A Comparison of Two Air Compressors for PEM Fuel Cell Systems

Galen W. Kulp

Virginia Polytechnic Institute and State University

Master of Science
in
Mechanical Engineering

Dr. Douglas Nelson, Chair
Dr. Walter O'Brien
Dr. Michael Ellis

December 14, 2001
Blacksburg, Virginia

Keywords: Fuel Cells, Alternative Energy Sources, Compressor Technology, Automotive

A Comparison of Two Air Compressors for PEM Fuel Cell Systems

Galen Kulp

(ABSTRACT)

Proton exchange membrane (PEM) fuel cells are considered one of the best potential alternative power sources for automobiles. For this application, high efficiency and high power density are required. Pressurizing the fuel cell system can give higher efficiency, higher power density and better water balance characteristics for the fuel cell, but pressurization uses a percentage of the fuel cell output power. The compressor used to elevate the pressure has a direct effect on the system efficiency and water balance characteristics. A variety of compressors are being developed for fuel cell applications. Two compressor and expander technologies are discussed in this paper: the Opcon 1050 positive displacement twin-screw compressor and expander, and a Honeywell turbocompressor and expander. The effect of these compressors and expanders on the system at maximum load, low load, and set minimum airflow are examined. The effects of ambient conditions, stack temperature, and increased twin-screw compressor pressure are also examined.

The turbocompressor proves to be a superior machine in terms of efficiency, and therefore offers the most promising effect on system efficiency of the two compressors. The twin-screw compressor, on the other hand, offers more flexible pressure ratio and better water balance characteristics at low fuel cell loads, which is an important factor with PEM fuel cell systems. Increased ambient and stack temperature has a significant negative effect on the water balance and a small positive effect on efficiency. Increasing the pressure for the twin-screw compressor significantly improves the water balance characteristics with some loss in efficiency. These results show the importance of determining the system operating range and operating conditions in the choice of a compressor for a fuel cell system

Table of Contents

Table of Figures

Chapter 1: Introduction

The hope of technology is to provide utility and comfort with no damage to the user or to the surroundings. For many years now, petroleum products and other fossil fuels have given us utility and comfort in a variety of areas, but are blamed for such disasters as lung disease and global warming. Also, the fears of depleting the petroleum supply demand that a new technology be developed to take its place. With the advent of fuel cell technology, a large step is being made to reduce emissions from and improve the efficiency of power production, while using alternative energy sources. With recent improvements in power density—especially in PEM fuel cells—it is becoming possible to use this technology in automobiles.

Several issues specific to fuel cell systems must be considered for such systems to be used successfully in an automotive application. Some of the more general issues to consider are the physical size of the system—whether it will fit under the hood—and the power capabilities of the system. Other issues are more specific to the actual system components being used. These include the effects of a particular compressor technology on the system, benefits from adding an expander to the system, and changes in water balance characteristics and efficiency due to variation in the inlet conditions and system parameters of a particular system. These more specific issues will be the focus of this paper.

Fuel cells have better efficiency and power density when operating at higher air pressures than ambient. Therefore, it is most likely that those fuel cells for automotive applications will have pressurized air supplies. Unfortunately, fuel cells require air mass flows and pressure ratios that are not readily available in current compressor technologies. Some superchargers are capable of these requirements, but other compressors, such as fans, blowers, and industrial compressors, produce combinations of flow and pressure ratio that are unusable for a fuel cell system, and often in sizes and weights that are inappropriate for automotive applications. As a result, the Department of Energy (DoE), in cooperation with the automotive industry, set guidelines for the development of an air compressor and expander combination that would meet the needs of a fuel cell system and be appropriate for automotive applications.

Several different technologies are being explored by private companies to fill the need for fuel cell compressors and expanders. However, this paper will cover only two of those technologies: a twin-screw compressor (an Opcon 1050 twin-screw supercharger) and a turbocompressor (a Honeywell compressor still in development). These two kinds of compressors show the highest potential for use in an automotive application, considering the DoE guidelines mentioned above. Each are mature technologies and are closest to the weight and volume guidelines in comparison to other technologies currently being developed. The twin-screw compressor has actually been used in automotive fuel cell applications, and so has proven itself useful in this area. Though other technologies may follow the pressure ratio and flow rate requirements better, they are currently too bulky to find room under the hood and too heavy to be considered. Additionally, these compressors with larger volumes and weights generally require larger electric motors to

power them than do the smaller and faster twin-screw and turbocompressor. This additional weight and volume reduces the benefits of having a compressed air supply for increased fuel cell system power density.

The two technologies chosen for this paper do not come without some problems, however. The twin-screw is noisy at operating speeds, and requires lubrication (though the compressed air is not exposed to the lubrication). The turbocompressor has poor turndown characteristics so that the pressure ratio at low flows is lower than the DoE guideline [Roan (2000)], and leads to problems with water management at light fuel cell loads. As compared to the other technologies, however, these compressors seem to have the qualities that will lead to their use in fuel cell automobiles.

Compression of the air supply for the fuel cell system makes it possible to recapture some of the energy used for compression by using an air expander downstream of the fuel cell. This is a device that would most likely be the same technology as the compressor and be connected directly to the compressor or the compressor drive system to help supply additional power for compression at no additional energy cost. Since space under the hood is at such a premium, it may not be in the manufacturer's best interest to use an expander for an automotive application, in spite of the "free" energy it would provide. The benefits of expanders corresponding to the two compressor technologies will be covered in this paper. This information needs to be weighed against the space and costs of utilizing an expander in an automobile.

If the compressor drive motor can be controlled by computer, then load following could be added to the system. Load following is when the compressor output is matched with the requirements of the fuel cell stack rather than keeping the compressor at a constant output. Since the compressor is the largest parasitic load in the fuel cell system, load following can have a significant effect on the system efficiency. This paper takes the load following approach.

Efficiency and power density changes are not the only effects of a compressor on a fuel cell system. Water balance is also affected. Water balance is the amount of water used to humidify the incoming air stream subtracted from the amount of water produced and collected from the fuel cell. Due to the nature of the membrane material in the fuel cell, it must be kept moist at all times in order to continue to operate. Otherwise, the electrical resistance of the membrane will increase as it dries until it develops "hot spots," which can burn holes right through the material. Fortunately, the electrochemical process within the fuel cell—the combination of hydrogen and oxygen—produces water as a byproduct. This water is not naturally distributed across the membrane enough to keep it moist, however, so the air stream entering the fuel cell must be humidified. If the fuel cell system is designed well, there will be a sufficient amount of water taken from the reaction in the fuel cell to humidify the inlet air stream, indicating a positive water balance. If the system is designed poorly, more water will be needed than what can be taken from the fuel cell reaction—a negative water balance. Additional water would then have to be supplied and replenished from time to time, adding an annoyance to drivers of fuel cell vehicles.

Water balance can be affected by variations in the fuel cell load and by the stoichiometric ratio (SR) of air that is supplied. The variations in the load affect the water balance primarily through variation in compressor pressure, which results from load following. The SR, which is the amount of supplied air divided by the minimum amount of air required by the fuel cell, is necessary to ensure that the fuel cell will have enough oxygen. Supplying just enough oxygen for the amount of hydrogen entering the fuel cell will result in oxygen starvation at some parts of the fuel cell since distribution of the air is not perfect. Therefore more air must be supplied than is necessary for the reaction. This additional air, however, must also be humidified, and therefore has a negative effect on the water balance.

Temperature of operation of the fuel cell or of the ambient air entering the system can have a significant effect on the water balance. Figure 1, shown below, shows how significant the temperature and pressure effects can be. Lower pressures and higher temperatures can drastically increase the amount of water that air can accept until it is saturated (100 percent relative humidity). At 1 atm and 90°C there is more water by mass than air by mass in saturated air.

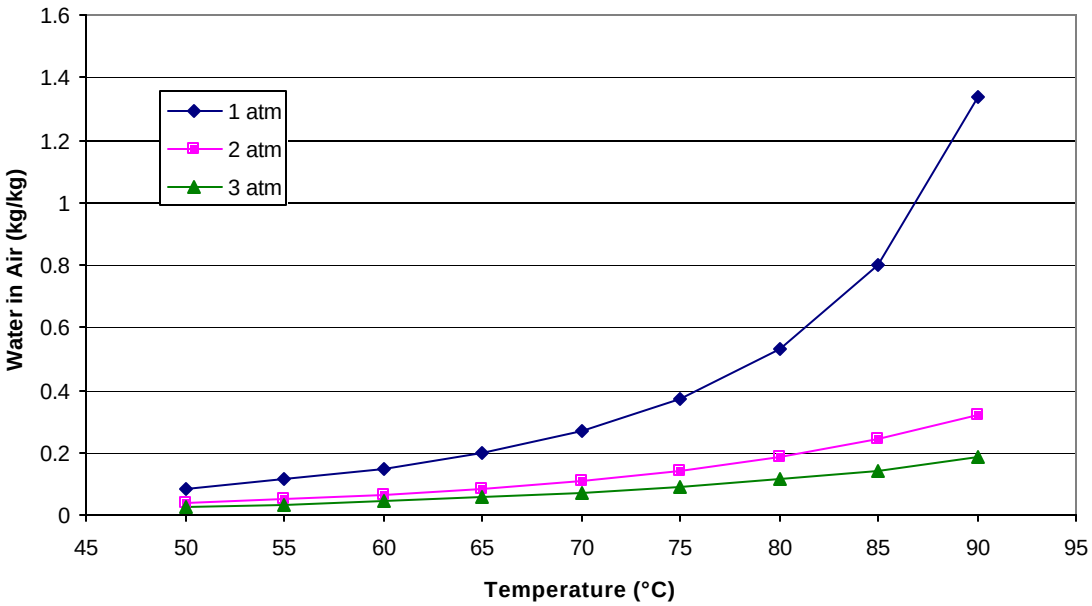

Figure 1: Water contained in air at 100 percent humidity

The objective of this paper is to examine the effects of a real twin-screw compressor/expander and a real turbocompressor/expander on the entire fuel cell system. Systems with and without expanders are used to show the effect of the expander. Optimum pressure and flow rate operating lines (with limits) for system efficiency are found. Different operating conditions and their effects on the system are also explored. After a full description of the system and the computer programs that simulate it, results such as system efficiency, system power, and compressor/expander power are given for the various operating conditions under examination.

Chapter 2: Review of Literature

Many other papers have been written on fuel cell system modeling and pressurization effects. Barbir, et al (1999), find an operating pressure and temperature for a particular fuel cell system at full or part load that optimizes the efficiency and the system's size while taking water balance into account. They use a steady state mathematical model for a fuel cell system, which includes a reformer, a compressor, a humidifier, a radiator, an expander, a condenser, water and fuel pumps, several water separators, and an exhaust heat exchanger. Because there is a reformer rather than a pure hydrogen supply, it is assumed that the fuel stream is approximately 40 percent hydrogen, 40 percent nitrogen, and 20 percent carbon dioxide, with a significant but varying amount of water vapor. There is no recirculation of the reformate in this system, but the tail gases are burned in the expander to recover as much energy as possible. Some of the results from their work include stack efficiency and optimum stack size at different pressures and temperatures, and radiator and condenser heat load and surface area at different temperatures and pressures. Barbir shows that for pressurized stacks, higher stack temperatures improve efficiency, while low-pressure systems have better performance at lower temperatures. Condenser fan loads at low pressures pull the efficiency down at high temperatures. Their resulting optimized system operates at 308 kPa (absolute), 60°C, and maintains a system efficiency of 32 percent at nominal load.

Wiartalla, et al (2000), compare many kinds of compressor/expander technologies and their suitability for use in fuel cell systems in their analytical study. These include turbomachinery and twin-screw compressors, as well as others. The fuel cell system used includes a 75 kW stack at a constant efficiency of 50%, a compressor, a fuel evaporator, a methanol reformer, a humidifier, a tail gas burner, and an expander. Results include a survey of compressor characteristics and their suitability for use in fuel cells, as well as fuel cell system output with some compressor/expanders that are most suitable for use with fuel cells: a twin-screw, a turbocharger, and a roots type supercharger. Best results came from a system with a turbocompressor and a tail gas burner/expander, giving a maximum system efficiency of approximately 37.6 percent.

Cunningham, et al (2001), offer a comparison of high and low-pressure operation of a fuel cell system. A twin-screw compressor is used for the high-pressure condition and a blower is used for the low-pressure condition. The fuel cell system consists of the respective compressors with a condenser, a water pump and storage unit, and a radiator. Both systems are optimized based on their compression method, meaning that the size of the components can change, including fuel cell stack size. Results of the comparison show that the system can obtain the same peak power values of 86 kW, but the system with the blower needs a fuel cell stack size that is 16.3 percent larger. Results from putting the two systems through a range of current densities show that the blower system obtains 1.5 to 2 efficiency percentage points higher than the twin-screw system. The paper acknowledges in that an expander could have a significant effect on the results.

Fronk, et al (2000), examine some key challenges in thermal management in a fuel cell system and discuss possible solutions through changes within the thermal system as well

as the fuel cell stack itself. The primary difficulty with thermal management in fuel cells is heat rejection due to the low temperature difference between the coolant and ambient air. Fronk examines potential solutions to this problem through thermal load reduction, radiator size and airflow, and heat rejection temperature. Stack issues that Fronk addresses are operating conditions, including pressure, and the optimization of the size and cost of the stack. Conclusions from this paper suggest that the solution of higher stack temperatures and pressures may be best to improve the difficulties with thermal management, though it would potentially lower the system efficiency. Other suggestions include developing new technology that would improve the coolants and lower the necessary water content in the PEM.

Chapter 3: Definition of System

The system that is used in this paper is a simplified steady state, proton exchange membrane (PEM) fuel cell system. Figure 2 shows a block diagram of the system components. The system is made up of a humidifier/heat exchanger, a compressor, an expander, two water separators, a water pump, a radiator and fan, and a fuel cell stack. Since the primary focus of this paper is the effect of each of the two compressor technologies on the air supply and the subsequent effects on the fuel cell, few specifics of hydrogen fuel delivery are included in the system. Calculations for fuel consumption assume that a regulated supply of pure hydrogen is available from a compressed gaseous storage tank. All energy produced from hydrogen is calculated from a percentage of the higher and lower heating values of hydrogen depending on the amounts of liquid and vaporized water exiting the fuel cell. Four computer programs are used to simulate the fuel cell system—two for each compressor technology.

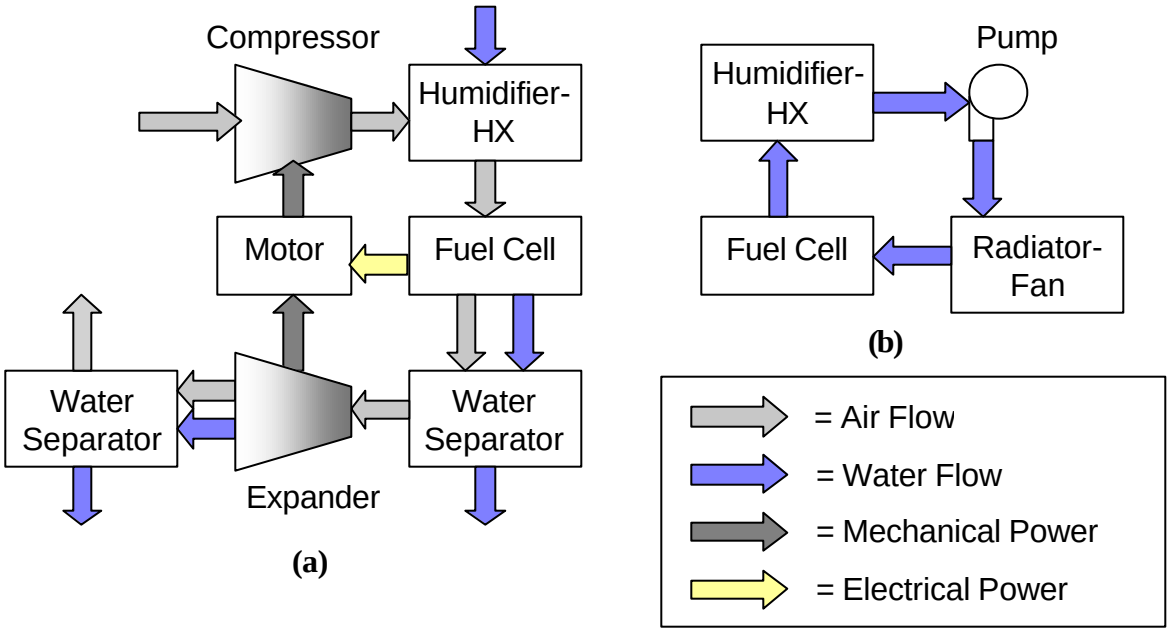

Figure 2: System Diagram - (a) air system, (b) coolant loop

Several assumptions are made in order to make the system a manageable project. Constant specific heats and ideal gases are assumed. Considering the operating temperature and pressure ranges of the system and the small change in the air and fuel properties within this range, these are fair assumptions. Also, there are no pressure drops throughout the system, except across the fuel cell and the expander. Flow resistance in tubes or through the other components is not considered. More assumptions are made for each component of the system, and these will be addressed as each component is discussed in turn.

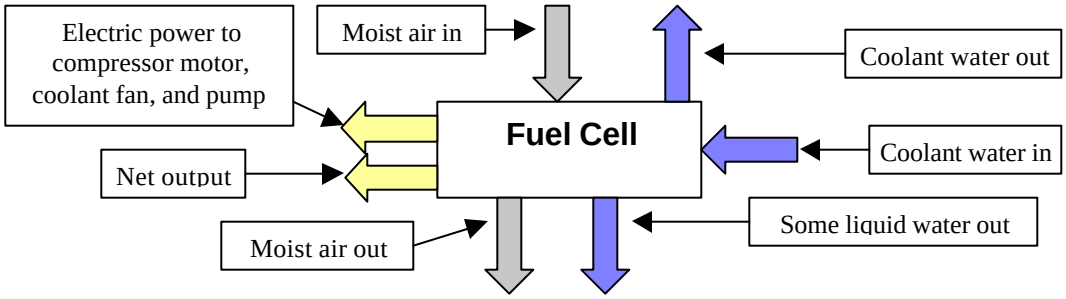

Figure 3: Fuel cell component

Fuel Cell

The power plant for the system is a proton exchange membrane fuel cell. Warm, moist air enters the fuel cell from the humidifier/heat exchanger. After traveling through the fuel cell, the air stream gains some water due to the electrochemical reaction of hydrogen and oxygen. Therefore, under most conditions, there is some liquid water along with the moist air stream at the exit to the fuel cell, which will be gathered by a water separator. The fuel cell is cooled by a coolant loop coming from the radiator and going to the humidifier/heat exchanger. The primary parasitic load on the system is the drive motor for the compressor, and this power is shown leaving the fuel cell with the coolant fan and pump power in Figure 3**.**

The fuel cell stack is modeled after a 60 kW net power/70 kW gross power proton exchange membrane stack. This size of a fuel cell system would be enough to power a small fuel cell vehicle, or a larger hybrid vehicle. The fuel cell stack is made up of 210 fuel cells and has a total area of 678 square centimeters. The structure of the hydrogen channels "dead ends" in the fuel cell, so any water taken to humidify the hydrogen stream is neglected since it does not leave the stack.

The primary element of the fuel cell in the computer program is the polarization curve, which is specified in the computer program section of this paper. The polarization curve model determines cell voltage based on current density, stack temperature, and the average oxygen partial pressure across the fuel cell. The partial pressure of oxygen is calculated as an average across the fuel cell, depending on the amount of oxygen consumed, the water vapor pressure in the air, and the total pressure. Changes in the partial pressure of oxygen affect the output voltage of the fuel cell. Also, a pressure drop across the fuel cell is calculated that is linear with mass flow. Other factors, such as fuel cell membrane moisture or hydrogen pressure, are not considered in the performance.

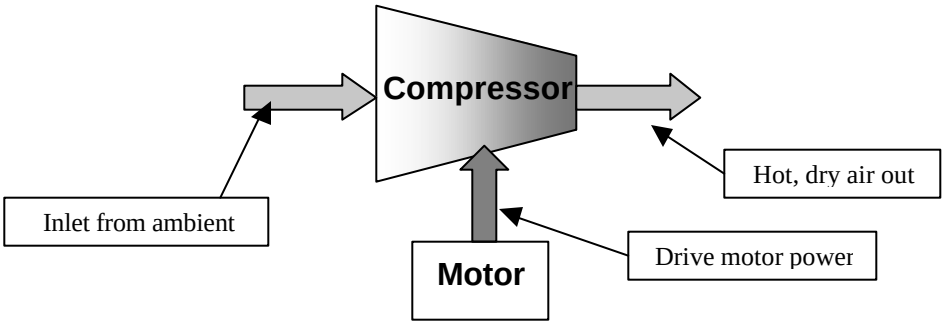

Figure 4: Compressor and drive motor components

Compressor and drive motor

The compressor component, as shown with the drive motor in Figure 4, is based on a real twin-screw compressor and a real turbocompressor. Ambient air enters the compressor and the hot, compressed air exits to the humidifier/heat exchanger. The drive motor powers the compressor. The motor and motor inverter combination is assumed to have a fixed efficiency of 88 percent at all speeds and loads.

There is a map for each compressor from which adiabatic efficiency, volumetric efficiency, outlet temperature, and compressor speed are found in terms of mass flow and pressure ratio. In the computer program, whenever information is required for a particular pressure ratio and mass flow that is not directly in the given data, the program uses bicubic interpolation or linear extrapolation to gather the desired data. As a result, the system is capable of giving output data at many different operating points for the compressor. The turbocompressor is characteristically more efficient than the twin-screw compressor, although the turbocompressor has more limitations on its operating line as will be explained later. The maps have been normalized—divided by their largest component—so that they can easily be scaled up and down according to the needs of a fuel cell system.

The turbocompressor is modeled based on a Honeywell turbocompressor. The maximum pressure ratio is approximately 3.5 and the maximum mass flow rate is approximately 95 grams per second at a maximum speed of 105,000 RPM with a maximum efficiency of 79 percent. Because this compressor is not yet in production, performance data are taken from estimated maps (Figures 5 and 6). Figure 5 shows the speed map for the turbocompressor. This was scaled up to better meet the needs of the fuel cell system used in this paper, as is evident by the fact that the normalized mass flow goes greater than 1.

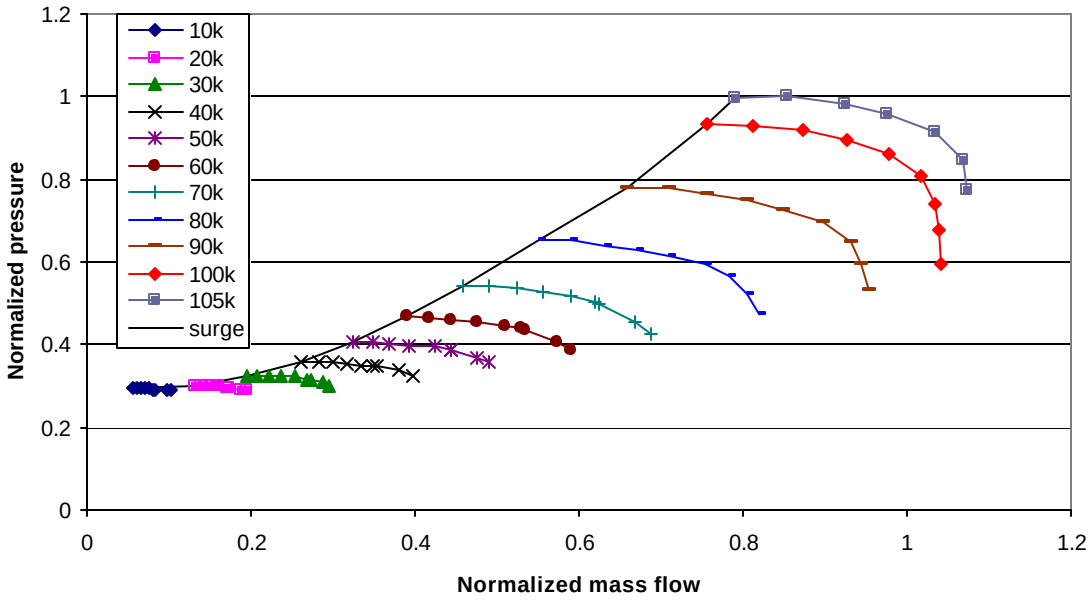

Figure 5: Turbocompressor speed map

The twin-screw compressor is an Opcon 1050 twin-screw supercharger. The displacement of the 1050 is 0.5 liters per revolution and the built-in pressure ratio is 1.44. The built-in pressure ratio depends on the location and size of the exhaust porting, and does not necessarily indicate the highest efficiency operation pressure. The maps for this machine, shown in Figures 7, 8, and 9, had to be constructed from two different compressors. Few performance maps were available for the 1050 machine, so data were taken from a 2089 machine with the same internal pressure ratio to complete the rest of the map. The data from the 2089 were scaled down to approximate the performance of the 1050 where the data were needed. Some of the data points had to be approximated using the trends of the data already present, because sufficient data were simply not available from either machine to complete the entire map. The maps developed for the 1050 reach a pressure ratio of 2.7 and a flow rate of approximately 150 liters per second at the maximum speed of 20,000 RPM, with a maximum efficiency of 67 percent.

