = list(range((len(Tstar) - 1))) #list to iterate through temperature intervals

for i in Hval:
    H.append((Tstar[i]-Tstar[i+1])*(CPtot[i])) #calculates delta H

print(H) #before column integration

count = 0
for i in range(len(Tstar)): #adds in column values
    for j in range(len(columntempsR)):
        if Tstar[i] == columntempsR[j][0]:
            H.insert(count + i, columntempsR[j][1])
            count = count+1
    for j in range(len(columntempsC)):
        if Tstar[i] == columntempsC[j][0]:
            H.insert(count + i, columntempsC[j][1])
            count = count+1
print()
print(H) #after column integration
```

```
[-24.39565742, 0.4083983525000012, 0.021415750000000067, 7.921495692060003, 77.36907013433992, 68.
92266509784002, 33.44776394454002, 0.24031924755998638, 47.82353026444002, 0.40176980580000166,
14.198369628300004, -2.516535258000001, -47.850558553000006, -107.60256638199999, -18.66914886645,
-9.40023703355, -18.678785953949998, -1.81413033895, 7.6755288209999994, 2.6129032260000002]

[-24.39565742, 0.4083983525000012, -2832, 0.021415750000000067, 7.921495692060003, 2552.7,
77.36907013433992, -286.5, 68.92266509784002, -95, 33.44776394454002, 0.24031924755998638,
47.82353026444002, -2403.626005, 0.40176980580000166, 186.8, 14.198369628300004, -
2.516535258000001, -47.850558553000006, -107.60256638199999, -18.66914886645, -9.40023703355, -18.
678785953949998, 35.7, -1.81413033895, 7.6755288209999994, 2.6129032260000002]
```

In [502]:

```
Hlen = list(range(len(H))) #list of delta H values for intervals

Hsum = []
n = 0
for i in Hlen:
```

```
for i in Hlen:
    n = n+H[i]
    Hsum.append(n)
Qh = -min(Hsum) #Qh is -largest negative number when adding through H

print(Qh)

Hnewsum = list(range(len(Hsum))) #***HEAT CASCADE Hs***

for i in Hlen:
    Hnewsum[i] = Hsum[i] + Qh
print()
print(Hnewsum)
```

```
2855.9872590675

[2831.5916016475, 2832.0, 0.0, 0.0214157499999601, 7.942911442059994, 2560.64291144206,
2638.0119815763996, 2351.5119815763996, 2420.43464667424, 2325.43464667424, 2358.88241061878,
2359.12272986634, 2406.9462601307796, 3.3202551307795147, 3.7220249365796008, 190.52202493657978,
204.72039456487983, 202.20385930687962, 154.35330075387947, 46.75073437187939, 28.081585505429302,
18.68134847187912, 0.0025625179291637323, 35.70256251792898, 33.8884321789792, 41.56396099997937,
44.17686422597944]
```

In [503]:

```
Qc = sum(H)+Qh #Qc is ending value of cacade after adding Qc
print(Qc)

pinch = 0
count = 0
recheckR = len(columntempsR)*[0]
recheckC = len(columntempsC)*[0]

Q = Qh
for i in Hlen: #finds pinch by checking for zero value in cascade
    Q = Q + H[i]
    if -.01 < Q < .01:
        pinch = Tstar[i+1-count]
        print(pinch)

    for j in range(len(columntempsR)):
        if Tstar[i+1-count] == columntempsR[j][0]:
            if recheckR[j] == 0:
                count = count + 1
                recheckR[j] = 1
            else:
                recheckR[j] = 0
    for j in range(len(columntempsC)):
        if Tstar[i+1-count] == columntempsC[j][0]:
            if recheckC[j] == 0:
                count = count + 1
                recheckC[j] = 1
            else:
                recheckC[j] = 0

#note that two pinches are printed. In the readout of the code above this, heat cascade values are
displayed,
#and the point at which the cascade is at ~3.32 kW is also a pinch, even if it is not reflected ex
actly through
#the code.
```

```
44.17686422597944
254.3
45.0
```

In [507]:

```
import networkx as nx #same graph procedure as above but for the steam raising network
G = nx.Graph()

G.add_node('1', node_color= '#f20d0d')
G.add_node('2', node_color= '#f2690d')
G.add_node('3', node_color= '#f2ba0d')
G.add_node('4', node_color= '#d9f20d')
```

xviii. Eigenvector Centrality for Steam Raising Network Code

```
for i in Hlen:
    n = n+H[i]
    Hsum.append(n)
Qh = -min(Hsum) #Qh is -largest negative number when adding through H

print(Qh)