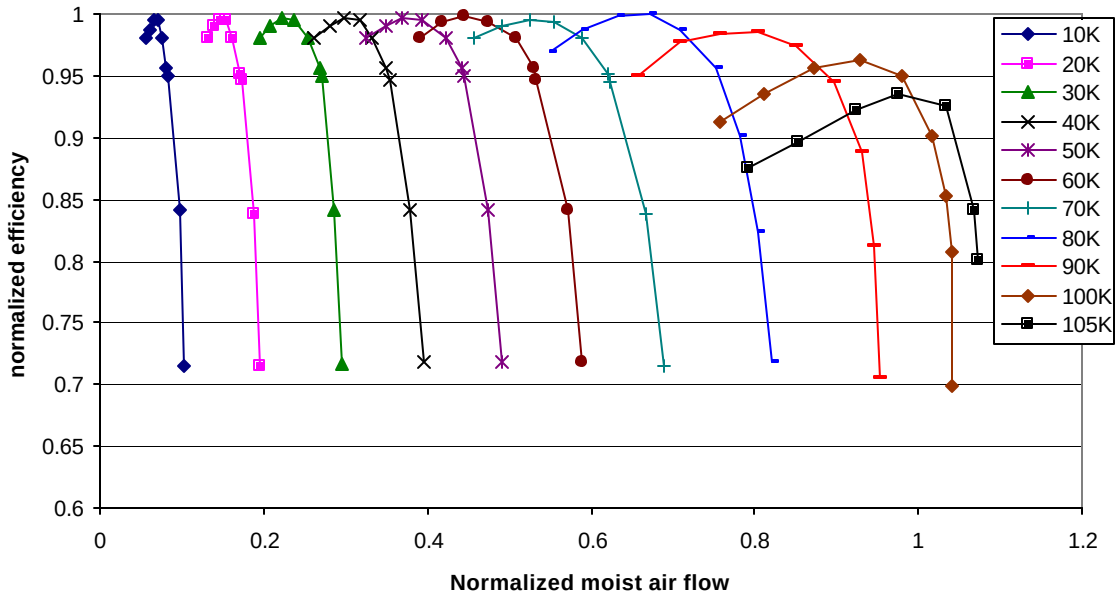

Figure 6: Turbocompressor efficiency map

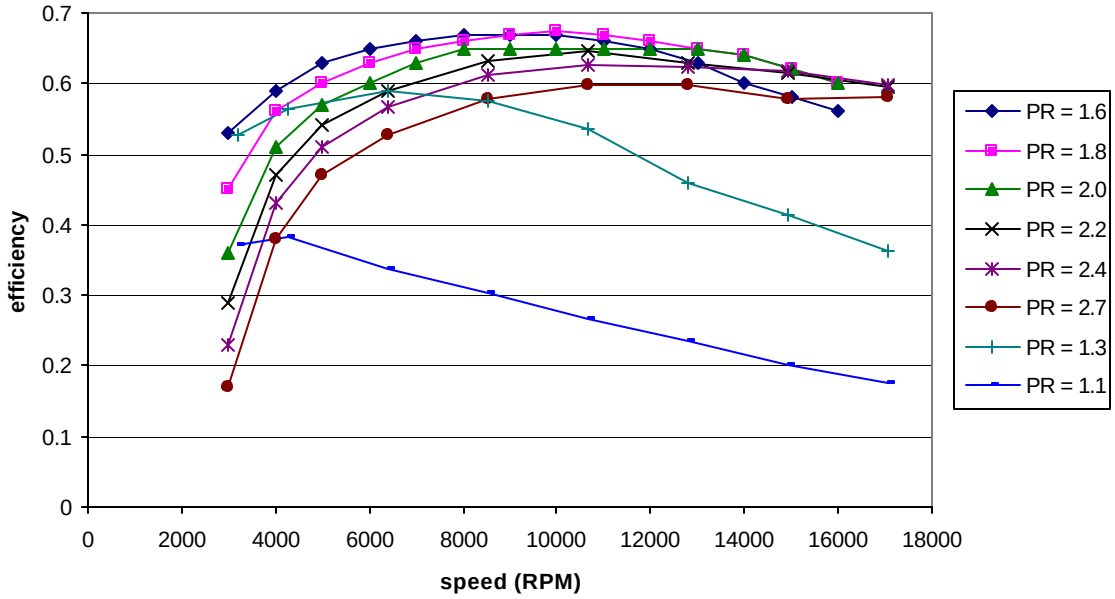

Figure 7: Twin-screw adiabatic efficiency map

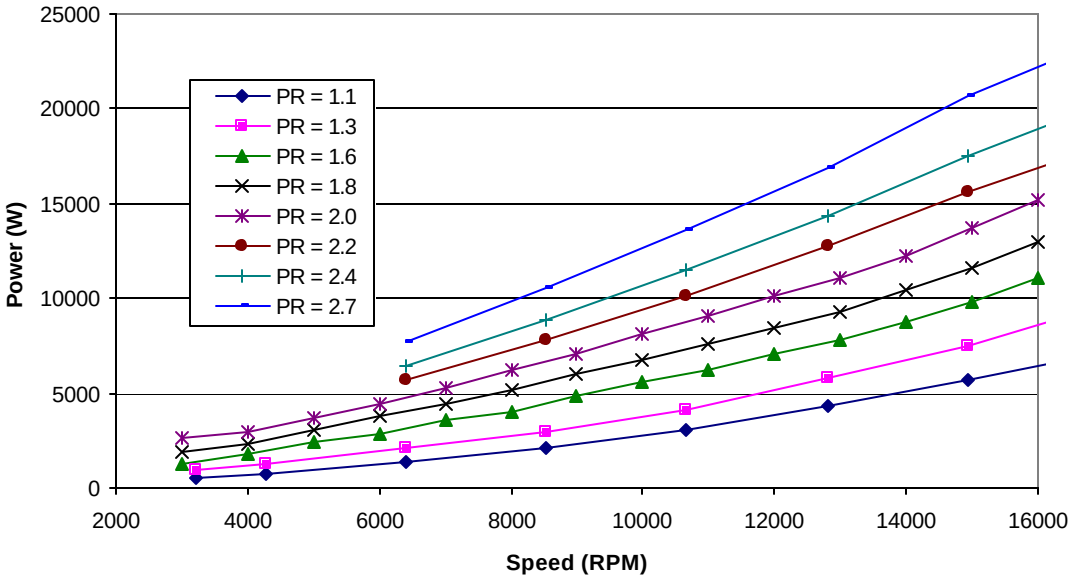

Figure 8: Twin-screw power consumption map

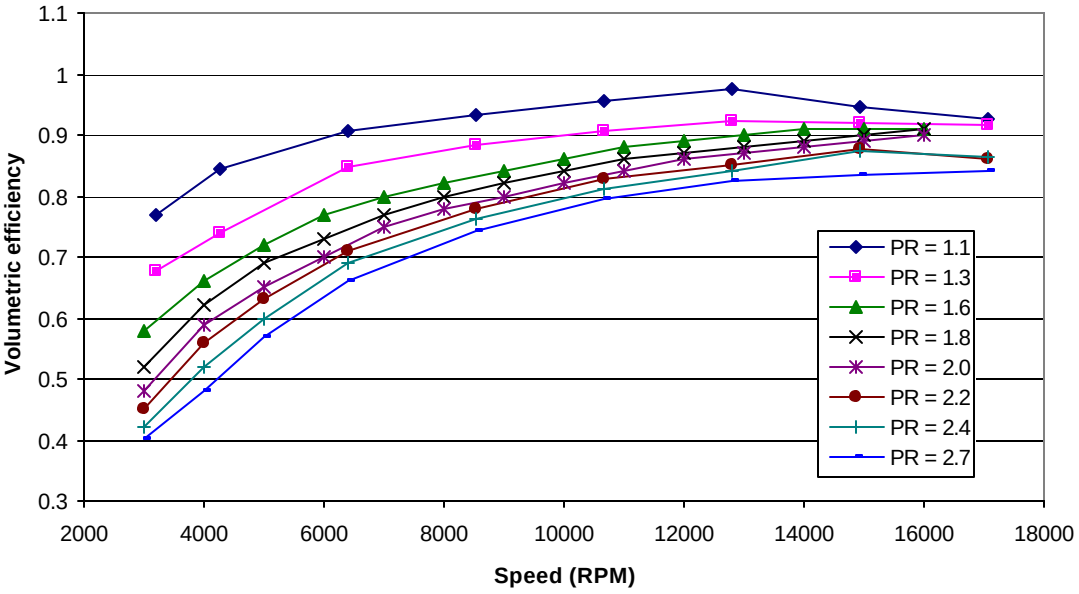

Figure 9: Twin-screw volumetric efficiency map

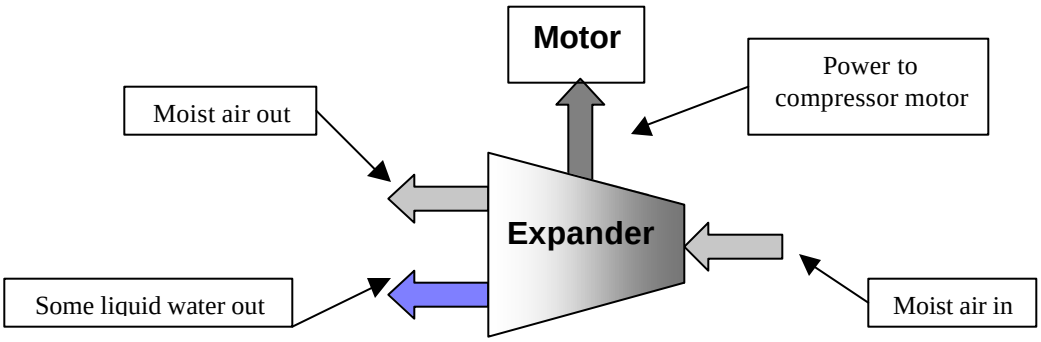

Figure 10: Expander component

Expander

The expander component is shown in Figure 10. Moist air coming from the fuel cell enters the expander. The water separator directly after the fuel cell has taken out all water from the air stream. Through the process of expansion, the moist air stream is cooled and lowered in pressure. Generally, more liquid water precipitates out of the air stream as a result of the cooling and in spite of the pressure drop.

The expander component in each of the compressor configurations is considered to be of the same technology as the compressor. In fact, the twin-screw expander uses the same adiabatic efficiency map as the twin-screw compressor to find power extracted. This is allowable for the twin-screw machine due to its positive displacement technology. The expander for the turbocompressor is based on a thermodynamic model for turbomachinery, and a constant expander efficiency of 81 percent is assumed. This is estimated from Department of Energy guidelines. It is assumed that the expander is able to maintain a speed nearly proportional to, but not the same as, the compressor speed. In a physical system, an expander would generally be directly connected to the compressor and would therefore have the same rotational speed. However, no data are given specifically for the expander, so it is difficult to determine how the output of the expander would vary with respect to its rotational speed if flow rate and pressure do not vary.

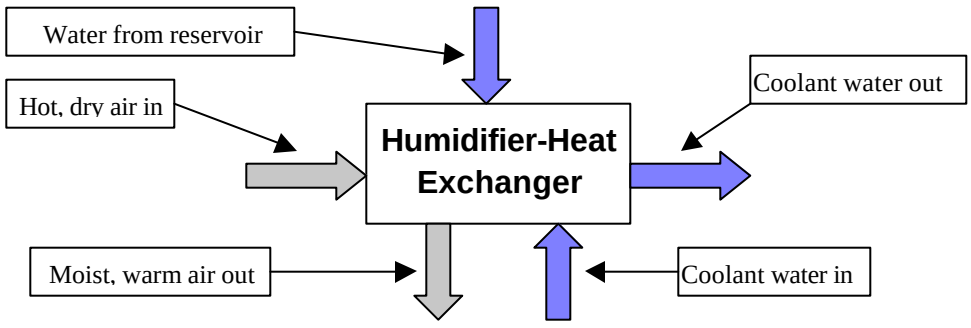

Figure 11: Humidifier/heat exchanger component

Humidifier/Heat Exchanger

The humidifier/heat exchanger component (Figure 11) in the system heats or cools and humidifies the air stream directly after the compressor to the desired fuel cell temperature

and relative humidity. The hot, dry air from the compressor enters the humidifier/heat exchanger. Heat from the coolant loop and water from the water reservoir are added to the air stream, and warm, moist air at the desired fuel cell temperature exits.

The heater and humidifier are combined into one component since it is useful to have a heat source available to help with evaporation of liquid water into the air stream. The heat source for the heat exchanger is the coolant flow coming from the fuel cell, and the water source for humidification is the water produced in the fuel cell and gathered by the water separators into the water reservoir. In this system, the desired humidity of air is set as a requirement. Since there is no external heat source to the fuel cell system, it is possible under certain conditions that insufficient heat will be available for evaporation. By reducing the exit temperature of the humidifier/heat exchanger, less heat is needed for humidification. This adjustment of temperature is covered in detail in the computer program explanation. Also, the effect of temperature on the ability of air to hold water vapor is covered in the Results section.

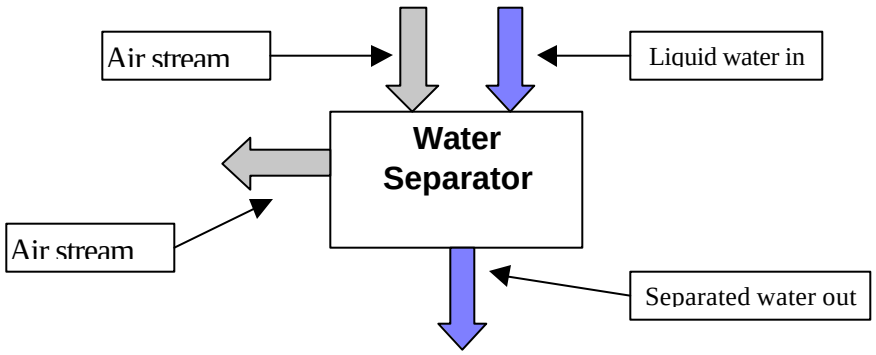

Figure 12: Water Separator Component

Water Separators

The water separator components are ideal in that they can extract all liquid water that is produced by the fuel cell and is condensed out of the air in the expander, as reflected in Figure 12. As previously noted, the water separator is assumed to have a negligible pressure drop. Also, though real water separators lose some air from the system, this is not taken into account in this system model.

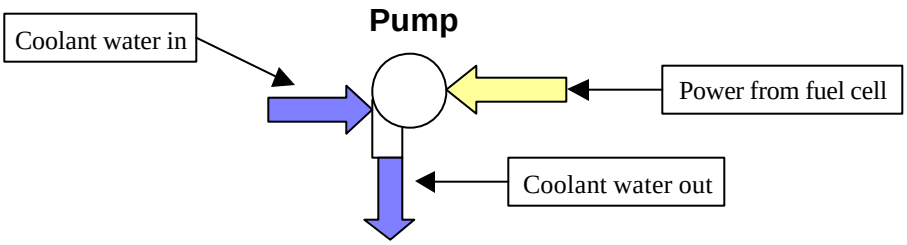

Figure 13: Water pump component

Water Pump

The water pump component (Figure 13) is based on a power and mass flow rating estimated from a 3600-RPM CD-100 Price Pump water pump that is rated at 3.2 L/s (50 gpm) and 930 W (1.25 bhp). The mass flow is calculated as a linear relationship with the amount of heat rejected from the system, and the power is estimated as a linear relationship with the mass flow.

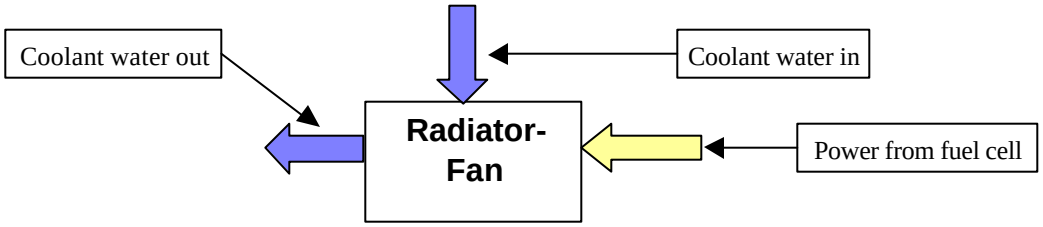

Figure 14: Radiator-fan component

Radiator

The radiator component (Figure 14) is based on a single-pass, crossflow heat exchanger with both fluids unmixed. A fan improves the effectiveness of the radiator and keeps the system at a stable temperature. The power output of the fan is determined by the percentage of time that it needs to operate in order to reject enough heat from the system and maintain the temperature. A fixed fan efficiency of 65 percent is assumed. Calculations are based on the NTU method. If the humidifier takes most of the heat from the coolant loop, or if the radiator would remove too much heat, then a bypass valve is used to force a percentage of the coolant to circumvent the radiator to maintain a minimum temperature of the coolant.

Operating Line

Another requirement for the system, though it is not necessarily a component, is the operating line. The operating line is a schedule of pressure ratio and the mass flow of the air used by the fuel cell system. The actual operating lines used for this paper are shown in Chapter 5. The SR (air mass flow relative to fuel cell oxygen consumption) varies according to the maximum efficiency, but is at least 2.0 at low fuel cell load to 2.5 at high load. This lower bound is necessary to ensure that the entire surface of the fuel cell receives sufficient oxygen for reaction. Imposing an operating line requires some active pressure control mechanism, such as a backpressure orifice. Otherwise the flow rate and

the resistance of the system components to that flow would determine the pressure of the system.

The operating line is an important part of the system since the efficiency and power of both the compressor and the fuel cell are greatly affected by it. The twin-screw compressor allows some freedom in the choice of an operating line, though the efficiency may suffer if the compressor is run at off-design operating points. Positive displacement machines are capable of a wide range of pressure ratios, because they discharge a fixed amount of fluid (depending on volumetric efficiency) for each revolution of the compressor regardless of the pressure ratio. The turbocompressor, on the other hand, does not allow as much freedom as the twin-screw compressor because of the surge line that is characteristic to every turbocompressor. If an operating point is chosen above the surge line (at a higher pressure than it can achieve at a certain mass flow), then it enters into surge. Surge is an unstable state where the compressor experiences dramatic pressure oscillations, which can damage the machine and the fuel cells. The different operating lines used for each compressor are discussed in Chapter 5.

Chapter 4: System Modeling Programs

Matlab ® is the programming language used in the simulation of the fuel cell system. Two main programs are used, one for each compressor, to find the pressure and mass flow operating lines for each of the compressors to give the highest fuel cell system efficiency. Two other main programs are used, one for each compressor, to simulate the fuel cell system with each of the compressors and give system efficiencies and other output. All four programs are attached in the appendix.

Basic Program Structure

All of the fuel cell system programs are based on the same structure. In fact, the central core of all the programs—the fuel cell system components—does not change from program to program. Only the form of the information put into those components changes. Figure 15 shows the structure of the programs and the functions for each part. Each section will be described in detail.

There are several programming loops common to all of the programs, as shown in Figure 15. These are the fuel cell current loop, the "check pressure of system" loop, and the "check heat for humidifier" loop. The fuel cell current loop runs the system through a range of current loads, from 5 A to 475 A. As shown in Figure 15, the current loop begins in the Initial Calculations section and ends in the System Output and Plots section. The "check pressure of system" loop simply makes sure that the compressor supplies enough pressure to overcome the pressure drop caused by the airflow through the system. This loop begins in the Compressor Section and ends at the end of the Fuel Cell Section. The "check heat for humidifier" loop checks to make sure enough heat energy is available to the humidifier to evaporate the amount of water needed for humidification. If not, the stack temperature is lowered to make it easier to reach the desired humidity. This loop begins in the Humidifier/Heat Exchanger Section and ends in the Fuel Cell Section.

The primary difference between the two types of programs lies in the loops. For the high efficiency line program, the loops include multiple pressures and flow rate values whereas the fuel cell simulator does not. These additional loops allow the high efficiency program to check multiple pressures and flow rates simultaneously to determine the parameters that will make the system run most efficiently. As a result, there is a fuel cell pressure loop and a fuel cell flow rate loop in the high efficiency program. For the turbocompressor, the high efficiency program is not significantly different from the fuel cell simulation other than the additional loops mentioned above. However, the twin-screw program is much different because of how the compressor map is accessed. The program calls an interpolation technique every time it needs to find information on the twin-screw compressor. Unfortunately, the interpolation technique is computationally expensive, and seems to take approximately as long when running with single values and with matrices. Therefore, instead of running a single calculation through the program at a time, an entire matrix of pressures and flows are input to the procedure to increase the processing speed. This makes the compressor code more complicated, however, because

of the need to scan through the matrices and execute commands on individual sections of each matrix.

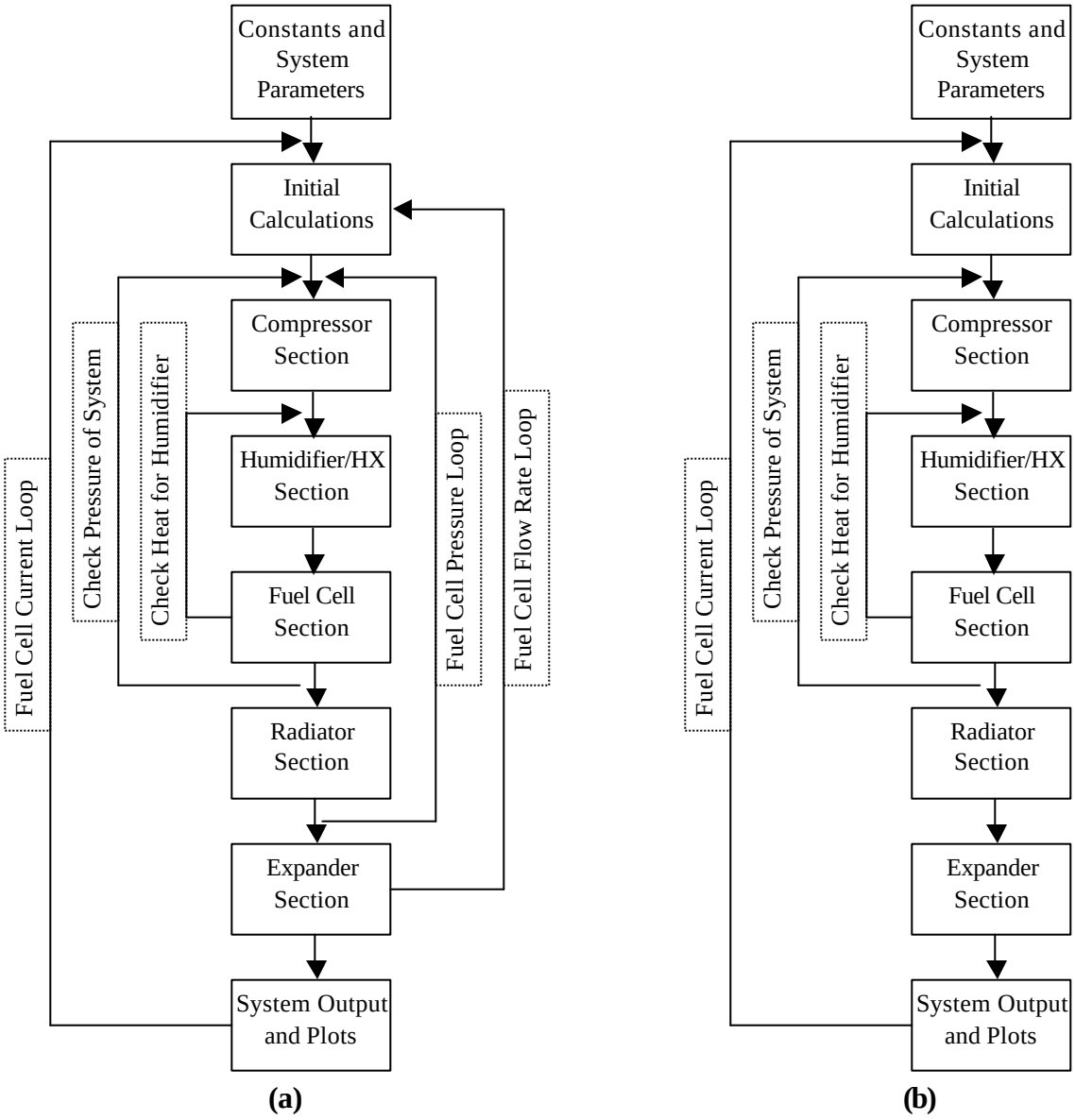

Figure 15: Structure of computer programs. (a) High efficiency pressure and flow rate operating line generator, (b) Fuel cell simulator

Constants and System Parameters

In this section are all of the values that are considered constant to the system. This includes physical constants, compressor-specific constants, fuel cell specific constants, and any user input values. The user has a fair amount of control over the system

parameters within this section. Key parameters for the compressor and fuel cell can be altered to examine different conditions. Because the compressor characteristic values are normalized, the maximum values for the compressor can be used to manipulate its sizing. Also, all ambient conditions (inputs to the compressor) are specified here, as well as the desired fuel cell temperature and relative humidity.

Initial Calculations

This section deals with several calculations performed before the primary components of the fuel cell system are encountered in the program. These include the maximum expected values for the fluid flow rates, an initial calculation of the fuel cell performance, and the calculation of the compressor pressure. The maximum flow rate values are used for the purpose of normalizing the flow rate of the moist air traveling through the system. This normalization is done because all of the compressor characteristic values are normalized, to facilitate scaling the compressor model to different system power levels.

An initial reading of the fuel cell output is necessary so that a guess can be made at the amount of waste heat generated by the fuel cell. The humidifier/heat exchanger section of the program uses this heat value for determining the maximum relative humidity possible in the system. Therefore, some preliminary values of the fuel cell voltage are calculated as a start of the humidification heat loop.

Between this section and the following section is the beginning of the for-loop for the fuel cell current.

Compressor Section

As mentioned before, both compressors in the programs are based on data from real compressors: the Opcon 1050 supercharger and the Honeywell turbocompressor. Actual maps of the compressors (estimated maps for the turbocompressor) were used to find the characteristic adiabatic efficiency, volumetric efficiency (twin-screw only), temperature rise, and speed, based on pressure and flow rates. The way in which the programs handle the compressor information, however, is different for the two compressors.

For the twin-screw compressor, all of the compressor maps display data with respect to pressure and flow rate: adiabatic efficiency, volumetric efficiency, temperature rise, and compressor speed. A grid was placed over the compressor maps to take data at regular intervals for different pressures and flow rates. Unfortunately, at the time of data collection, there were data for only three pressure ratings: 1.8, 2.0, and 2.2 atm. It was important to obtain data for both higher and lower pressures for this machine in order to get useful data about the fuel cell at high and low power. The solution was to take data from a larger compressor and scale it down. An Opcon 2089, with the same internal pressure ratio as the 1050 machine, was used to give extra data on the upper and lower ends of the pressure scale. Some estimations had to be made with some of the data, however, since the Opcon 2089 data did not cover all of the pressure/flow conditions that were necessary. After the data were determined for the Opcon 1050 machine, they were then normalized so that the compressor could be scaled up or down, if desired. The

results in this paper do not scale the twin-screw compressor at all, but the turbocompressor has been scaled up slightly to better fit the desired pressure characteristics.

When compressor data are needed in the program, they will most likely not be the data points that were taken directly from the compressor maps. Therefore, bicubic interpolation is used between the data points to give approximate values. Some data requests lie outside of the data range, so it was necessary to develop a system for approximating this information. The only functions in Matlab that extrapolate from data, rather than interpolate, are in the Simulink program. Therefore, a system was developed that finds the slope of the data at their edges and linearly extrapolates to the data point that is desired. To find the slope of the data at its edge, the function "linear" is called, which chooses two points from the data: one from the edge, and one slightly inside the edge. The slope of the line passing through these two points is used as the slope of the edge data. Then simple functions (all with names that are some variation of "linear") extend that line outside of the data range to where the desired data point is.

For the turbocompressor, handling of the data was done differently. Because of the way the data were distributed on the compressor map, the grid approach wasn't as useful as in the twin-screw compressor case. Only about half of the pressure and flow rate area on the map is used by the turbocompressor, primarily because of the surge line. As mentioned before, the surge line is the line where higher pressure at the same flow rate, or lower flow at the same pressure, would cause violent, intermittent, and damaging pressure fluctuations in the compressor. Use of a grid on the map would place about half of the points above the surge line—in the surge condition. Also, although the compressor speed map is drawn with the two axes as pressure and flow rate, the efficiency map of the compressor is drawn so that the two axes are flow rate and efficiency. Therefore, to use the same procedure as was used in the twin-screw compressor code would require the efficiency map to be redrawn with pressure and flow rate as the two axes.

The turbocompressor data were fewer than the twin-screw compressor data—only compressor speed and adiabatic efficiency were given. All other data had to be calculated. Instead of taking a grid over the compressor map data, each individual speed data point was taken at its pressure and flow rate and normalized. Unlike the twin-screw compressor, the turbocompressor data were scaled up slightly to help the compressor fit the fuel cell system better. The compressor map shows individual points and lines connecting them, rather than only lines as in the twin-screw maps. These data form a particular shape as shown in Figures 16 and 17. These figures are only a representation of what turbocompressor maps look like, and are not the actual compressor maps. Figure 16 shows the compressor surge line and the speed lines resulting from a particular pressure and air mass flow rate. Figure 17 shows the same speed lines as in Figure 16, only on an efficiency plot.

The program takes a requested data point P (pressure and flow rate) and finds the speed line that is directly above it. To find the speed line directly above the point, the flow rate of the point is checked with the flow rate range of the speed lines. For instance, since

point P's flow rate is within line A's flow rate range, the program checks the pressure on line A at point P's flow rate. It is lower than P, so the next line, line B, is checked. Finally line C is checked and is accepted. A perpendicular line is dropped from line C down to line B through point P. The length of the lines from point P to the two speed lines is calculated, and their ratio is used to interpolate the expected compressor speed at the requested point. Since the efficiency lines are based on the speed of the compressor as well, this same ratio can be used in the efficiency map data to calculate the efficiency of the requested data point (see Figure 17).

To find the efficiency of the compressor at point P, lines are drawn at several places between lines B and C on Figure 17. A point is placed on each of the connecting lines so that the ratio between the lengths of the lines on either side of the point is the same as the previous ratio. Figure 17 shows a line drawn from lines B and C, and point 1 is placed on that line at the correct ratio. So points 1 through 4 begin to outline a new efficiency line for the specific speed of the compressor where the requested point, point P, is located. The program continues to find more and more points at higher and higher flow rates between lines B and C until the flow rate of point P is encountered. So, as shown in the figure, data points 1 and 2 would be calculated first. Since these points do not include the range of point P's flow rate, data point 3 is calculated. Finally, data point 4 is calculated, and the program finds that the range of data points 3 and 4 include point P. Then linear interpolation is used between data points 3 and 4 to find the efficiency of point P.

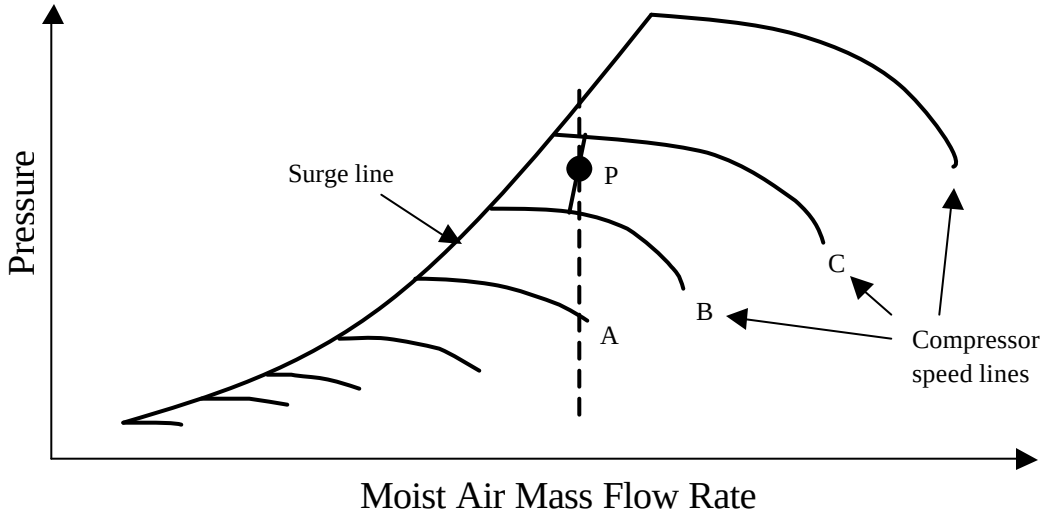

Figure 16: Shape of the turbocompressor speed map

Some special conditions can exist that are given special treatment in the program. For instance, if the requested data point is at a flow rate lower than the lowest flow rate point on the map, the program will automatically push the data up to the lowest flow rate point on the lowest speed line. If the requested data point is simply below the lowest speed line, the program will push the data point up to that line. Sometimes the requested data point is above the surge line. In this case, there are two possible solutions available to the

user. One is to keep the original pressure value and force the flow rate up to the point where it reaches the surge line. The other solution is to keep the original flow rate value and force the pressure down to the surge line. In this paper, the requested pressure takes precedence, and the flow rate is increased to where it intersects the surge line. In the program, errors will result if the requested pressure goes above the highest speed line. The only solution in this case is to lower the operating pressure of the compressor or to increase the size of the compressor. Note that airflow velocities are low enough so that the difference between static and stagnation temperatures is negligible.

The equation for determining the power (in Watts) required by either compressor is the following:

$$Pwr\_c = \frac{\left[\frac{mair1}{Ma} + \frac{mvap1}{Mh2o}\right] * 8314 * T1 * \frac{k}{k-1} * \left\{\left[\frac{P2}{P1}\right]^{\frac{k-1}{k}} - 1\right\}}{eff\_overall * nm}$$

Where:

mair1 = inlet dry air flow rate (kg/s)
Ma = molecular mass of air (kg/kmole)
mvap1 = inlet water vapor flow rate (kg/s)
Mh2o = molecular mass of water (kg/kmole)
T1 = ambient air temperature (K)
k = specific heat ratio
P2 = outlet pressure (atm)
P1 = inlet pressure (atm)
eff_overall = adiabatic efficiency
nm = efficiency of compressor motor

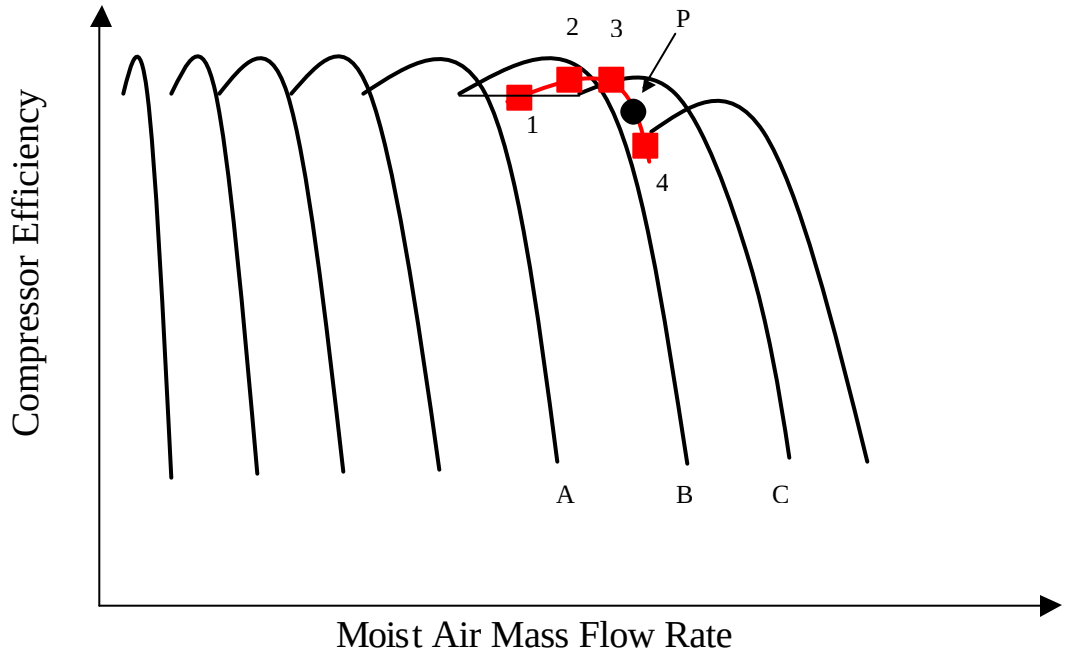

Figure 17: Shape of the turbocompressor efficiency map

Humidifier/Heat Exchanger Section

The humidifier and heat exchanger are the components directly before the fuel cell that condition the moist air stream to be at a temperature and humidity suitable for the fuel cell. These combined components use water from a reservoir for humidification and heat from the coolant loop exiting the fuel cell to vaporize the water and heat the moist air stream. The total heat rejection from the fuel cell determines the achievable output temperature and water vaporization for this component, and the working fuel cell temperature.

At first, the humidifier and heat exchanger use the fuel cell heat rejection data calculated in the initial calculations section to determine the limits to what temperature both the humidifier/heat exchanger and the fuel cell can achieve. Since the fuel cell heat rejection data is only a preliminary guess, it needs to be adjusted accordingly. Therefore, a loop is set in the program around the humidifier/heat exchanger and the fuel cell to recalculate the fuel cell heat rejection and the achievable temperatures of the two components. After the calculations for the humidifier are complete, the fuel cell calculations are made. From the fuel cell calculations, a better heat rejection value is obtained, the humidifier/heat exchanger equations are recalculated, and so on. Finally, once a 1% difference or less between consecutive fuel cell heat rejection values is reached, the loop is ended.

The humidifier and heat exchanger combination is one of the more idealized components in this system. If there is sufficient heat supplied from the fuel cell coolant, it is assumed that the humidifier is always able to reach the requested air humidity, and that the heat exchanger can warm the air stream to the desired average stack temperature. If there is insufficient heat to reach the fuel cell temperature, then the fuel cell average temperature is lowered to compensate until the humidifier can match the fuel cell and air stream temperatures. However, it is also assumed that the heat exchanger can only heat the moist air to 5°C cooler than the fuel cell coolant exit temperature. Normally, the coolant pump is set to keep a 10°C difference between the inlet and exit of the fuel cell coolant, so ideally the humidifier should be able to warm the air stream to the average fuel cell temperature. At low fuel cell power, however, the coolant pump may reach its minimum flow rate—a value that can be modified by the user. As a result, the 10°C difference across the fuel cell is no longer possible and the coolant temperature exiting the fuel cell drops, limiting the achievable temperature of the humidifier/heat exchanger. In these cases, the humidifier attempts to evaporate more water (up to 100% relative humidity) into the air so that it would have the desired humidity if it were at fuel cell temperature. This is done so that the fuel cell will not be dried out when the air from the humidifier warms up in the fuel cell.

The primary equation involved in the humidifier/heat exchanger is the equation for the amount of heat necessary to evaporate the required amount of water for humidification and warm the air stream to the requested temperature. The equation has 3 sections, dealing with the energy change in the air, water vapor, and liquid water added to the air stream:

$$Q\_needed = mair1 * cp\_air * (T3 - T2) + mvap1 * cp\_vap * (T3 - T2) + mh2o\_addh * [e\_vap + cp\_vap * (T3 - T\_h2o)]$$

Where:

mair1 = inlet dry air mass flow (kg/s)
cp_air = specific heat of air (J/kg-K)
T3 = exit temperature (K)
T2 = inlet temperature (K)
mvap1 = inlet water vapor mass flow (kg/s)
cp_vap = specific heat of water vapor (J/kg-K)
mh2o_addh = water added (kg/s)
e_vap = energy of water evaporation (J/kg)
T_h2o = temperature of added water (K)

Fuel Cell Section

The fuel cell section of the Matlab program calculates several parameters having to do with the fuel cell operation. These include the pressure drop across the fuel cell stack, the consumption of oxygen, the water produced, the voltage of the fuel cell, and the average partial pressure of oxygen. The most important aspect of this section is the voltage equation, also called the polarization curve:

$$V = 1.03 - 0.06 * \log_{10}(1000 * J) - (1.12 - 2.49 * 10^{-3} * T4) * J + 0.14 * \log_{10}(pO_2) + (T4 - 333) * 0.00041 * \log_{10}(1000 * J)$$

Where:

V = fuel cell voltage (V)
J = fuel cell current density (A/cm^2)
T4 = fuel cell average temperature (K)
pO_2 = average partial pressure of oxygen in fuel cell (atm abs)

This equation gives the voltage of each fuel cell based on the current load, temperature, and oxygen levels available. Note that higher voltages result from low current densities, high temperatures, and high oxygen partial pressure. The first term in the equation stands for the open circuit voltage of 1.03 Volts. The next term is the result of the Tafel equation, which is a measure of the overpotential of the fuel cell due to activation energy. Next is the ohmic loss term. The next term accounts for the effect of changes in oxygen partial pressure, derived from the Nernst equation. Finally, the last term is an additional temperature effect term, added to correct for the particular fuel cell system in use. No term is added that accounts for the drop-off voltage at high current density due to mass transfer limitations.

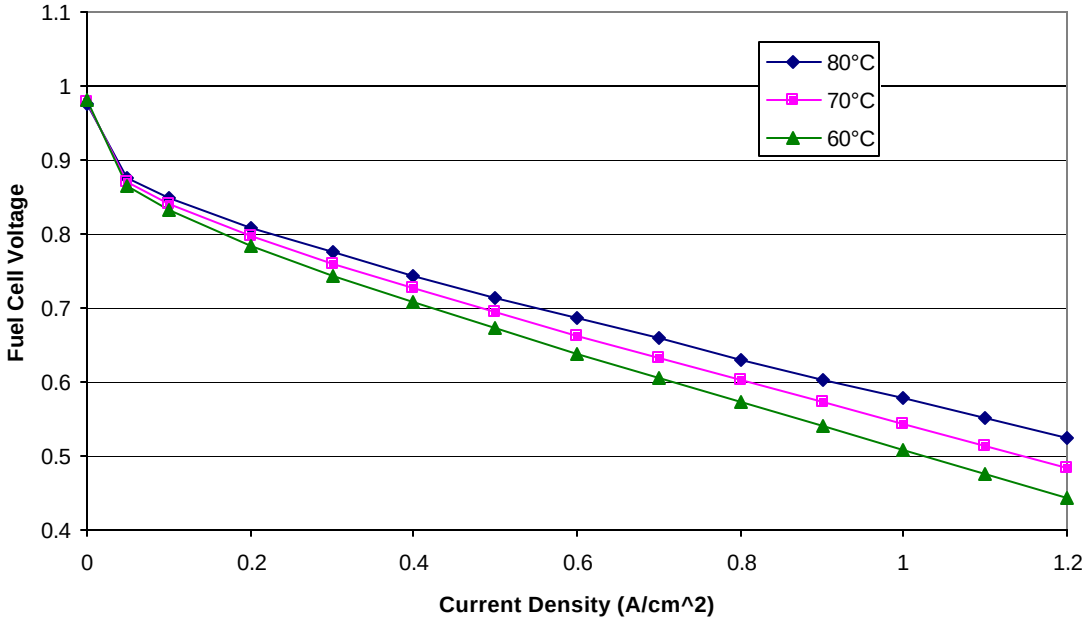

Figure 18: Fuel cell polarization curve with different temperatures at 3 atm

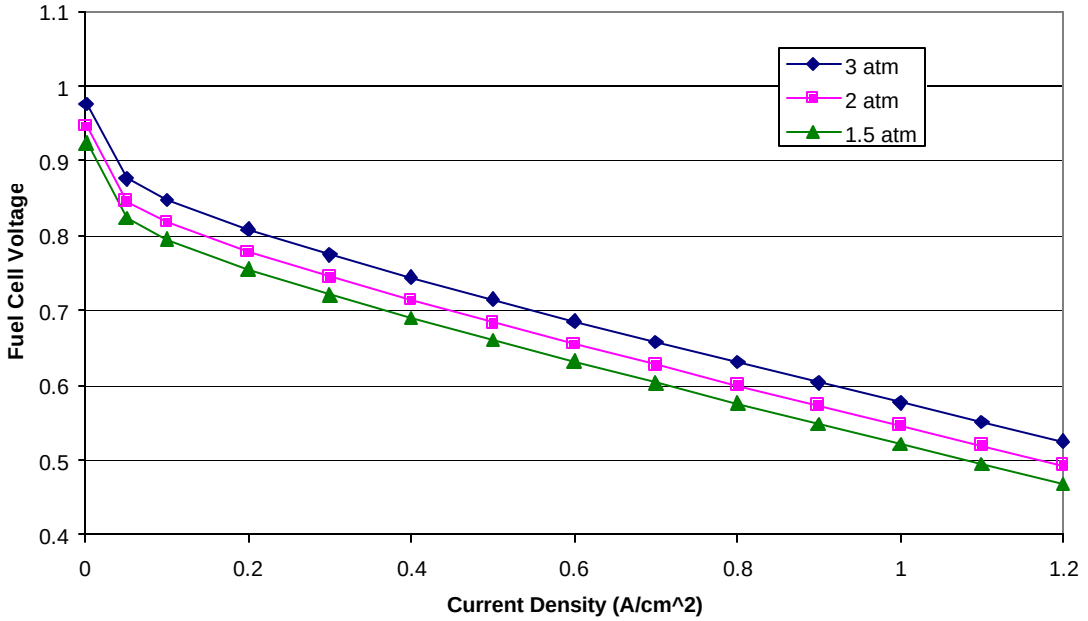

Figure 19: Fuel cell polarization curve with different pressures at 80°C

Figures 18 and 19 show the effects of temperature and pressure on the fuel cell voltage, respectively. For these curves there is a SR of 2, an inlet fuel cell RH of 75 percent, and an exit fuel cell RH of 100 percent. The change in partial pressure of oxygen due to consumption of oxygen in the fuel cell and humidity is reflected in the calculations. Also, a linear pressure drop over the fuel cell is assumed going from 0.01 atm at almost 0 A/cm^2 to 0.4 atm at 1.2 A/cm^2. Note that the topmost curve in both figures is the same (P

= 3 atm and T = 353 K). The figures show that a temperature drop of approximately 20°C will result in about a 0.1 Volt loss per cell for the highest point in the operating range, but effects at lower operating ranges shows less of an effect. Reducing the pressure from 3 atm to 1.5 atm has less of an effect than the temperature, but it is consistent throughout the range of the fuel cell load.

Radiator Section

The radiator section, as mentioned before, is based on a single-pass, crossflow heat exchanger with both fluids unmixed. The NTU method was utilized in the calculations. A fan and a coolant bypass valve help to regulate how much of the heat is rejected through the radiator to the environment. The fan enhances the heat rejection ability of the radiator while the bypass valve redirects the coolant so that less heat is rejected from the radiator. The program sets the inlet coolant temperature to be 5°C less than the average fuel cell temperature, so this component calculates whatever percentage of fan or bypass valve usage is needed.