Hnewsum = list(range(len(Hsum))) #***HEAT CASCADE Hs***

for i in Hlen:
    Hnewsum[i] = Hsum[i] + Qh
print()
print(Hnewsum)
```

```
2855.9872590675

[2831.5916016475, 2832.0, 0.0, 0.0214157499999601, 7.942911442059994, 2560.64291144206,
2638.0119815763996, 2351.5119815763996, 2420.43464667424, 2325.43464667424, 2358.88241061878,
2359.12272986634, 2406.9462601307796, 3.3202551307795147, 3.7220249365796008, 190.52202493657978,
204.72039456487983, 202.20385930687962, 154.35330075387947, 46.75073437187939, 28.081585505429302,
18.68134847187912, 0.0025625179291637323, 35.70256251792898, 33.8884321789792, 41.56396099997937,
44.17686422597944]
```

In [503]:

```
Qc = sum(H)+Qh #Qc is ending value of cacade after adding Qc
print(Qc)

pinch = 0
count = 0
recheckR = len(columntempsR)*[0]
recheckC = len(columntempsC)*[0]

Q = Qh
for i in Hlen: #finds pinch by checking for zero value in cascade
    Q = Q + H[i]
    if -.01 < Q < .01:
        pinch = Tstar[i+1-count]
        print(pinch)

    for j in range(len(columntempsR)):
        if Tstar[i+1-count] == columntempsR[j][0]:
            if recheckR[j] == 0:
                count = count + 1
                recheckR[j] = 1
            else:
                recheckR[j] = 0
    for j in range(len(columntempsC)):
        if Tstar[i+1-count] == columntempsC[j][0]:
            if recheckC[j] == 0:
                count = count + 1
                recheckC[j] = 1
            else:
                recheckC[j] = 0

#note that two pinches are printed. In the readout of the code above this, heat cascade values are
displayed,
#and the point at which the cascade is at ~3.32 kW is also a pinch, even if it is not reflected ex
actly through
#the code.
```

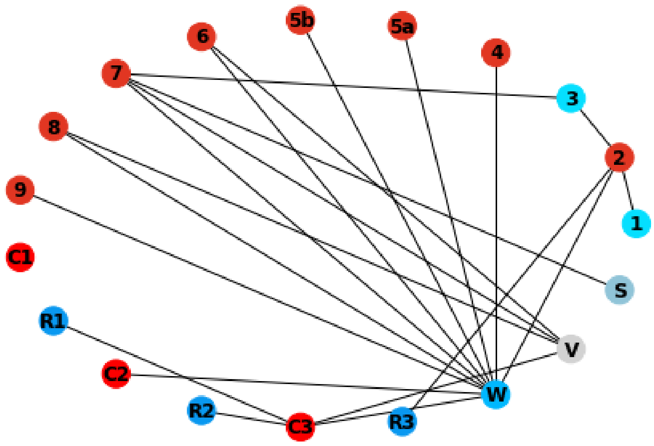
```
44.17686422597944
254.3
45.0
```

In [507]:

```
import networkx as nx #same graph procedure as above but for the steam raising network
G = nx.Graph()

G.add_node('1', node_color= '#f20d0d')
G.add_node('2', node_color= '#f2690d')
G.add_node('3', node_color= '#f2ba0d')
G.add_node('4', node_color= '#d9f20d')
```

```python
G.add_node('5a', node_color= '#5af20d')
G.add_node('5b', node_color= '#0df23f')
G.add_node('6', node_color= '#0df279')
G.add_node('7', node_color= '#0df2b3')
G.add_node('8', node_color= '#0df2ec')
G.add_node('9', node_color= '#0d79f2')
G.add_node('C1', node_color= '#0d28f2')
G.add_node('R1', node_color= '#2c0df2')
G.add_node('C2', node_color= '#710df2')
G.add_node('R2', node_color= '#d90df2')
G.add_node('C3', node_color= '#f20d98')
G.add_node('R3', node_color= '#f20d52')
G.add_node('W', node_color= '#f20d52')
G.add_node('V', node_color= '#f20d52')
G.add_node('S', node_color= '#f20d52')