Expander Section

The expander section is similar to the compressor section in that it is based on the same technology as the compressor—turboexpander for the turbocompressor, and twin-screw expander for the twin-screw compressor. However, unlike the compressor section, only the twin-screw expander derives its calculations from real data. Also, this data is the same as the compressor data since no data were available specifically for the expander. The compressor data make a reasonable approximation of the expander characteristics because they are positive displacement machines. However, for the turboexpander, general centrifugal expander equations were used to calculate the expected output power and airflow temperature based on a fixed expander efficiency.

For the twin-screw compressor, the system of finding the desired data point in the expander characteristic data is similar to the compressor system. If the requested data point is within the expander data, bicubic interpolation is used to find the desired data. If the requested point is outside of the expander data, the program uses linear extrapolation by using the slope of the data at its edge.

System Output and Plots

This section simply gives output such as efficiency and net power, and offers the option of plotting that output.

Chapter 5: Results

In order to produce the most efficient fuel cell system, it is important to learn how much of an effect the operating parameters will have on the system. The operating parameters of a fuel cell vehicle are constantly changing due to start-up, ambient conditions, and compressor control strategies. These results help to define what to expect from a fuel cell system in a variety of operating conditions and parameters. These conditions include ideal operation and conditions (highest expected efficiency), minimum mass flow operation, varied ambient conditions, varied stack temperatures, and changes in operating pressures. Ideal conditions are defined as:

Ambient temp. = 25°C
Ambient RH = 50 %
Ambient pressure = 1 bar
Fuel cell stack temp. = 80°C
Stack inlet RH = 75 %

In the following situations, if certain conditions are not specified, the above ideal conditions apply.

The following figures are a result of the program finding the steady state results of the fuel cell system at many different fuel cell current load values. No transients are considered in the results. The majority of the results are generated using curve fits to the pressure lines found by the program that give the highest system efficiency (Figure 15b), and a predetermined operating line for the airflow. However, some data were generated using the most efficient pressure and air mass flow operating lines. For these lines, each point on the operating lines were determined by the highest steady-state system efficiency at that fuel cell load (Figure 15a). As a result, the lines look somewhat irregular and choppy. The reason for this could be the irregularities in the data transfer from the maps to the program. Figures 20 and 21 are the pressure lines for the highest efficiency operation of both the twin-screw and the turbocompressor fuel cell systems.

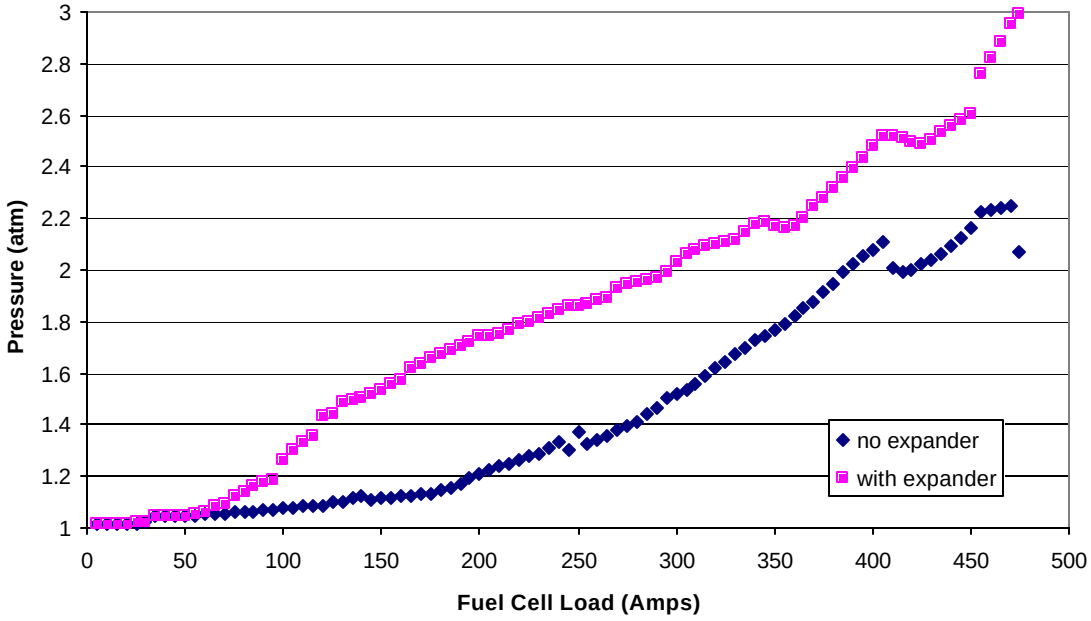

Figure 20: Highest System Efficiency Pressure Lines with Turbocompressor

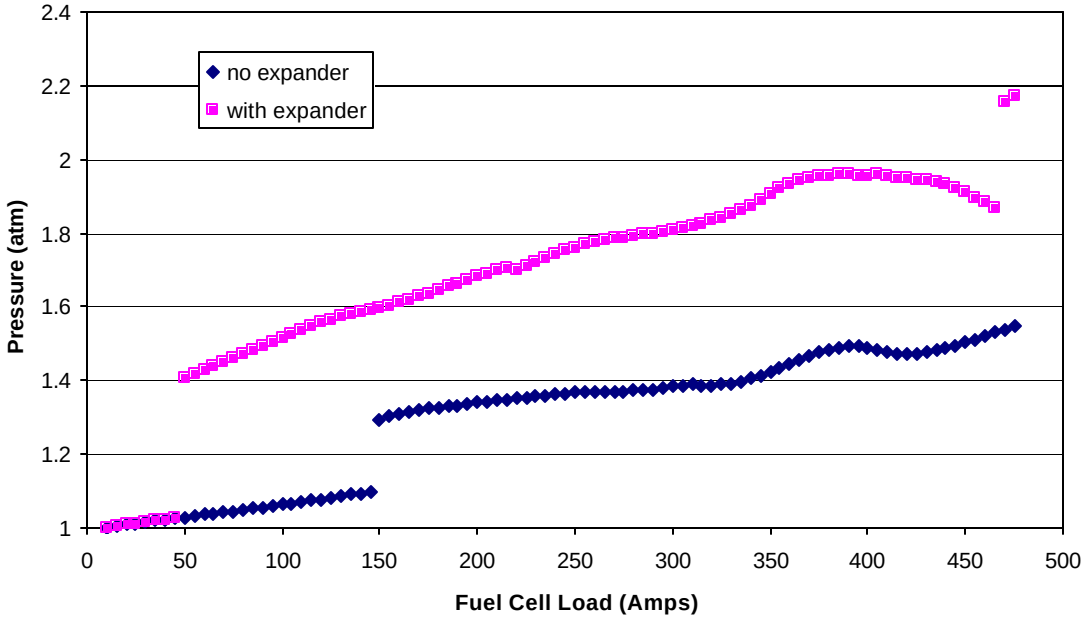

Figure 21: Highest System Efficiency Pressure Lines with Twin-screw Compressor

As is expected, the pressure lines with the expander go higher for the best efficiency, since the expander can recapture some of the power required for the compressor. Also, as expected, the pressure schedule for the turbocompressor is much higher than for the twin-screw. The pressure lines for the turbocompressor are relatively smooth, but the twin-screw compressor lines have discontinuities where the pressure jumps a large

amount. The first jump in the twin-screw pressure line with the expander could be explained as being the point where the expander begins to have its effect. Before the jump, the pressure line supplies just enough pressure to overcome the pressure drop over the system, but then it leaps far above that point, perhaps to take advantage of the expander at higher pressures. Another possibility could be that the compressor encountered the limit of airflow (shown in Figure 23), and forced it to move to a higher pressure for highest efficiency. The second jump in the line, and the jump in the line without the expander are more difficult to explain, however. In the output of the programs, it was noted that there were peaks in efficiency at different pressures and fuel cell loads. As the fuel cell load increased, the peaks in efficiency would begin to shift from one pressure to another until one peak would overcome the other, showing up as leaps in the pressure line. One explanation could be irregularities in the compressor map data. Another explanation could be that the characteristics of the twin-screw compressor and expander, along with the changing efficiency of the fuel cell with pressure changes, cause these discontinuities to occur. Each component has its maximum efficiency at some particular mass flow and pressure. As these interact, one efficiency point may become more dominant and another less. The likelihood that such a choppy pressure schedule would actually be put into use is low, however, so curve fits of the highest efficiency pressure lines are made.

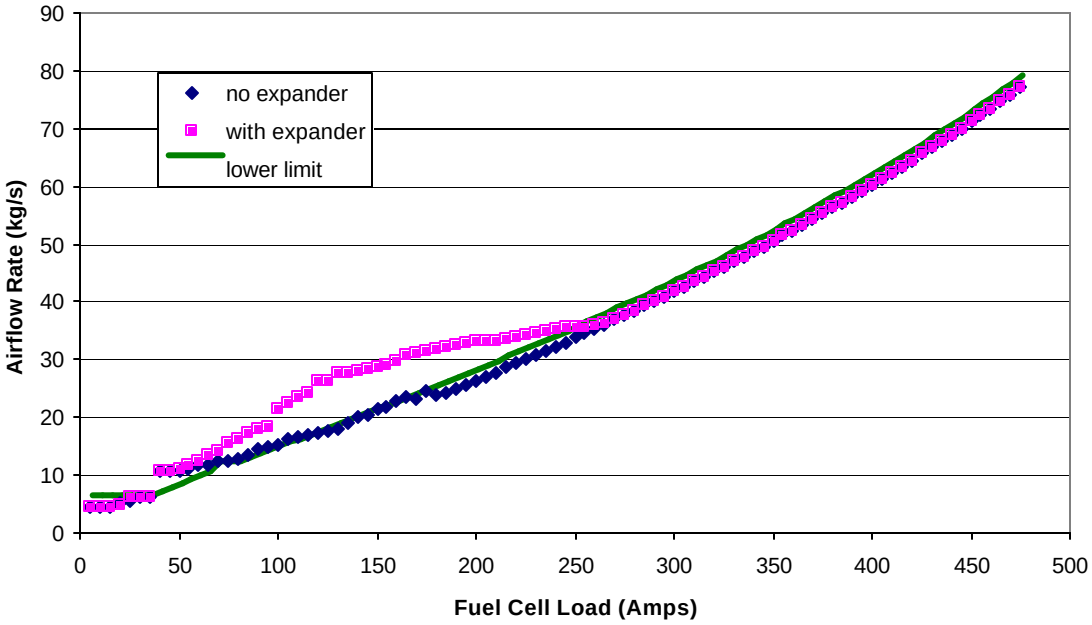

Figure 22: Airflow rate of fuel cell system with turbocompressor

The airflow rate schedules for the two compressors are shown in Figures 22 and 23. The high efficiency lines—"no expander" and "with expander"—are compared with the lower limit line used in the fuel cell simulator program. A lower limit was placed on the SR of oxygen in consideration of stack health. The limit imposed is a minimum SR of 2 at low fuel cell load and 2.5 at high fuel cell load. Between low and high load, the minimum SR is determined by a quadratic equation that gradually increases the minimum from 2 to 2.5.

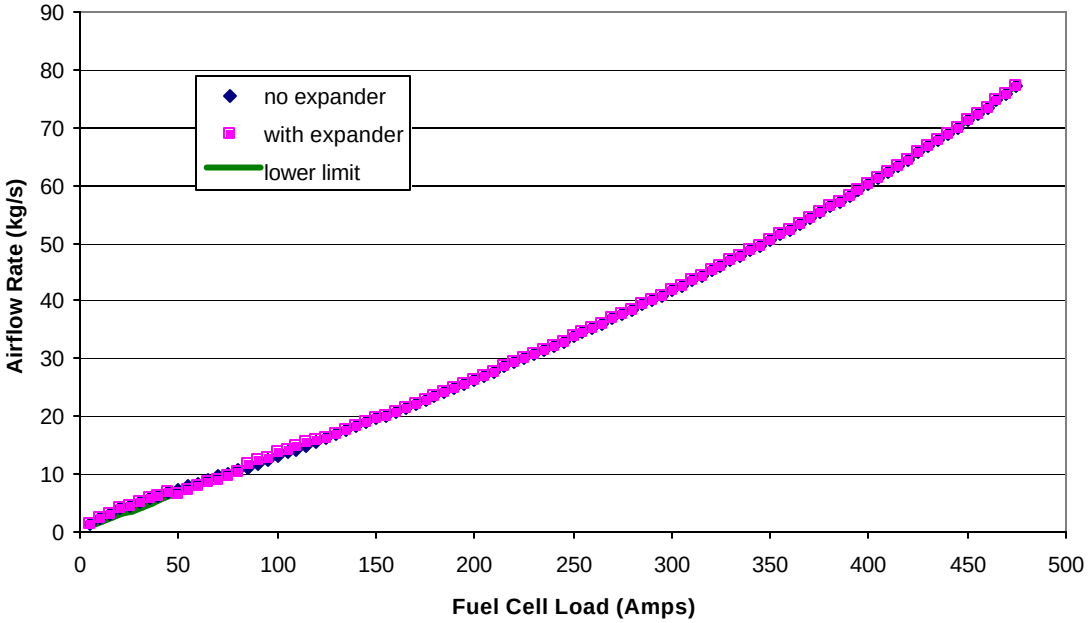

Figure 23: Airflow rate of fuel cell system with twin-screw compressor

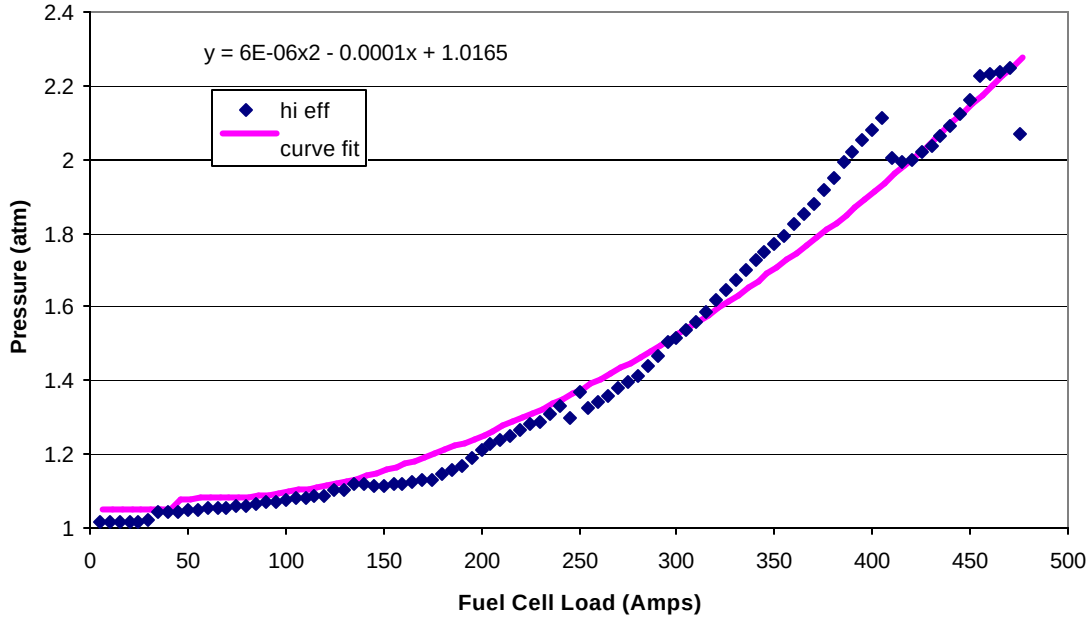

Figure 24: Pressure Lines for Turbocompressor Without Expander

As shown, the twin-screw compressor has very little difference between the systems with and without an expander and the simulator line. For most of the fuel cell load, the lower limit of SR is used. The turbocompressor, however, has significant differences between the systems with and without expanders. This follows from the compressor map since an

expander can extract more power at higher pressures, and at low flow rates an increase in pressure requires a fairly substantial increase in flow rate to avoid the surge zone.

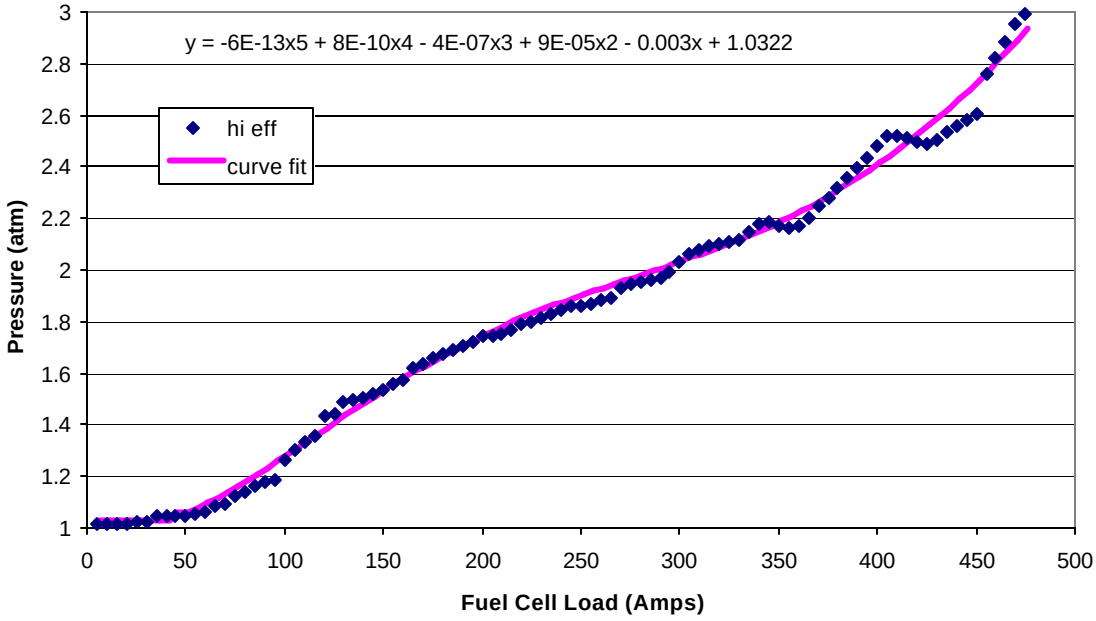

Figure 25: Pressure Lines for Turbocompressor With Expander

The curve fit for the turbocompressor line without an expander, shown in Figure 24, is taken from a 2nd order polynomial curve fit. The equation to the curve fit is on the figure. This pressure line was modified from the shown 2nd order trendline equation so that at low fuel cell amperage it would not go below the minimum pressure capable by the compressor. Also, the line was lowered at high fuel cell load so that the pressure would not go above that of the highest efficiency line and pull the efficiency down.

The curve fit for the turbocompressor line with an expander, shown in Figure 25, is taken from a 5th order polynomial curve fit. The equation for the curve fit is shown on the figure. The curve fit was modified from the shown 5th order trendline equation so that at low fuel cell amperage it would not go below the minimum pressure supplied by the compressor. The effects of the expander can be seen after the 50-amp point, when the pressure operating line begins to increase compared to the pressure operating line of the system without an expander.

The curve fit for the twin-screw compressor pressure line without an expander, shown in Figure 26, is taken from a 3rd order polynomial curve fit. The equation for the curve fit is shown on the figure. This pressure line was modified from the original 3rd order trendline to remain above ambient pressure at low fuel cell loads and to better follow the higher fuel cell load pressure line points.

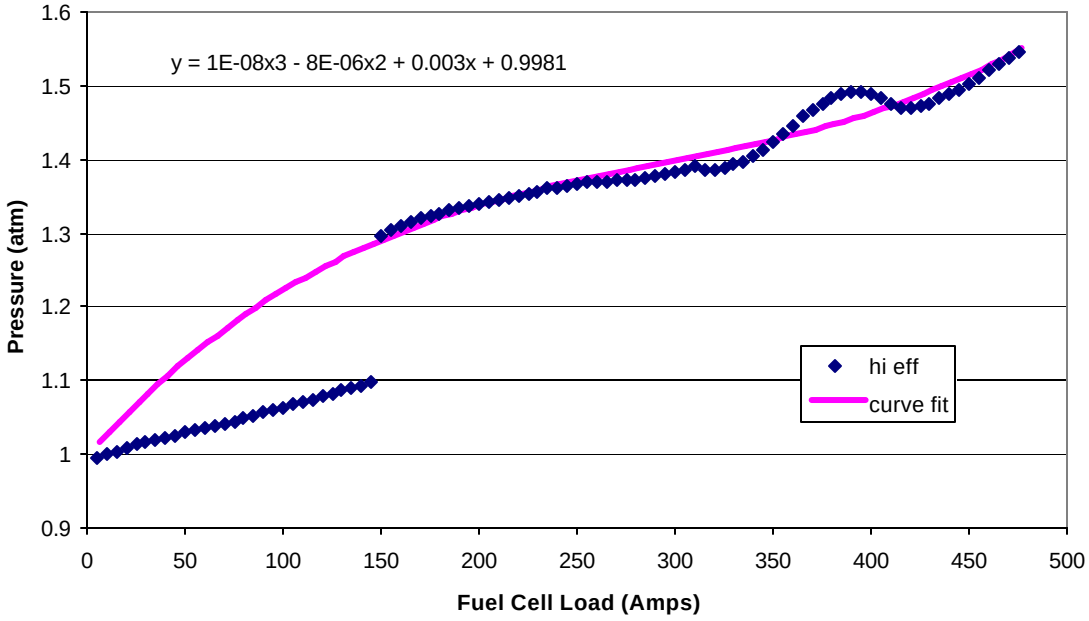

Figure 26: Pressure lines for twin-screw without expander

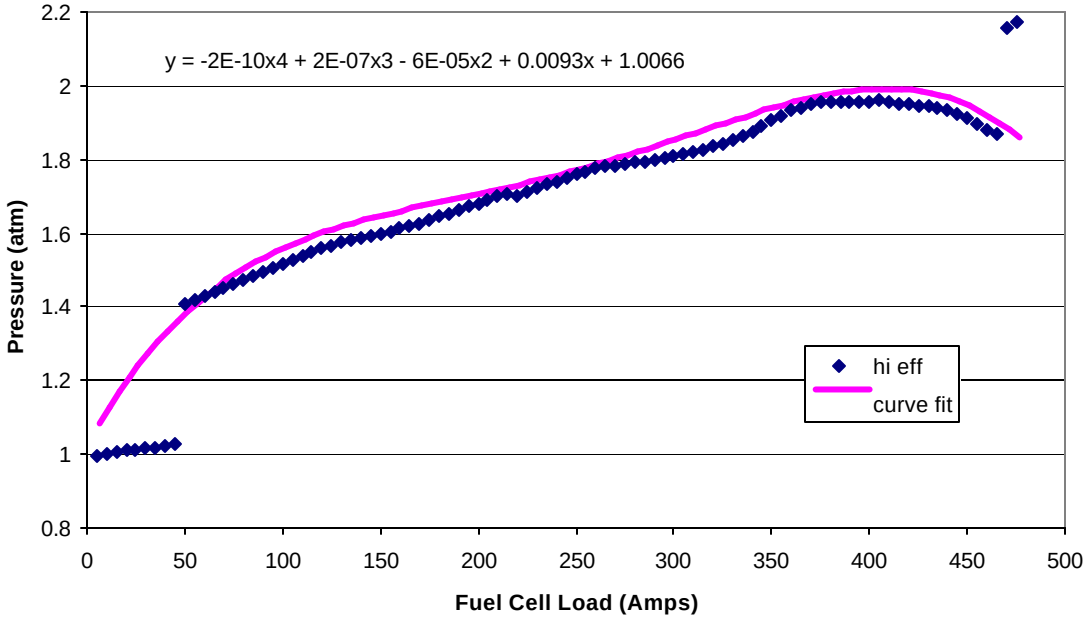

Figure 27: Pressure lines for the twin-screw compressor with an expander

The curve fit for the twin-screw compressor line with an expander shown in Figure 27, is taken from a 4th order polynomial curve fit. The equation for the curve fit is shown on the figure. This pressure line was modified from the original 4th order trendline to remain above the ambient pressure at low fuel cell load and to ignore the two data points at the highest fuel cell load that jump above the previous pressure points.