G.add_edge('1', '2',  weight = 465.2252 + 10.978 + 510.6)
G.add_edge('2', 'R3', weight = .16726)
G.add_edge('W', '4',  weight = .2763)
G.add_edge('W', '8',  weight = 1.7582 + 7.062)
G.add_edge('W', '5b', weight = .847 + 1.2)
G.add_edge('W', '9',  weight = 1.9012 + 3.8182)
G.add_edge('W', '2',  weight = 13.439)
G.add_edge('W', 'C2', weight = 48.5793 + 118.54)
G.add_edge('2', '3',  weight = .448)
G.add_edge('3', '7',  weight = 42.65177)
G.add_edge('S', '7',  weight = 42.51344)
G.add_edge('W', '7',  weight = 55.11)
G.add_edge('W', '5a', weight = 22.6)
G.add_edge('W', '6',  weight = 34.509)
G.add_edge('V', '6',  weight = 22.891)
G.add_edge('V', '7',  weight = 196.0251)
G.add_edge('V', '8',  weight = 16.3955)
G.add_edge('C3', 'R1',weight = 95)
G.add_edge('C3', 'R2',weight = 286.5)
G.add_edge('C3', 'V', weight = 2168.3166666666666666666666)
G.add_edge('W', 'C3', weight = 2.883)


color_map = []
for node in G:
    if node == '1':
        color_map.append('#05dfff')
    if node == '2':
        color_map.append('#e13622')
    if node == '3':
        color_map.append('#05dfff')
    if node == '4':
        color_map.append('#e13622')
    if node == '5a':
        color_map.append('#e13622')
    if node == '5b':
        color_map.append('#e13622')
    if node == '6':
        color_map.append('#e13622')
    if node == '7':
        color_map.append('#e13622')
    if node == '8':
        color_map.append('#e13622')
    if node == '9':
        color_map.append('#e13622')
    if node == 'C1':
        color_map.append('#fd0000')
    if node == 'R1':
        color_map.append('#0998f0')
    if node == 'C2':
        color_map.append('#fd0000')
    if node == 'R2':
        color_map.append('#0998f0')
    if node == 'C3':
        color_map.append('#fd0000')
    if node == 'R3':
        color_map.append('#0998f0')
    if node == 'W':
        color_map.append('#00b8ff')
    if node == 'S':
```

```
        color_map.append('#90c4d8')
    if node == 'V':
        color_map.append('#d2d2d2')


import matplotlib.pyplot as plt
nx.draw_circular(G,node_color = color_map,with_labels = True, font_weight='bold')
plt.show()

EC = nx.eigenvector_centrality(G)
print(EC)
print()
ECW = nx.eigenvector_centrality(G, max_iter=100, tol=1e-02, weight = 'weight')
print(ECW)
```




```
{'1': 0.06698449238761839, '2': 0.24325102068509669, '3': 0.15333991736577712, '4': 0.164134359666950 8, '5a': 0.164134359666950 8, '5b': 0.164134359666950 8, '6': 0.24759309681177238, '7': 0.3135979384319429, '8': 0.24759309681177238, '9': 0.164134359666950 8, 'C1': 2.909283080721106e-15, 'R1': 0.08036737752710574, 'C2': 0.164134359666950 8, 'R2': 0.08036737752710574, 'C3': 0.2918539561092076, 'R3': 0.06698449238761839, 'W': 0.5960543690891199, 'V': 0.30308174340111277, 'S': 0.08635542497816165}

{'1': 0.025685992371378627, '2': 0.025436759991558335, '3': 0.0012645215193602972, '4': 4.947086731948657e-07, '5a': 4.046476808163789e-05, '5b': 3.6651053455444326e-06, '6': 0.007131398460144780 5, '7': 0.06068672449786642, '8': 0.005079345968489468, '9': 1.0240451106845926e-05, 'C1': 2.6574779989223003e-14, 'R1': 0.0313112901962263, 'C2': 0.0002992231731250043 4, 'R2': 0.09442825938119732, 'C3': 0.6825299132706437, 'R3': 4.353693938423002e-06, 'W': 0.0031145665844965016, 'V': 0.7205390620858562, 'S': 0.0012487969752413418}
```

In [506]:

```
print(['%s %0.4f'%(node,EC[node]) for node in EC])
print()
print(['%s %0.4f'%(node,ECW[node]) for node in ECW])
```

```
['1 0.0670', '2 0.2433', '3 0.1533', '4 0.1641', '5a 0.1641', '5b 0.1641', '6 0.2476', '7 0.3136', '8 0.2476', '9 0.1641', 'C1 0.0000', 'R1 0.0804', 'C2 0.1641', 'R2 0.0804', 'C3 0.2919', 'R3 0.0670', 'W 0.5961', 'V 0.3031', 'S 0.0864']

['1 0.0257', '2 0.0254', '3 0.0013', '4 0.0000', '5a 0.0000', '5b 0.0000', '6 0.0071', '7 0.0607', '8 0.0051', '9 0.0000', 'C1 0.0000', 'R1 0.0313', 'C2 0.0003', 'R2 0.0944', 'C3 0.6825', 'R3 0.0000', 'W 0.0031', 'V 0.7205', 'S 0.0012']
```