Figure 28 shows the compressor operating lines together. Note that the two twin-screw pressure lines are highest at the low fuel cell loads, which will be shown to improve water balance over the turbocompressor later.

Figure 29 compares the difference of efficiency between the highest system efficiency pressure line and the curve fit of that line for the twin-screw compressor without an expander. The comparisons between the high efficiency line and the curve fit for the other system configurations were a better match, so this plot shows the worst of them all. The largest efficiency difference between the two lines is 1.3% at the lowest fuel cell load current rating. However, the difference between the two lines becomes better at higher fuel cell load currents, and they soon become indistinguishable.

The system efficiency with the various compressor systems is shown in Figure 30. This figure shows that there is very little difference in the systems when the expander is absent. Although the turbocompressor by itself has about a 1 percent advantage at low fuel cell loads over the twin-screw compressor without an expander, the efficiency lines cross at higher fuel cell powers due to the high-pressure requirement for the turbocompressor. When the expander is introduced, however, the turbocompressor can operate at those higher pressures with much less of a power penalty. This increase in efficiency has a dramatic effect on the output power, as shown in Figure 31. The system including the turbocompressor with an expander has a 4 kW advantage over the system with the twin-screw compressor and expander at the highest fuel cell load.

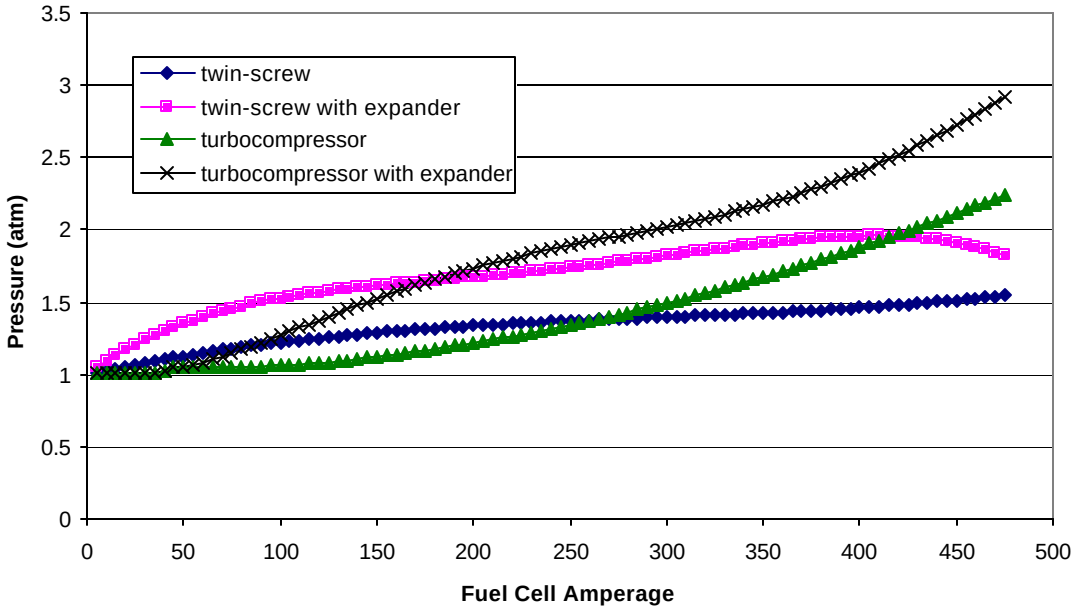

Figure 28: Comparison of compressor pressure curve fit operating lines

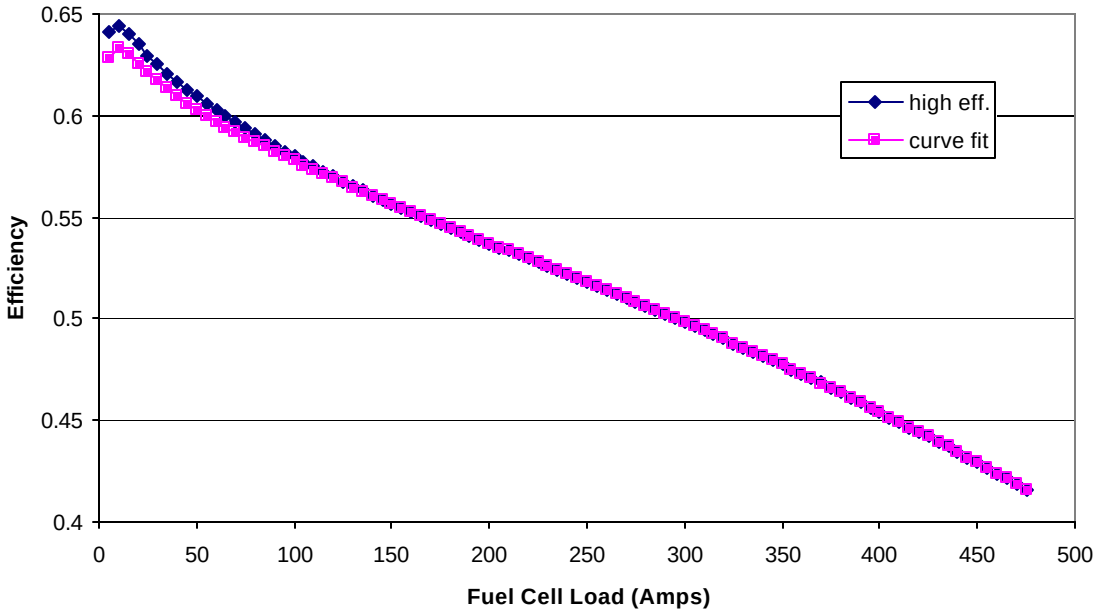

Figure 29: Comparison between highest system efficiency pressure line and curve fit for twin-screw compressor without an expander

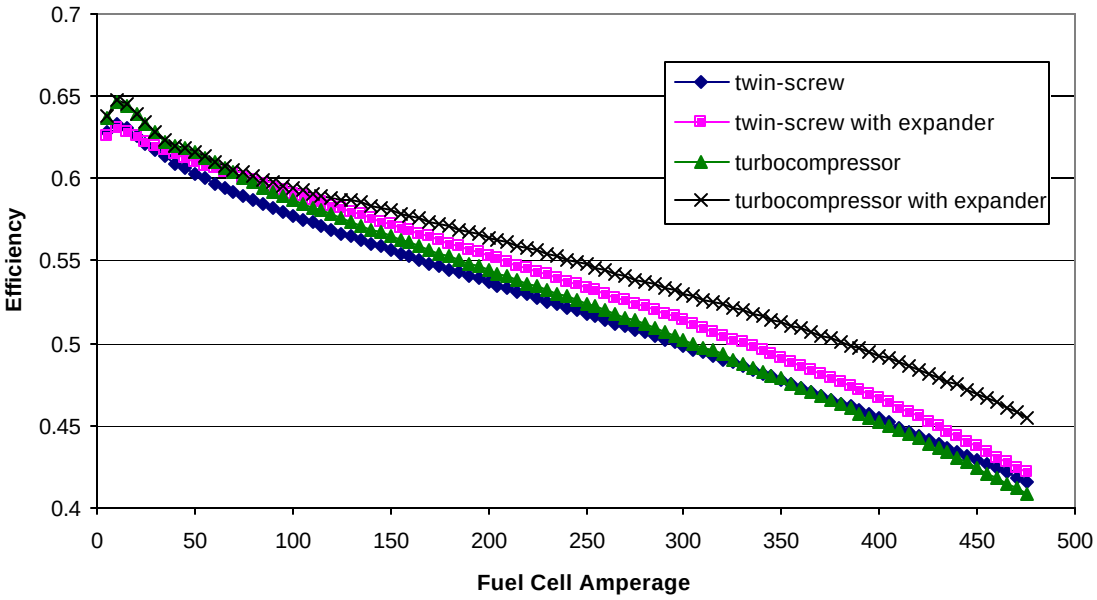

Figure 30: System efficiency for compressor pressure operating line curve fits

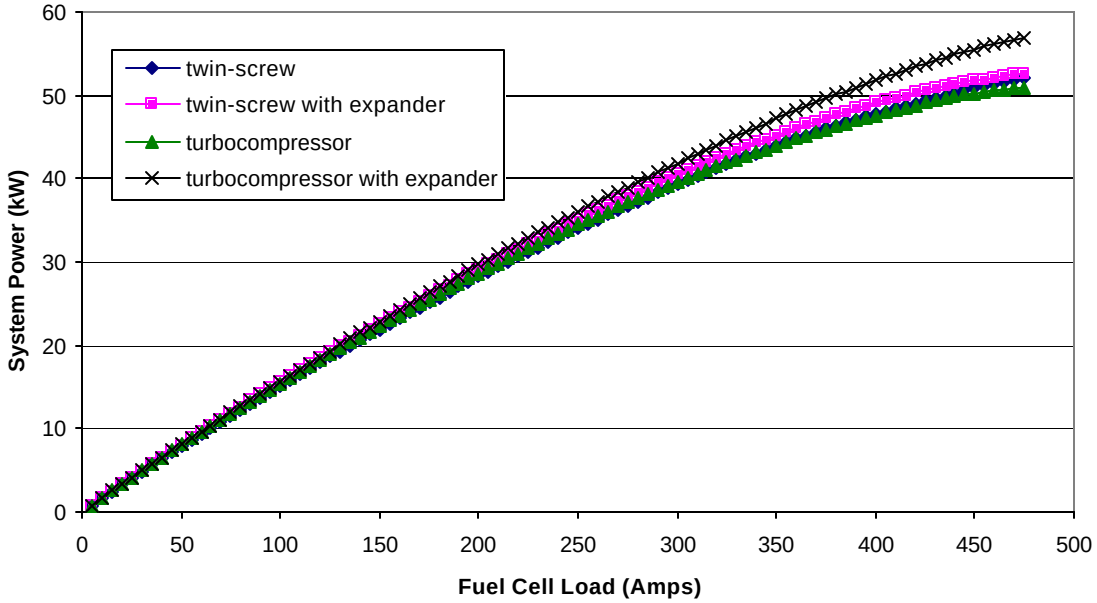

Figure 31: Net system power for compressor pressure operating line curve fits

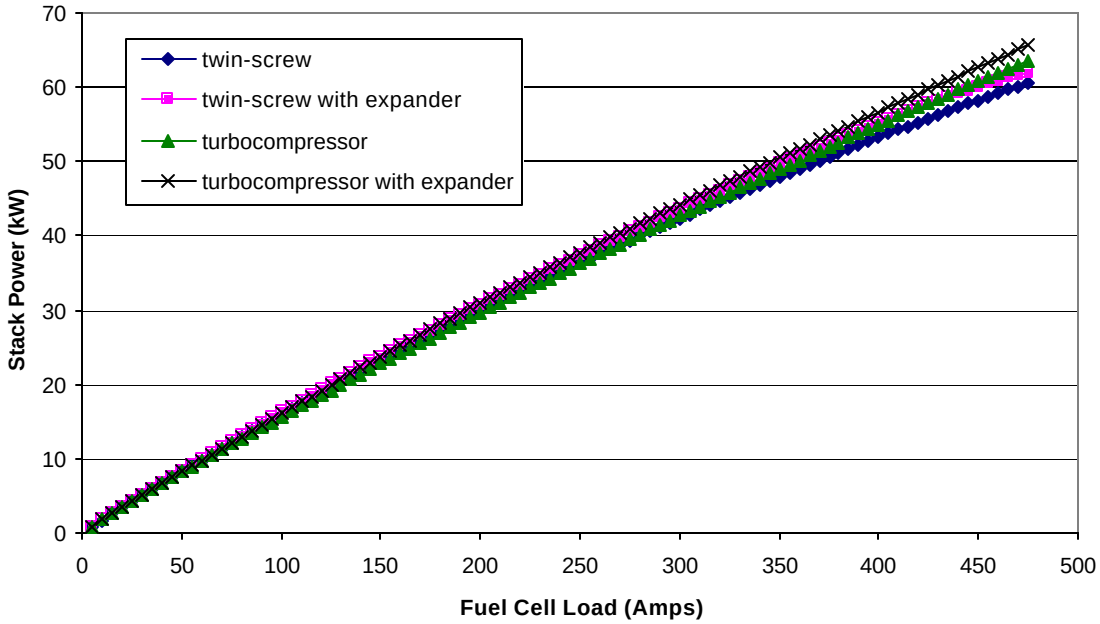

Figure 32: Stack power for compressor pressure operating line curve fits

Figure 32 shows the stack power for the different compressor configurations. The pressure of the system is the primary driver of the differences between the four curves, so the turbocompressor has the highest power ratings at the highest fuel cell load. When Figure 32 is compared to Figure 31, the effect of the parasitics can be seen on the power curves; the curves in Figure 31 begin to curve downward.

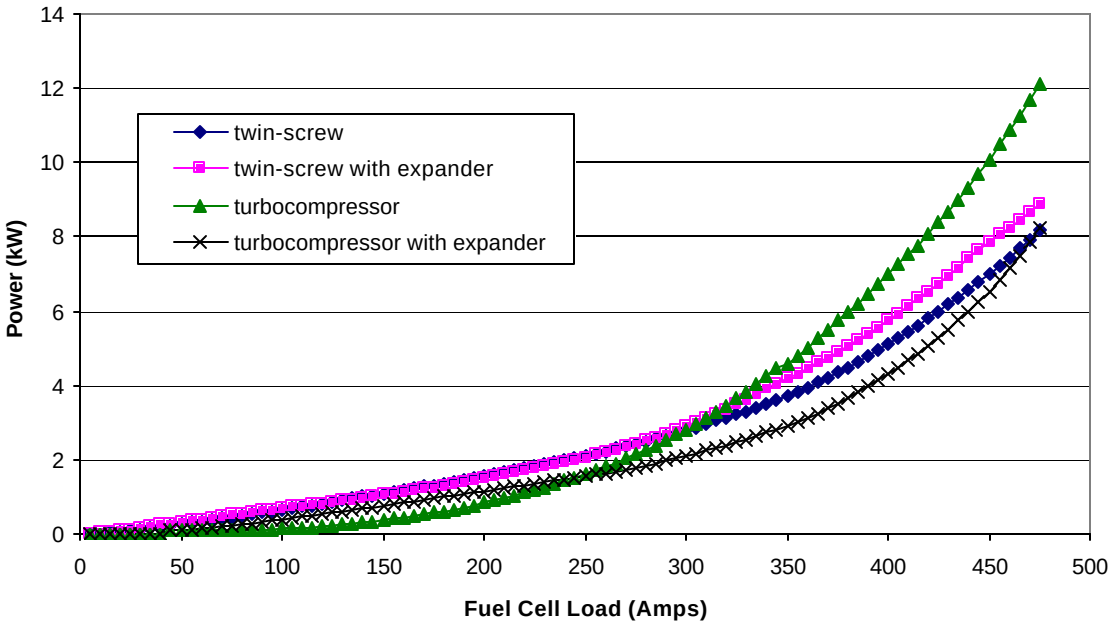

Figure 33: Compressor power for compressor pressure operating line curve fits

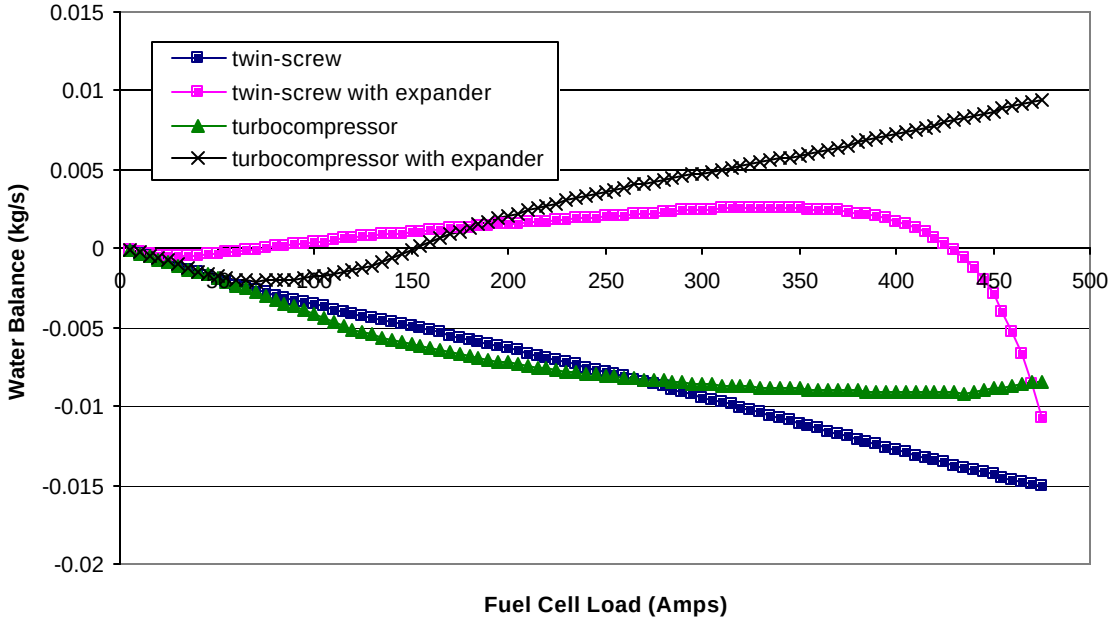

Figure 34: Water balance for compressor pressure operating line curve fits

The greatest parasitic, compressor power, is shown in Figure 33. At low power, the twin-screw is able to produce higher pressures, and so the twin-screw power is larger at low fuel cell loads. However, the turbocompressor can produce much higher pressures at high load, which accounts for the large power required for the turbocompressor without an expander at high load. The turbocompressor coupled with an expander makes a large difference in the power curve, primarily because of the high pressures that the

turbocompressor can produce at high efficiency. The twin-screw expander seems to make less of a difference than the turboexpander. This is primarily because of the lower pressure of the twin-screw; there is less power available to capture.

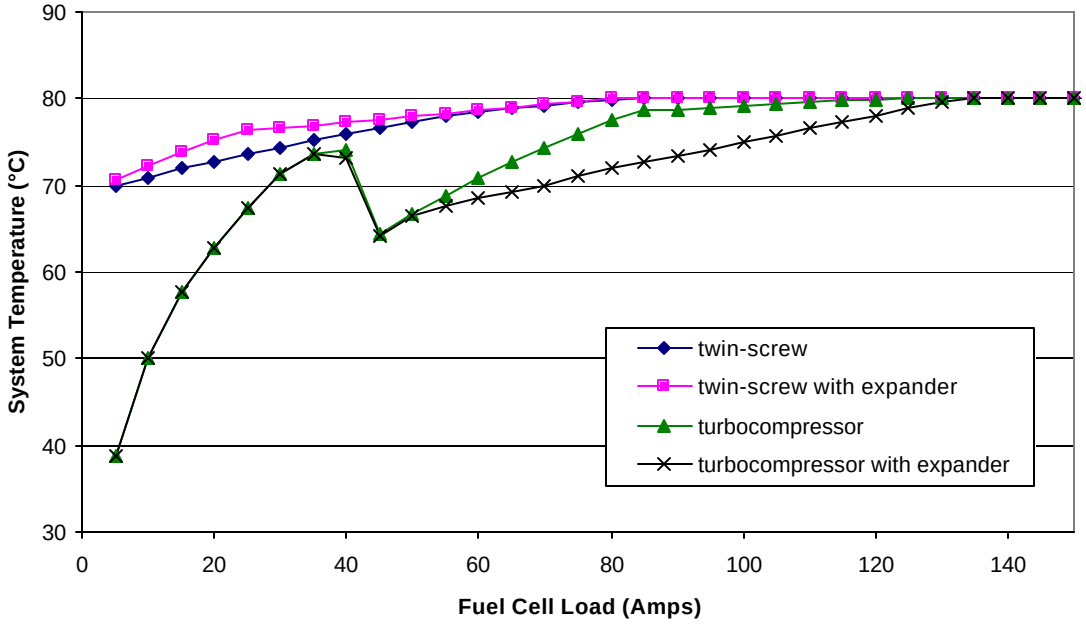

Figure 35: Stack temperature for compressor pressure operating line curve fits

The water balance for the system with all of the different compressor configurations is shown in Figure 34. The water balance is closely related to the pressure operating line for the compressors. This is apparent by comparing the twin-screw and turbocompressor operating pressure lines. With or without the expander, the turbocompressor starts with a lower pressure than the twin-screw and therefore has a lower water balance. This reverses at higher fuel cell current loads, however, because the pressure increases for the turbocompressor. The pressure operating line for the twin-screw compressor with an expander decreases at higher fuel cell loads, and this lower pressure has an effect on the water balance, as shown in Figure 34. There is a dramatic drop-off in water balance, even though the system efficiency is best on that line.

There is a dramatic difference between the water balance for the configurations with and without expanders. This is because of the temperature decrease that results from extracting work from the air. Without the expander, the temperature of the stream remains the same after it is throttled from high to low pressure, according to the 1st law equation. The drop in temperature resulting from the expander reduces the ability of the air to hold water and therefore precipitates condensation.

Figure 35 shows the stack temperature for the low fuel cell load range of the system. Because the humidifier/heat exchanger uses only the waste heat from the fuel cell to heat and humidify the air, it is possible that there is not enough heat to reach the desired stack temperature at certain fuel cell loads. In the figure, the desired stack temperature is 80°C, but the system is incapable of maintaining that temperature for most of the low power

range. The main differences between the stack temperature lines in this figure are caused by the differences in mass flow. The turbocompressor has a much higher mass flow at low fuel cell loads so that it can remain efficient, while the twin-screw can have much lower mass flows. The higher mass flow from the turbocompressor requires a greater amount of heat to warm and humidify it, so the temperature of the stack needs to be lowered accordingly.

The reason for the large kink in the turbocompressor stack temperature line in Figure 35 is the strange pressure operating line that was chosen for it. The compressor map for the turbocompressor is very limited at low flows and has a strange form at those low flows. As a result, the compressor pressure line takes a strange leap in mass flow and pressure as the fuel cell load increases. Future work could eliminate this kink by gathering more data about the turbocompressor at low flow. The mass flow increase dominates over the pressure increase, however, and causes the stack temperature to lower. If only pressure were taken into account, the pressure increase would lower the amount of water evaporation necessary to humidify the air, thus raising the achievable stack temperature. Another place where the mass flow of air dominates the fuel cell stack temperature is in the difference between the turbocompressor system stack temperature lines. The system with an expander has a lower stack temperature line than that of the system without an expander. The higher pressure operating line for the system with an expander would require less water evaporation energy for humidification, but the mass flow difference is greater. For the twin-screw system, the opposite is the case; pressure dominates over any mass flow differences, and the expander system has a higher stack temperature.

Minimum mass flow

At low fuel cell power, if the compressor is allowed to approach or reach a stop, it may take some time for it to return to full speed if a sudden demand is placed on it. In an automotive application, a stopped compressor would prevent a pure fuel cell vehicle from accelerating as fast as it could since the fuel cell would be air-starved for a short period of time. Some time would be required to allow the compressor to speed up again. Since performance is at a premium for automotive applications, anything reasonable that can be done to fix this problem should be considered. One possible solution is to maintain an elevated minimum flow rate for the compressor even at low power. This would keep the speed of the compressor up and would keep a high SR of air within the fuel cell, ready for use in the event of a large power demand.

There are a few drawbacks to this solution, however. With higher airflows at low fuel cell power, the efficiency will suffer. Depending on the minimum flow rate that is required of the compressor, the efficiency can drop to near zero at low fuel cell power. Also, the system temperature is difficult to maintain at low fuel cell power.

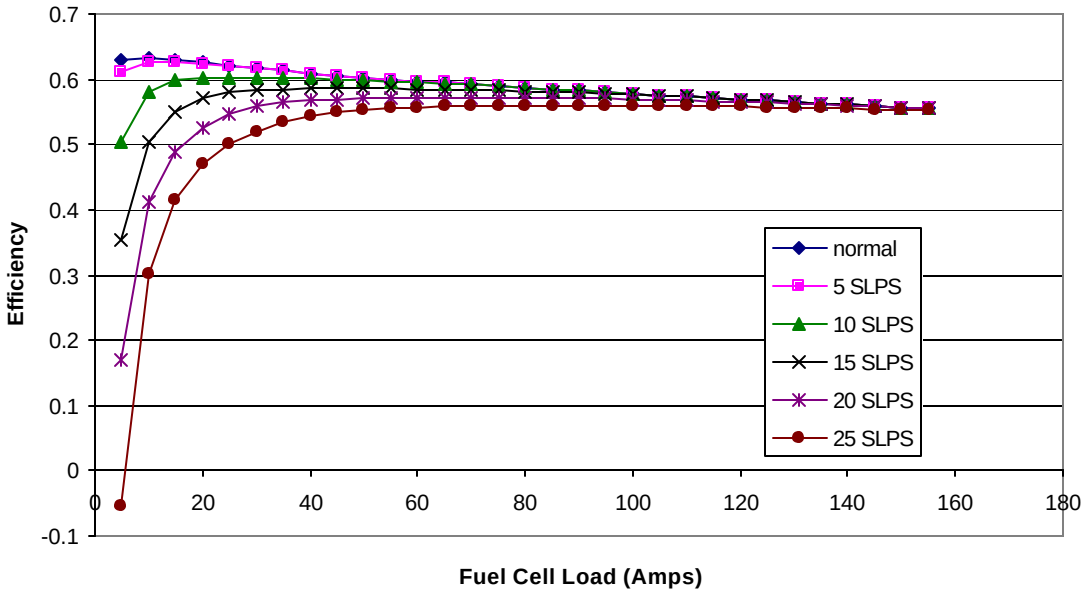

Figure 36: System efficiency using twin-screw compressor without an expander at several minimum airflow rates

Figures 36 through 39 show the effects of minimum flow rates on systems using compressors with and without expanders. These figures show a close up of the region of fuel cell current where the minimum flow rates have an effect. For the twin-screw compressor, the largest minimum flow rate is only up to 20 standard liters per second (SLPS), whereas the turbocompressor is up to 30 SLPS. The reason for this is the range of the compressor speeds and flow capacities. The twin-screw compressor is a smaller machine and 20 SLPS is about the same percentage of its maximum flow rate as 30 SLPS is for the turbocompressor. The efficiency plots show this similarity, since at the highest flow rates the efficiencies are at similar positions.

At low power and high mass flow, the expander has little effect on system efficiency. The small differences in efficiency that are shown between the systems with and without expanders are a result of the differences in the operating pressure lines. For example, the twin-screw system with an expander has a higher pressure operating line than the system without the expander. Since the expander does little at this low fuel cell power, the system with an expander has slightly lower efficiency at low loads and high minimum mass flows. In fact, in a real system that takes flow losses into account, an expander may reduce the efficiency even more at low power.

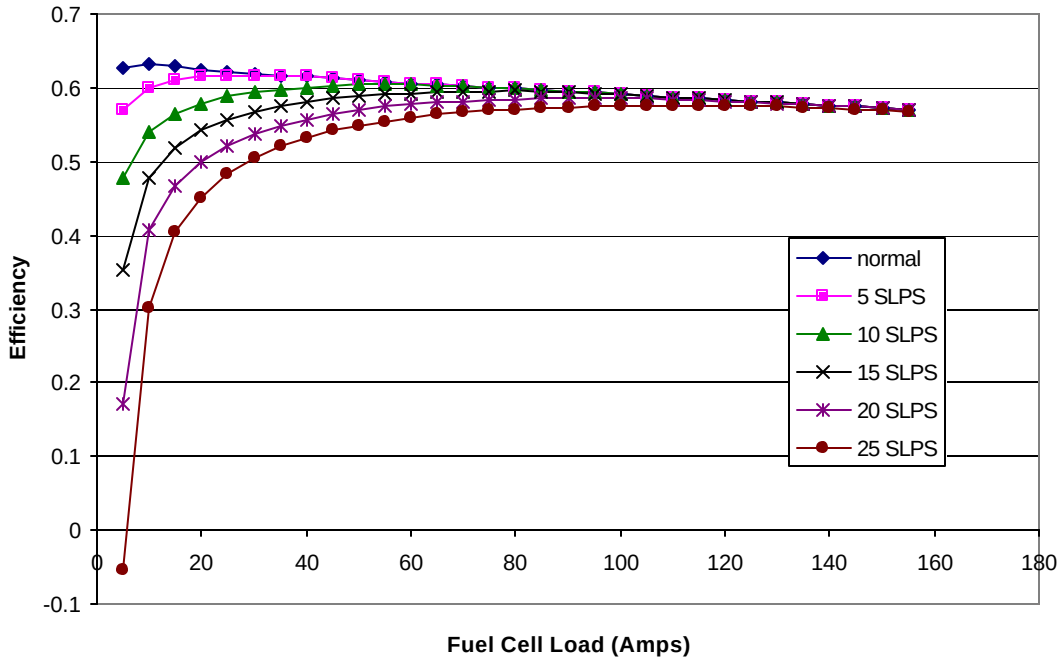

Figure 37: System efficiency using twin-screw compressor with an expander at several minimum flow rates

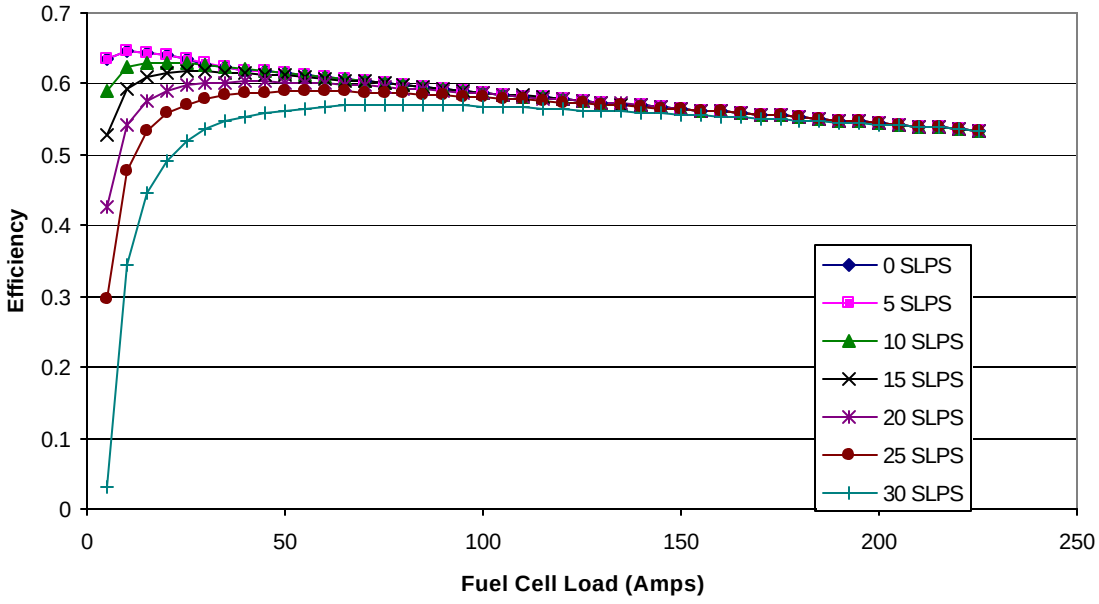

Figure 38: System efficiency using the turbocompressor without an expander at several minimum airflow rates

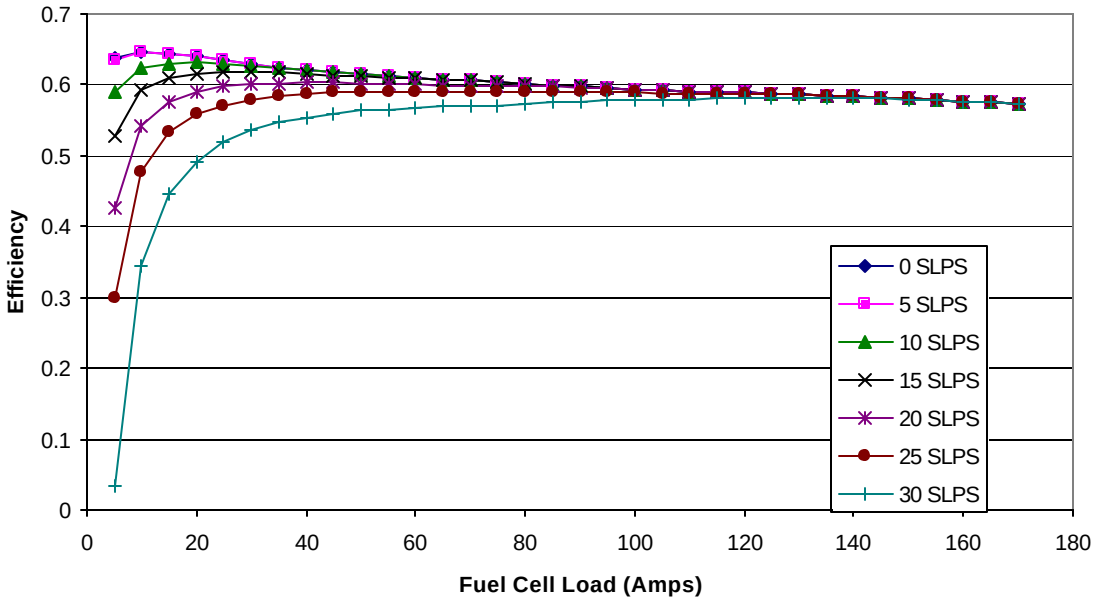

Figure 39: System efficiency using the turbocompressor with an expander at several minimum airflow rates

Figures 40 through 43 show the water balance and the compressor power used by the compressor systems at different airflow rates. The twin-screw compressor shows significant changes in water balance as a result of the changes in airflow at low power. These changes are not nearly as noticeable in the turbocompressor. The main reason for the difference is the pressure, especially with the twin-screw compressor/expander. There appears to be a lower bound of the water balance that is set by the air temperature pressure and flow. The turbocompressor systems and the twin-screw system without an expander seem to follow this lower limit closely. The large jump in pressure at low loads with the twin-screw compressor/expander changes the water balance characteristics enough to pull the water balance off of that lower bound.

Increasing the airflow has a greater effect on the twin-screw compressor power than on the turbocompressor, as shown in the figures. The twin-screw compressor has an increase of approximately 900 Watts at low power and 25 SLPS flow, while the turbocompressor only has an increase of about 450 Watts. This difference is attributed to the higher turbocompressor efficiency.

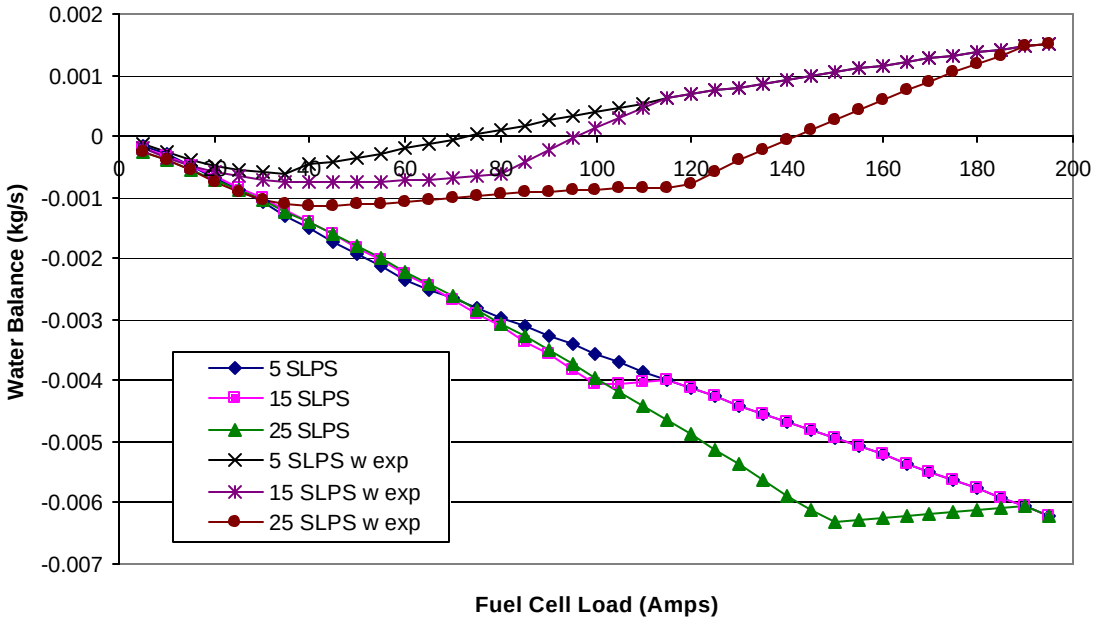

Figure 40: Water balance for twin-screw compressor at several minimum airflow rates

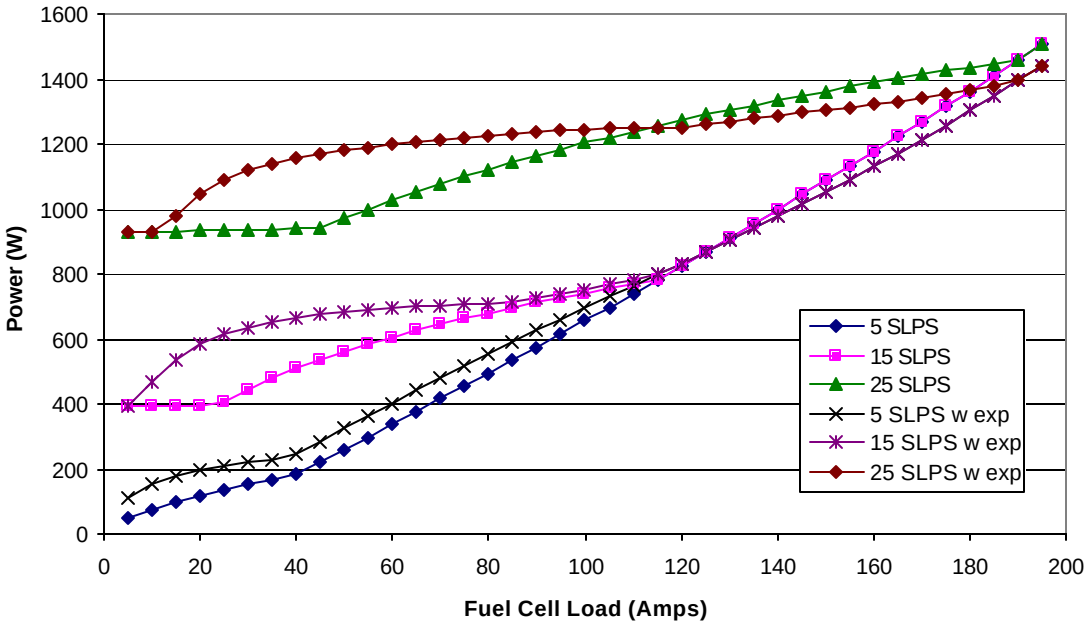

Figure 41: Twin-screw compressor power at several minimum airflow rates

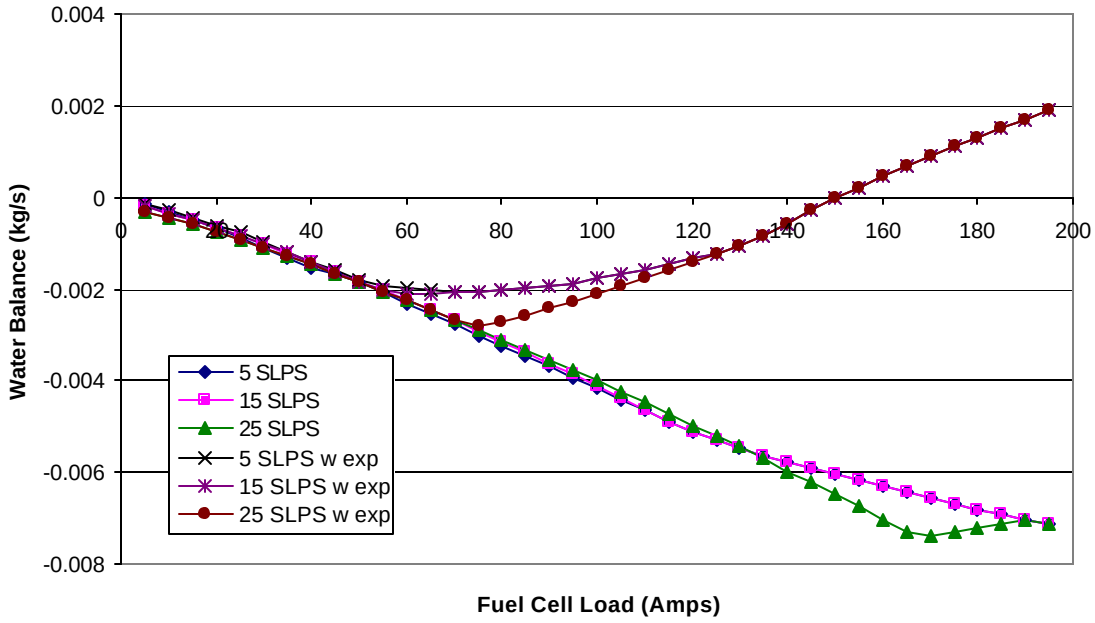

Figure 42: Water balance for turbocompressor at several minimum airflow rates

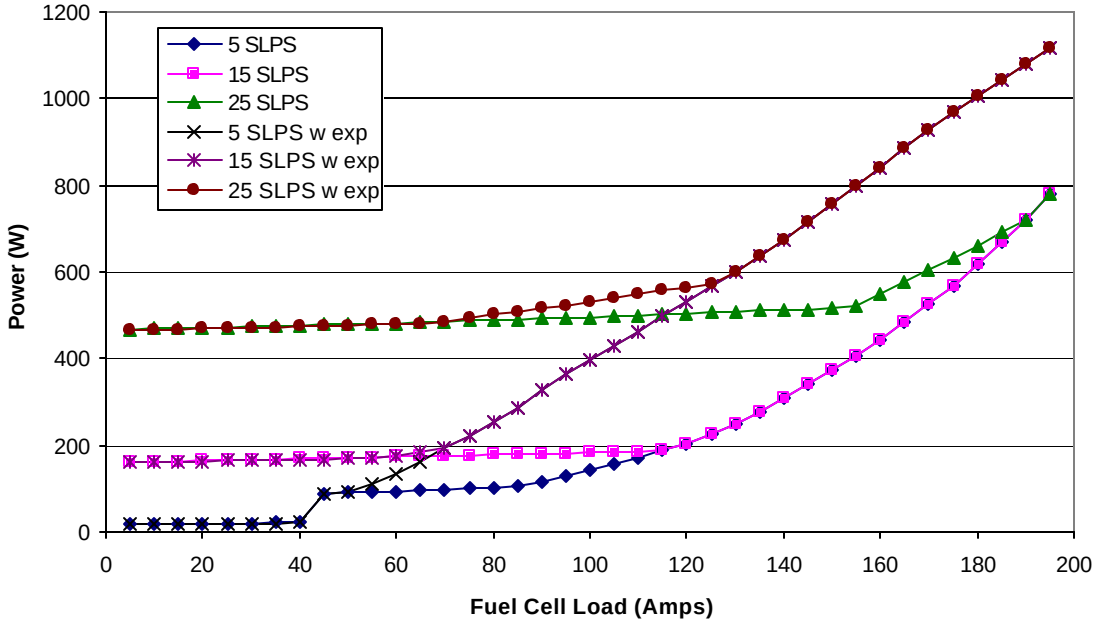

Figure 43: Turbocompressor power at several minimum airflow rates

The achievable stack temperature for the system is greatly affected by the minimum mass flow restrictions. Figures 44 to 47 show the changes in achievable stack temperature as a result of the minimum mass flows. The differences between the two twin-screw figures are primarily due to the changes in pressure between the systems with and without an expander. The same is true for the turbocompressor system. Higher pressure reduces the

amount of water evaporation needed for humidification, therefore increasing the achievable stack temperature.

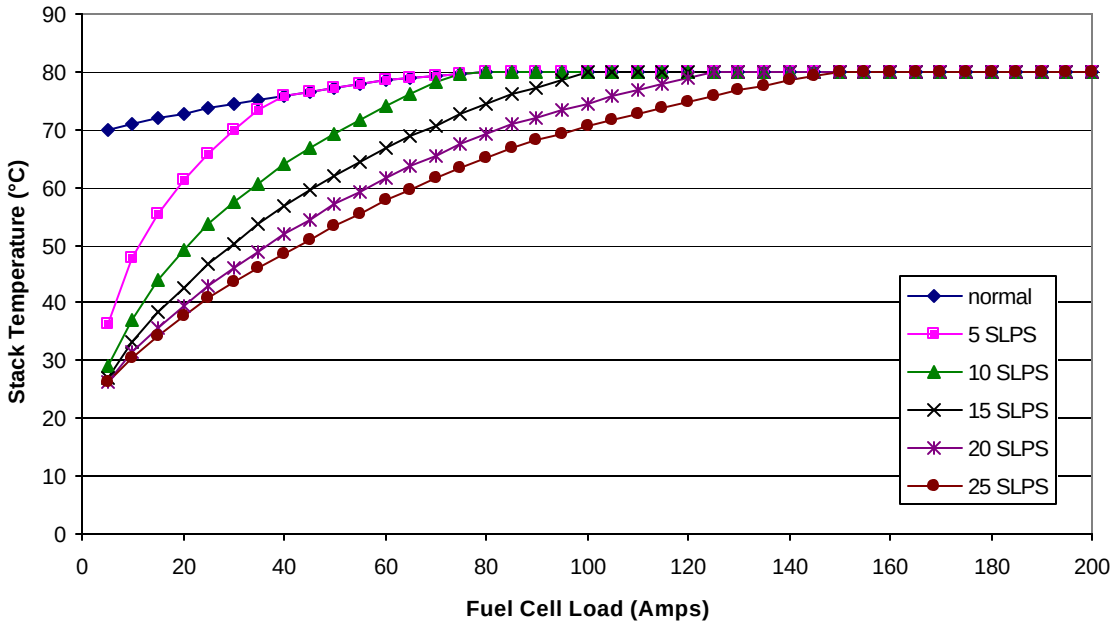

Figure 44: Achievable stack temperature with the twin-screw compressor at several minimum airflow rates

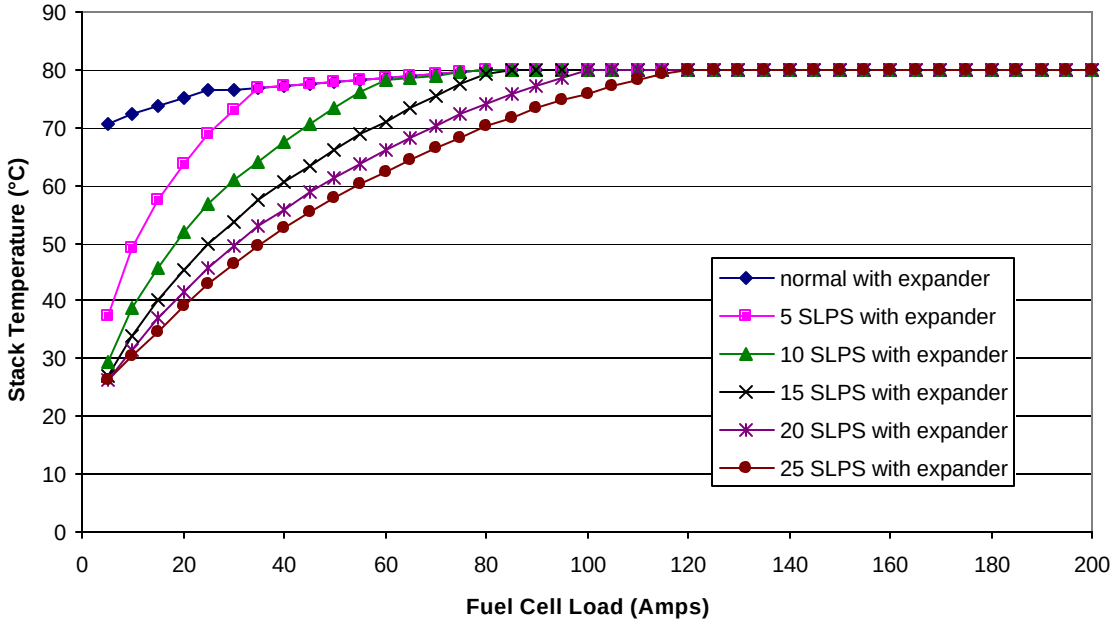

Figure 45: Achievable stack temperature with the twin-screw compressor and expander at several minimum airflow rates

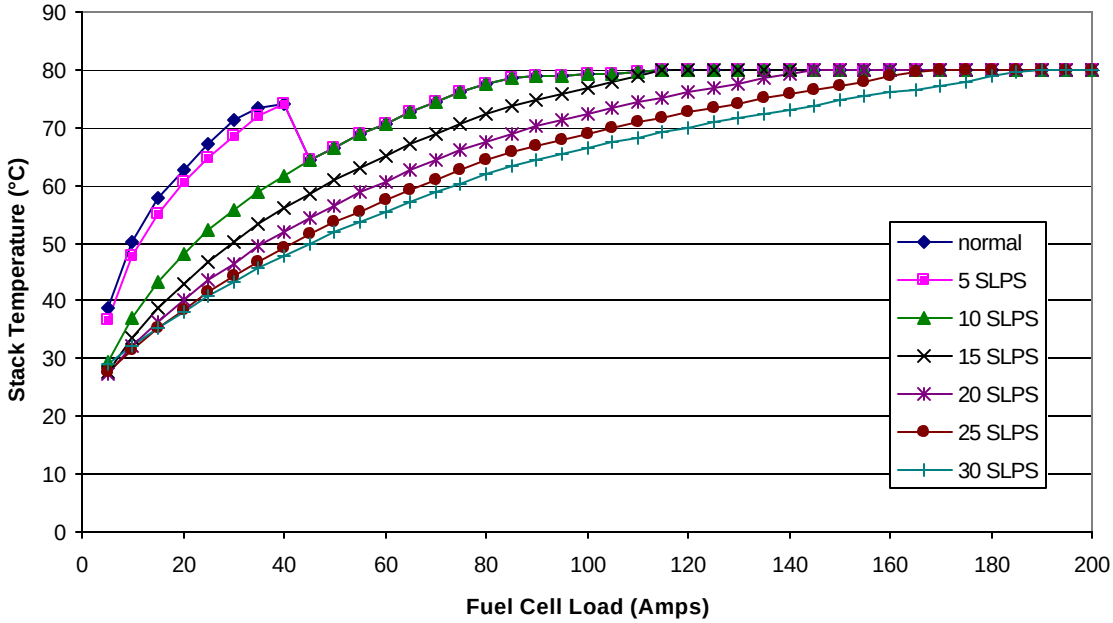

Figure 46: Achievable stack temperature with the turbocompressor at several minimum airflow rates

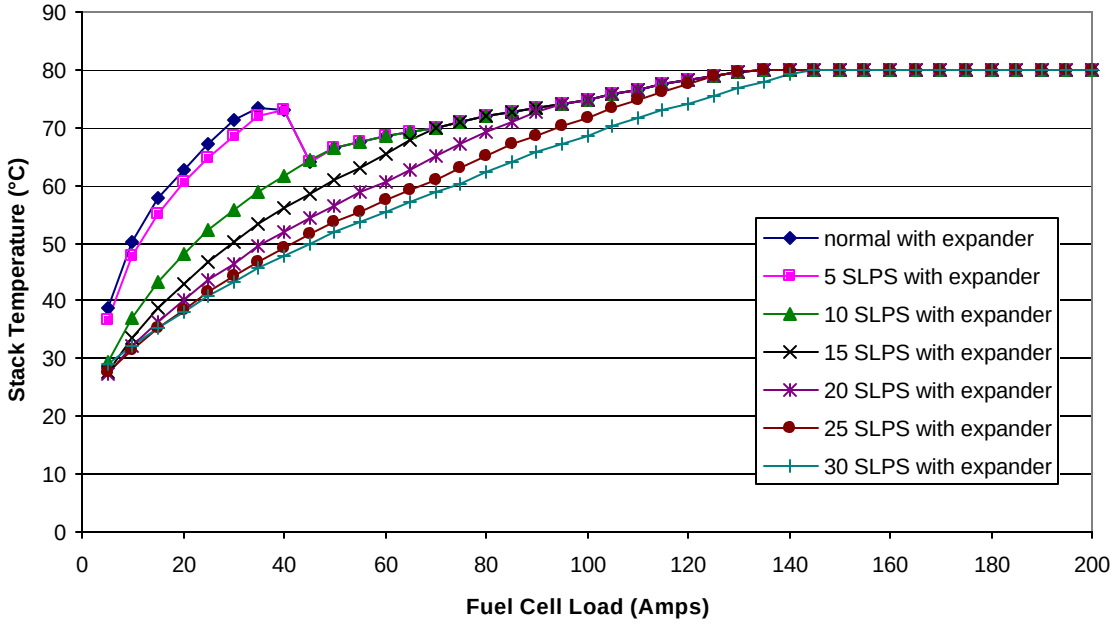

Figure 47: Achievable stack temperature with the turbocompressor and expander at several minimum airflow rates

Ambient conditions

If fuel cells will be used in automotive applications, the environment in which they will operate will constantly be changing. It's important to know what to expect from the

system if it were to operate in extreme cold or hot conditions, or if the relative humidity of the air entering the system were extremely high or low.

The systems were tested at normal inlet conditions and four different extreme inlet conditions:

Normal:	25°C,	50% RH;
Low temp, low RH:	0°C,	0% RH;
Low temp, high RH:	0°C,	100% RH;
High temp, low RH:	50°C,	0% RH;
High temp, high RH:	50°C,	100% RH.

The oxygen partial pressure of the air entering the system is the primary factor on efficiency. At low temperatures, the air is denser, and therefore the compressor does not need to work as hard to supply enough oxygen to the fuel cell. Similarly, at low humidity, there is more oxygen per mass of flow entering the compressor. So for both of these conditions the efficiency is higher. The lowest efficiency group is high temperature and high humidity, which gives the lowest oxygen per mass flow of any of the conditions. Figures 48 through 51 show the efficiency plots of the different systems. They are similar, but there are some differences. The higher pressures of the systems with expanders cause the efficiency lines to spread apart. As mentioned above, these differences are a result of the oxygen density of the air.

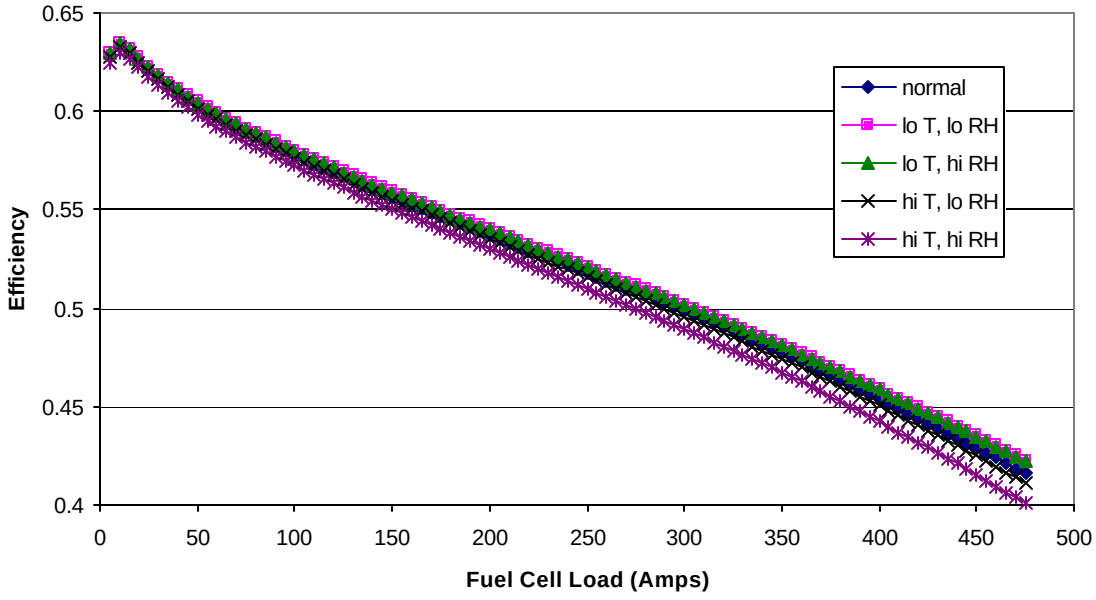

Figure 48: System efficiency with twin-screw compressor without an expander at different inlet air conditions

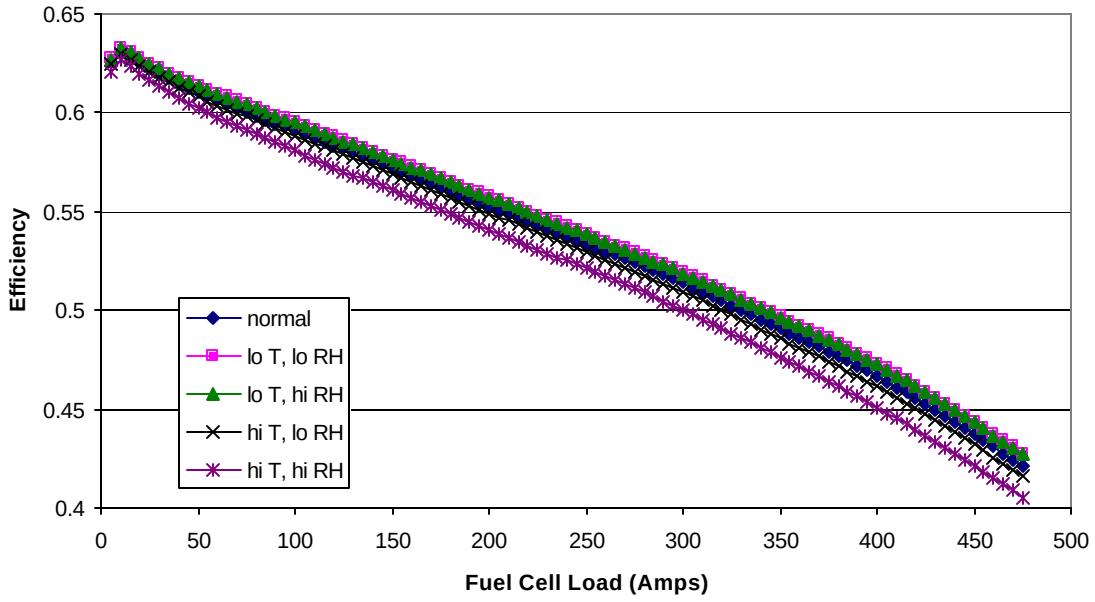

Figure 49: System efficiency with twin-screw compressor with an expander at different inlet air conditions

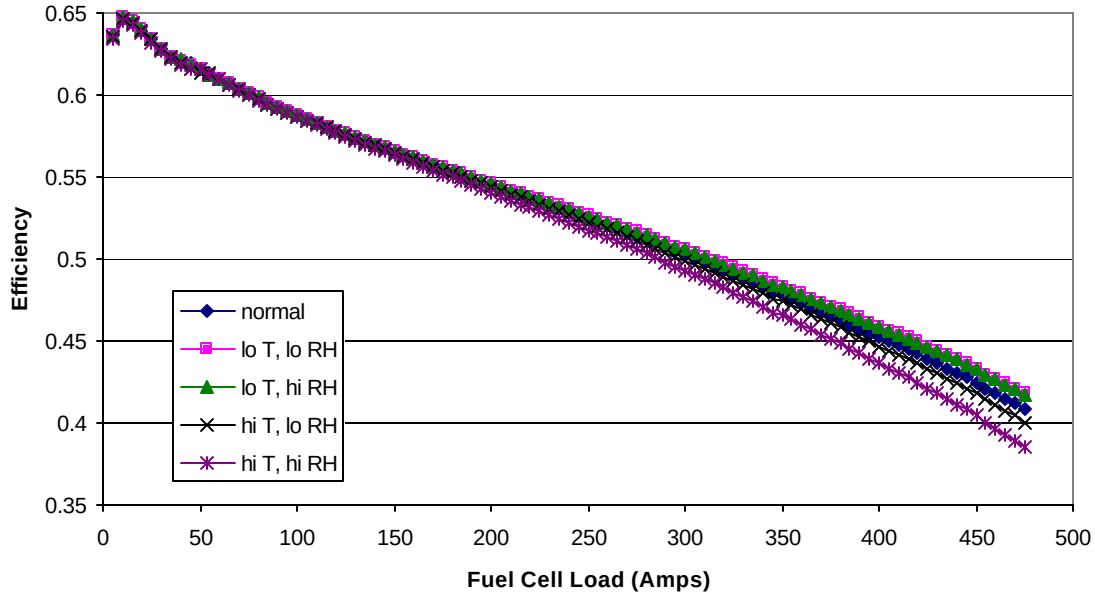

Figure 50: System Efficiency Using Turbocompressor without an expander at Different Inlet Air Conditions

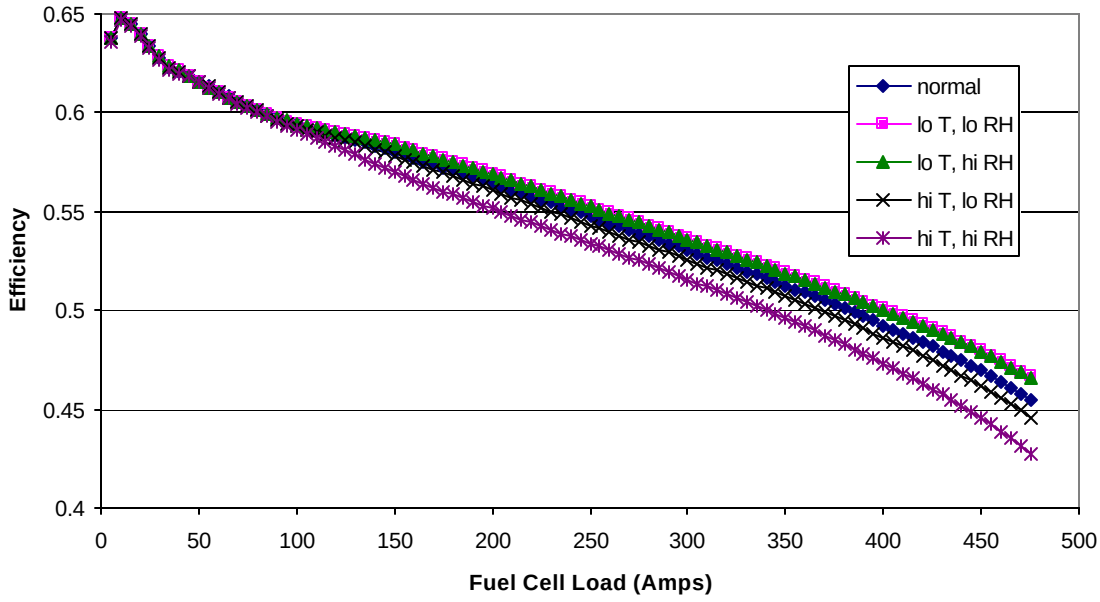

Figure 51: System efficiency using the turbocompressor with the expander at different inlet conditions

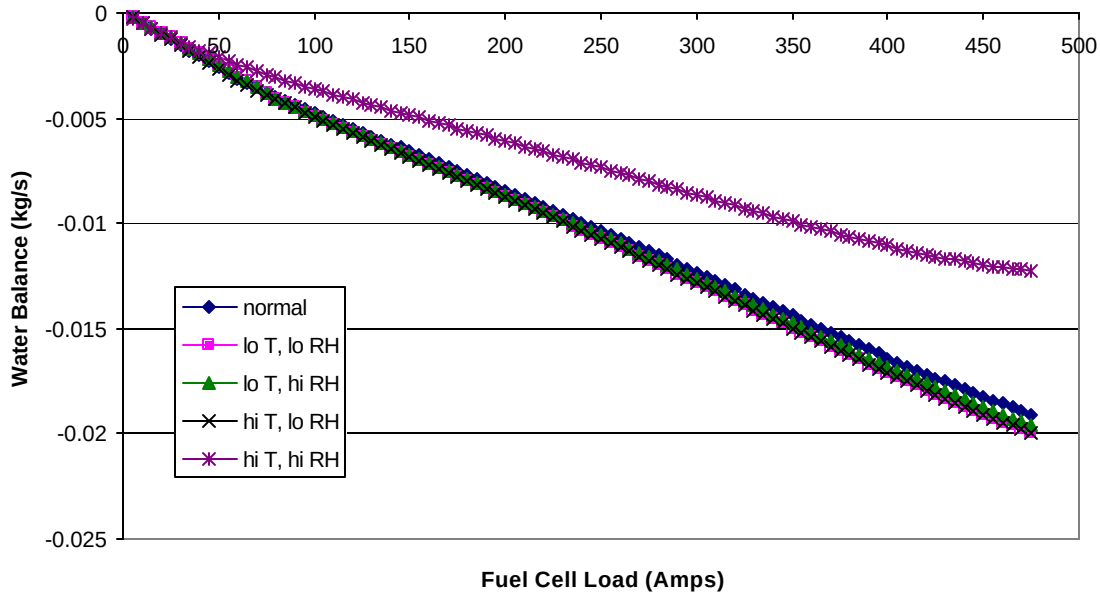

Figure 52: Water Balance for Twin-screw Compressor (no expander) at Different Inlet Air Conditions

The water balance follows what is expected for these conditions. At low temperatures, the humidity of the air does not have much of an effect on the system, since colder air holds less water overall. The only condition that has a significant and positive effect on the overall water balance is the high temperature and high humidity condition. Hot air can hold the most water, and therefore has the greatest effect on the system.

Unfortunately, this is the worst scenario for the efficiency. Figures 52 through 55 show the different systems and the water balance characteristics of each. For all four systems, the high temperature, high humidity condition has a dramatic effect. In Figure 54, the turbocompressor system without an expander shows a difference of almost 9 grams per second in water balance between the low temperature, low humidity condition and the high temperature, high humidity condition.

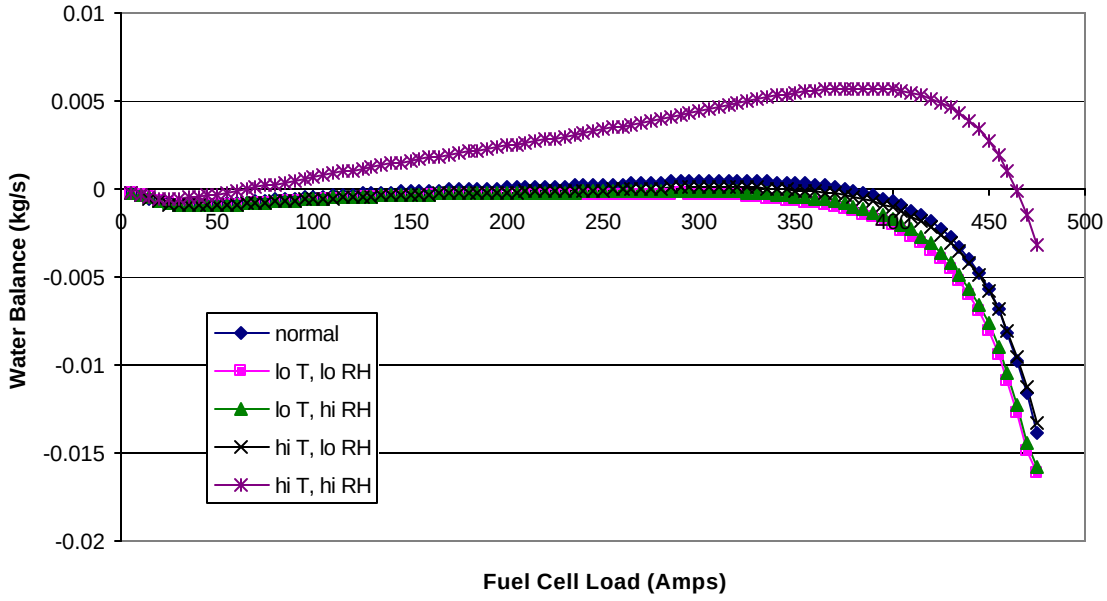

Figure 53: Water balance with twin-screw compressor with an expander at different inlet air conditions

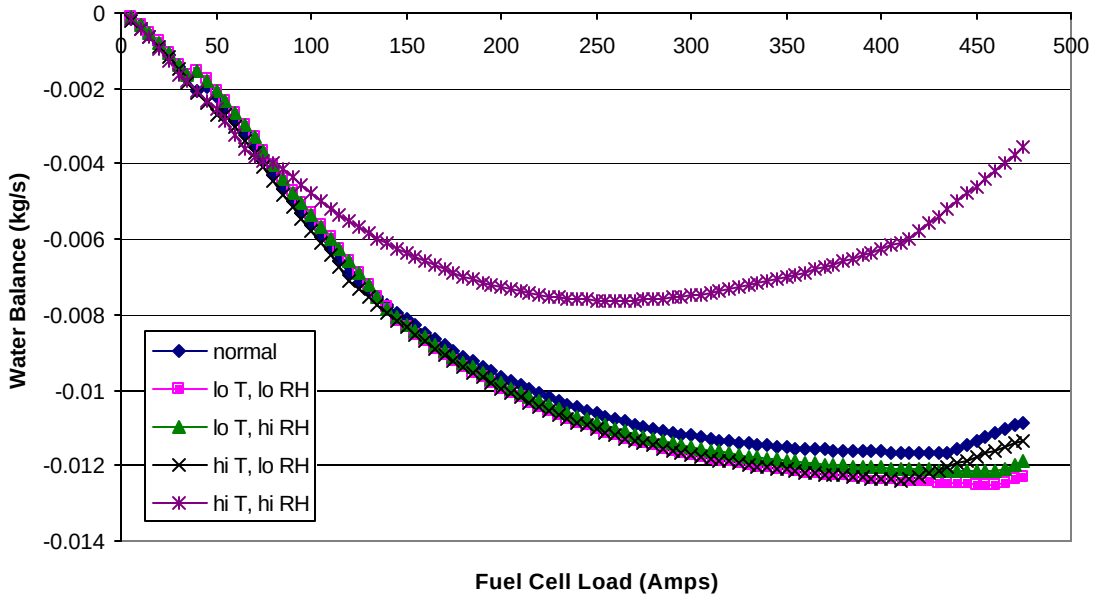

Figure 54: Water balance using turbocompressor without an expander at different inlet air conditions

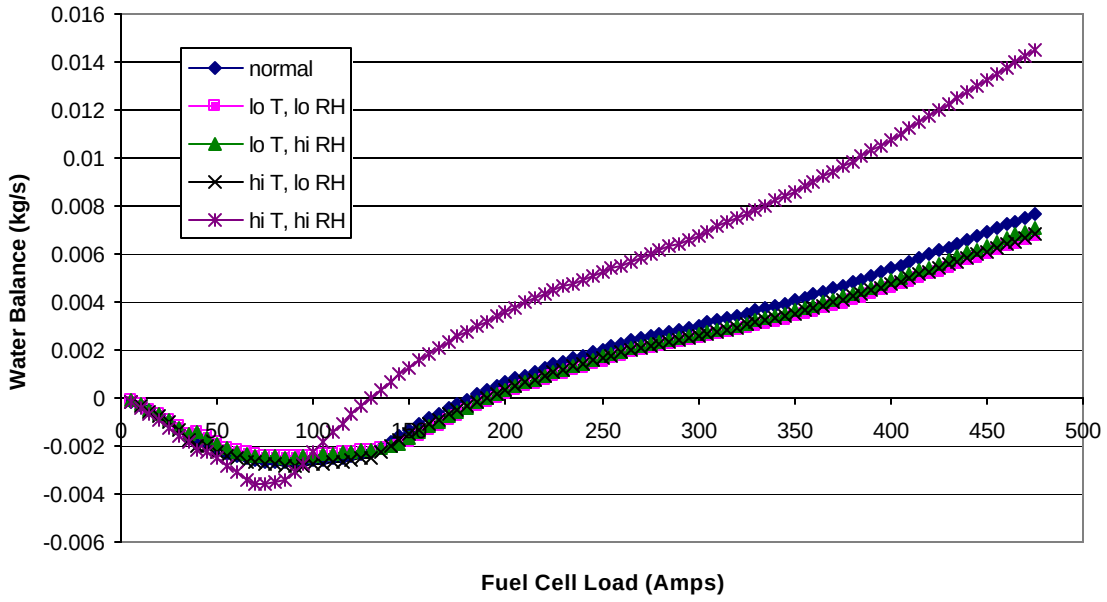

Figure 55: Water balance using the turbocompressor with the expander at different inlet conditions

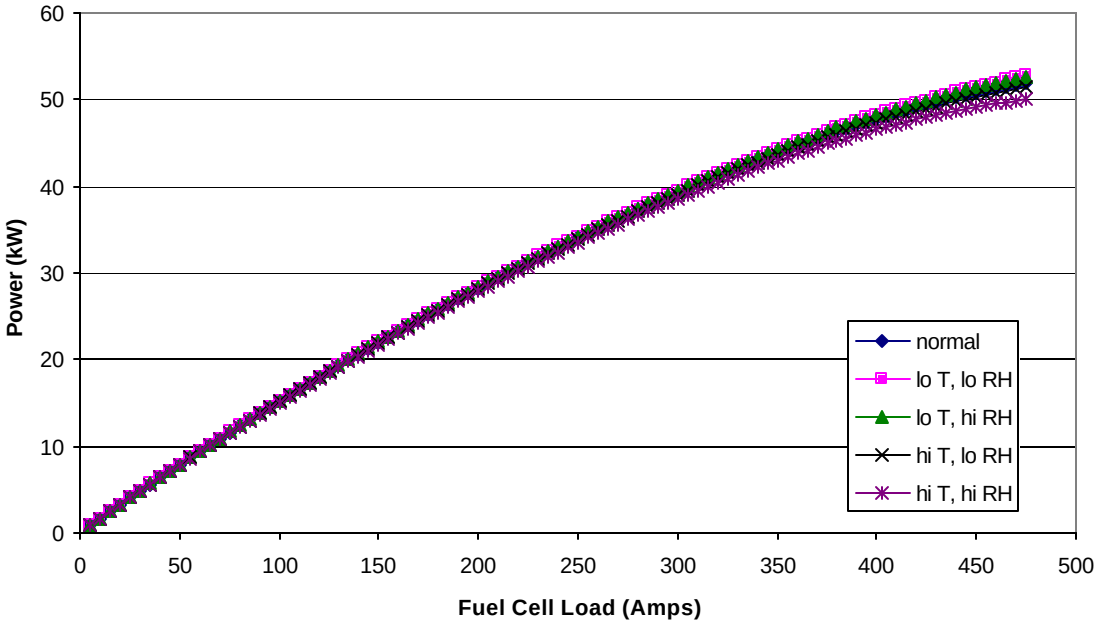

Figure 56: Net system power with twin-screw compressor at different inlet air conditions

The net system power is directly affected by the ambient conditions. Because of the lower efficiency of the compressor at high temperature, the net power output suffers. Figures 56 through 59 show the net power for the four fuel cell system configurations. The turbocompressor suffers worse than the twin-screw compressor at high fuel cell load due to its higher pressures. There is a difference of approximately 4 kW of power

between the high and low temperature and pressure conditions for the turbocompressor systems at high fuel cell load, whereas the twin-screw compressor systems have about a 3 kW difference.

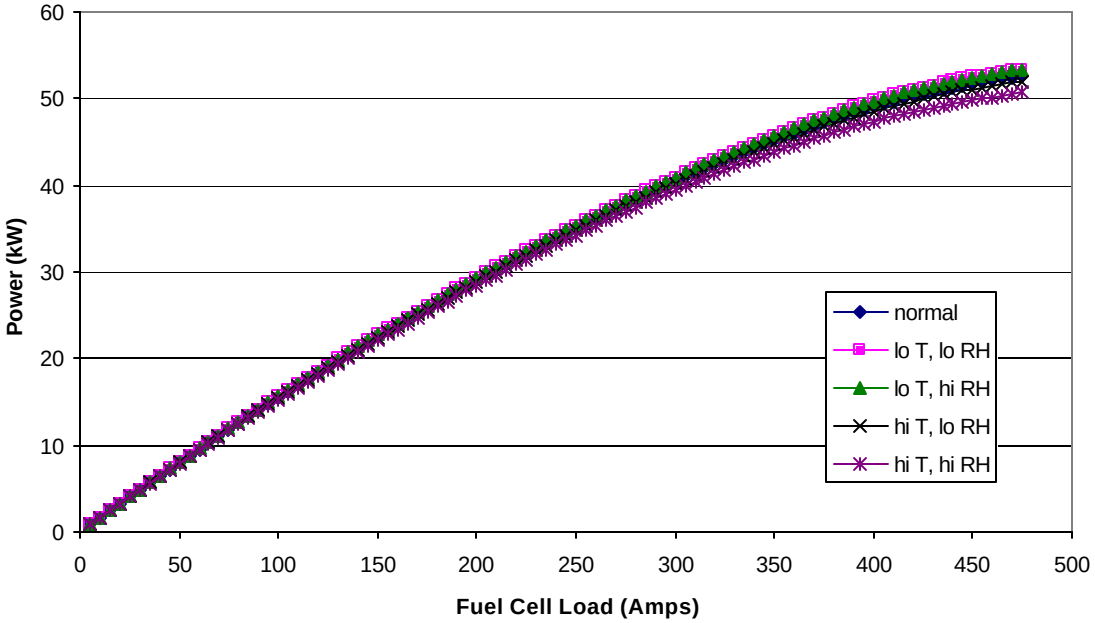

Figure 57: Net system power with twin-screw compressor and expander at different inlet air conditions

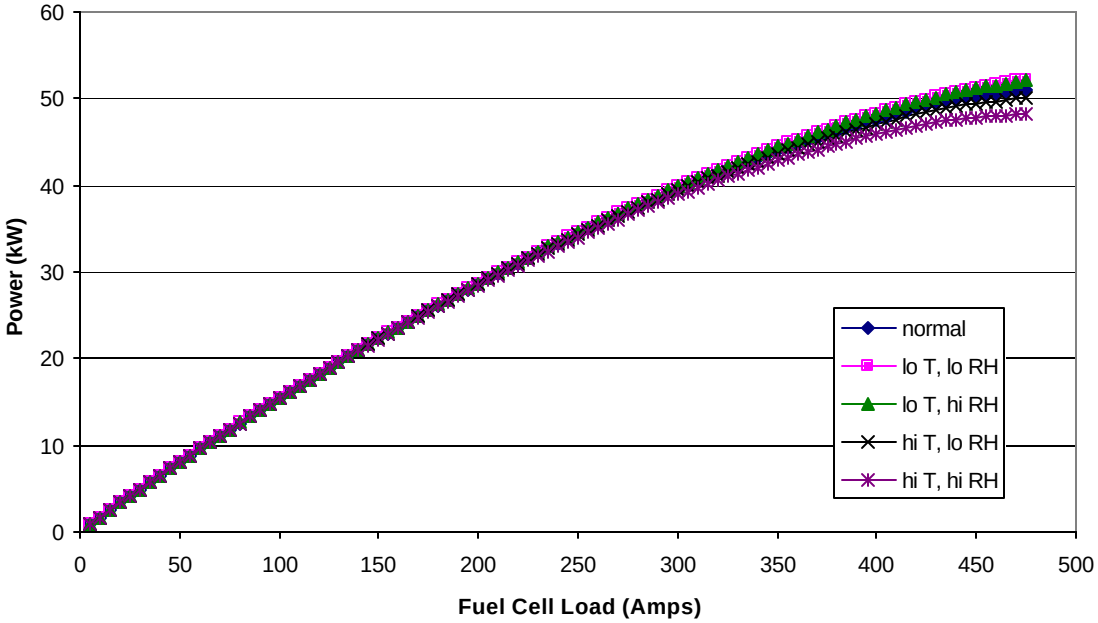

Figure 58: Net system power with turbocompressor at different inlet air conditions

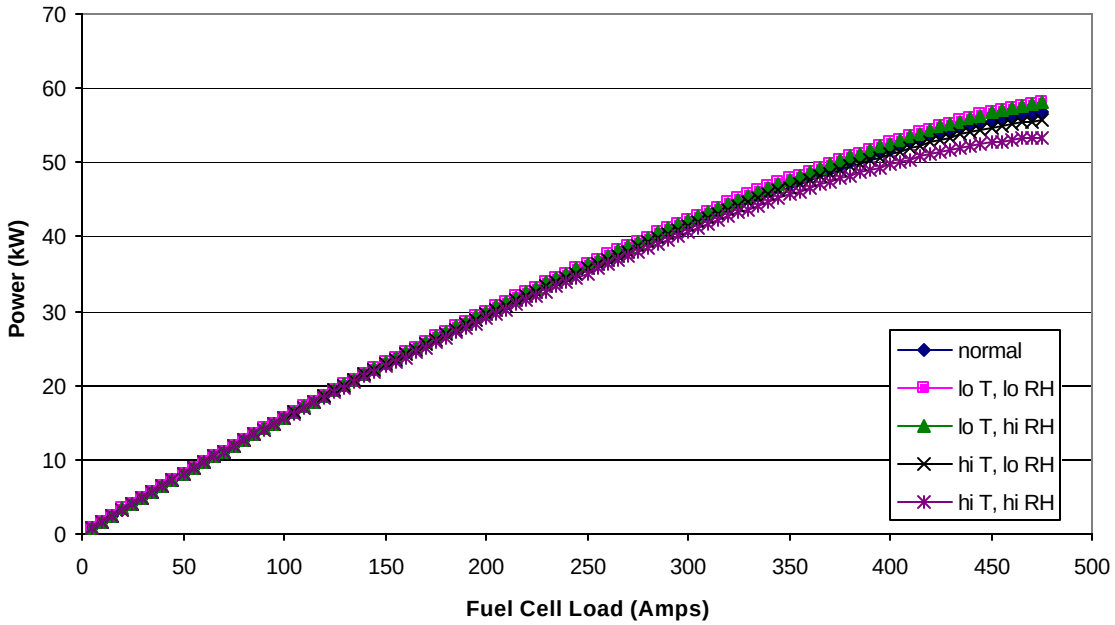

Figure 59: Net system power with the turbocompressor and expander at different inlet air conditions

Stack temperature

The temperature of the stack is an important value for consideration. A higher temperature positively affects the voltage of the stack and increases the ability of the system to reject heat into the surroundings. A lower temperature helps the water balance of the system and makes it easier for the system to humidify the incoming air to the desired relative humidity.

Three possible temperature choices for the fuel cell—60, 70, and 80°C—are investigated, as well as the very low value of 30°C. This lower value is just to see what would happen in startup conditions. For instance, a fuel cell automobile might be turned on and immediately forced to a high power level before it was able to warm to operating temperature. At this lower condition, the power needed for the radiator fan and the coolant pump is neglected. The reason for neglecting these components is because, at startup, higher temperature operation is desired. Therefore, cooling the stack is not necessary at startup.

As expected, the higher temperatures give higher efficiency for both the twin-screw system and the turbocompressor system, shown in Figures 60 through 63. One anomaly shown in the figures is the higher efficiency at low power on the 30°C line. This is caused by the fact that the pump and fan power are neglected at this temperature.

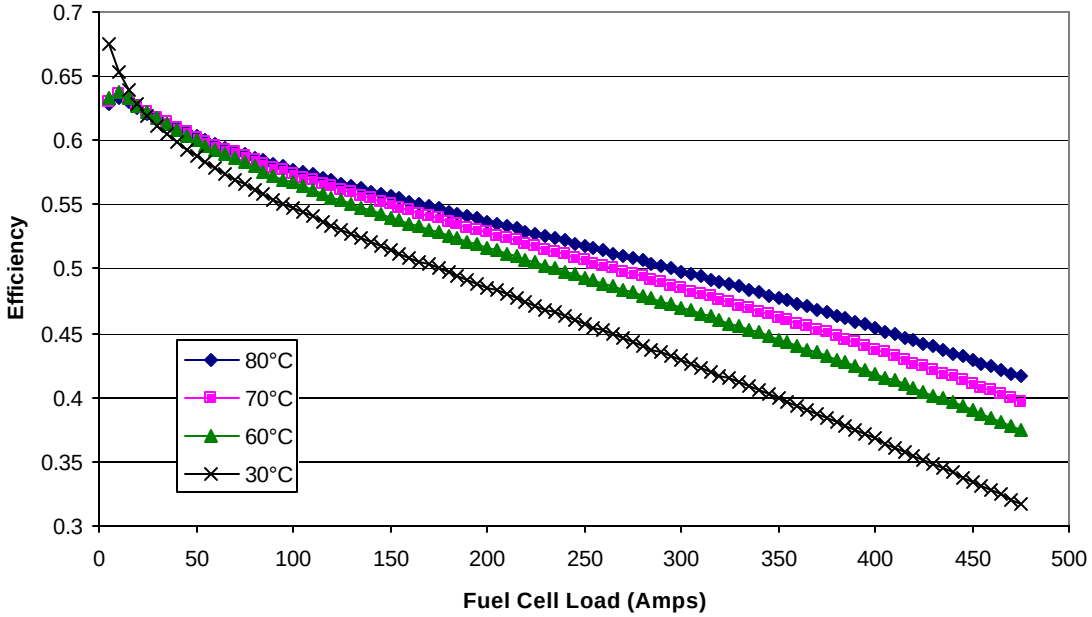

Figure 60: System efficiency at different fuel cell temperatures using the twin-screw compressor without an expander

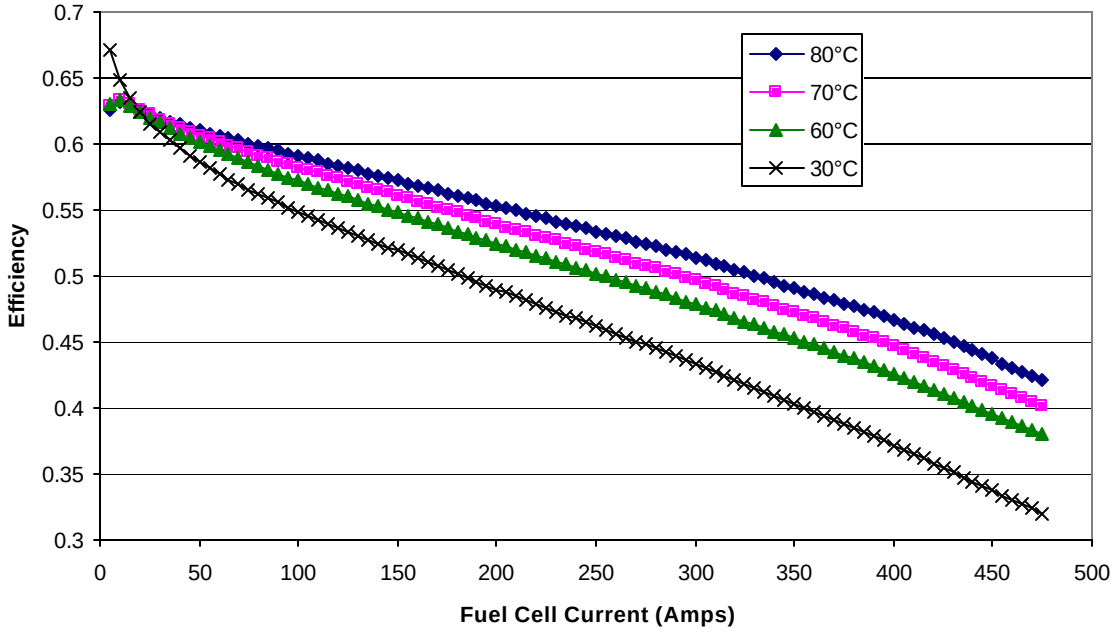

Figure 61: System efficiency at different fuel cell temperatures using the twin-screw compressor with an expander

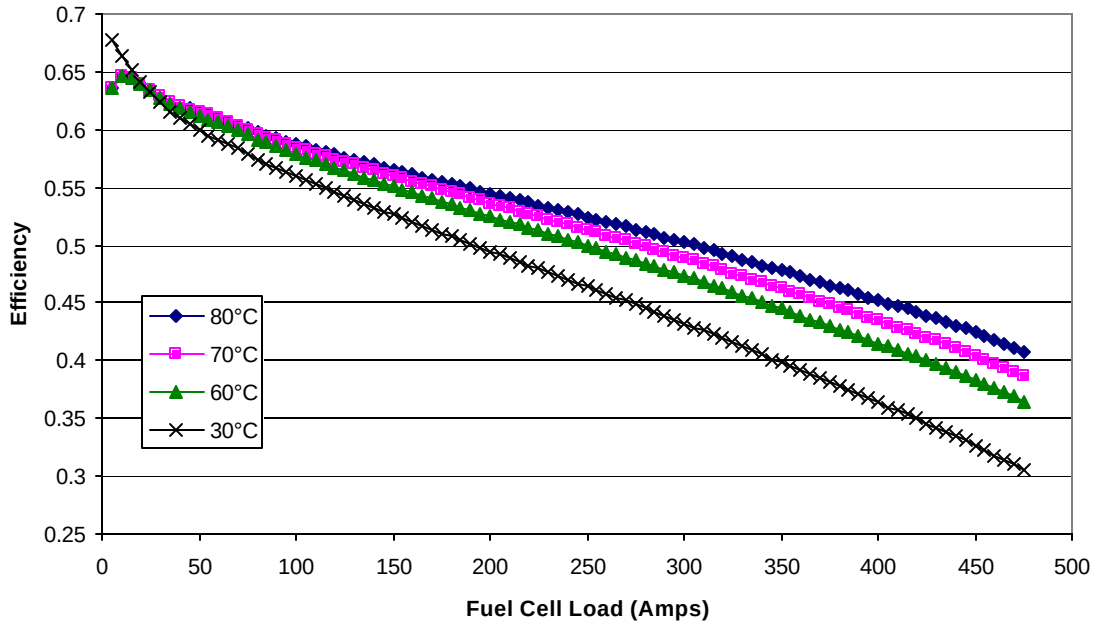

Figure 62: System efficiency at different fuel cell temperatures using the turbocompressor without an expander.

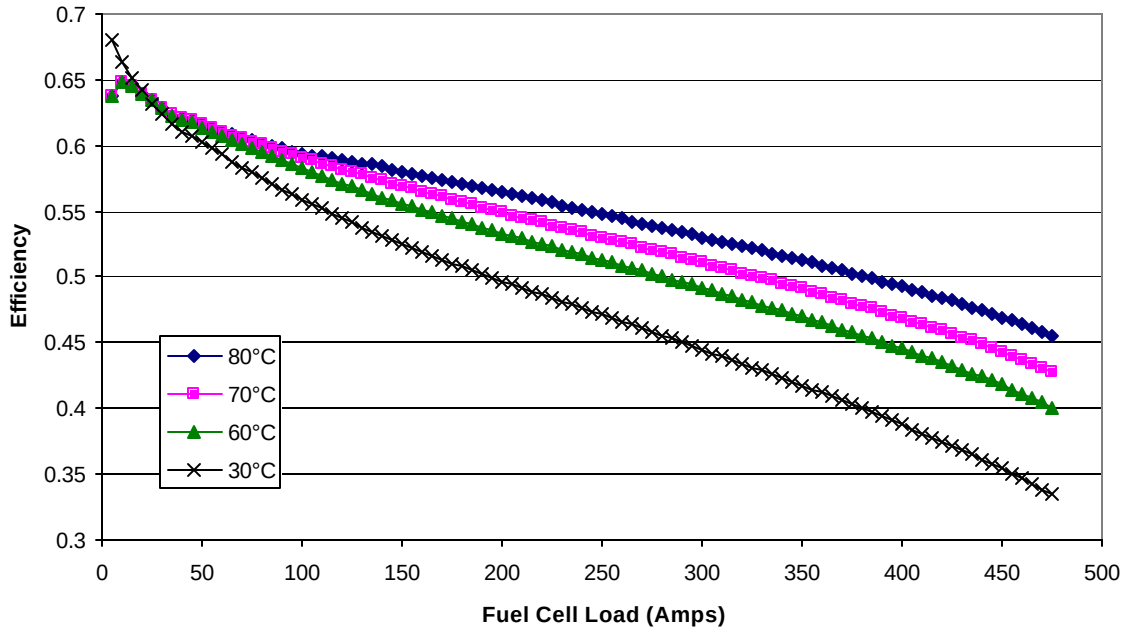

Figure 63: System efficiency at different temperatures using the turbocompressor with an expander

Again, as expected, the water balance is inversely proportional to the temperature of the stack. Higher stack temperature results in lower water balance, and vice versa. Note that the anomalies in the turbocompressor maps at temperatures of 60 and 70°C are a result of

the strange low flow characteristics imposed on the turbocompressor. At those low flows, no data was given in the compressor map, so the speed, pressure, and flow rate of the compressor are held at a minimum. Once the flow rate overcomes the minimum, there is a sudden jump in the pressure and flow rate values given by the compressor, which shows up in the figures as a discontinuity.

Figures 64 through 67 show the water balance for the four fuel cell systems at different stack temperatures. The differences in water balance are largest in those systems without an expander. The twin-screw compressor with an expander also shows a significant difference in water balance between the temperatures because of the operating pressure line downward curve at high fuel cell load. However, the turbocompressor/expander operating pressure line continues to rise at high load, and the water balance reflects this rise with very little difference between the temperature lines.

Figures 68 through 71 show the net system power for the four fuel cell systems at different stack temperatures. There is a direct relationship between temperature and fuel cell voltage, so higher temperatures benefit the output power of the fuel cell system. There is very little difference between all of the net system power figures, since the compressor efficiency is not greatly affected by the stack temperature.

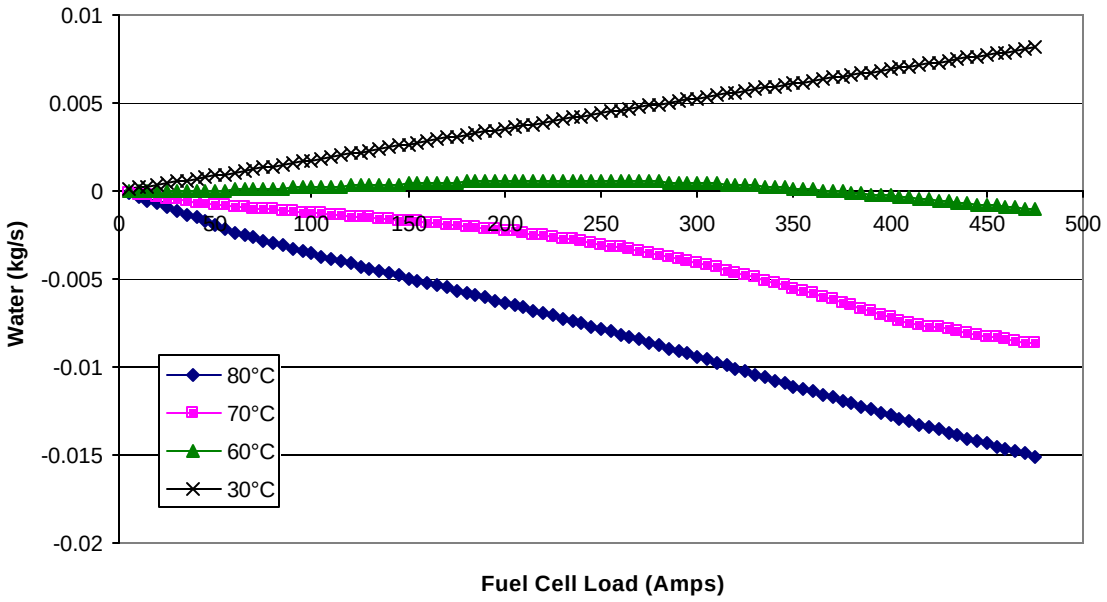

Figure 64: Water balance at different fuel cell temperatures using the twin-screw compressor without an expander

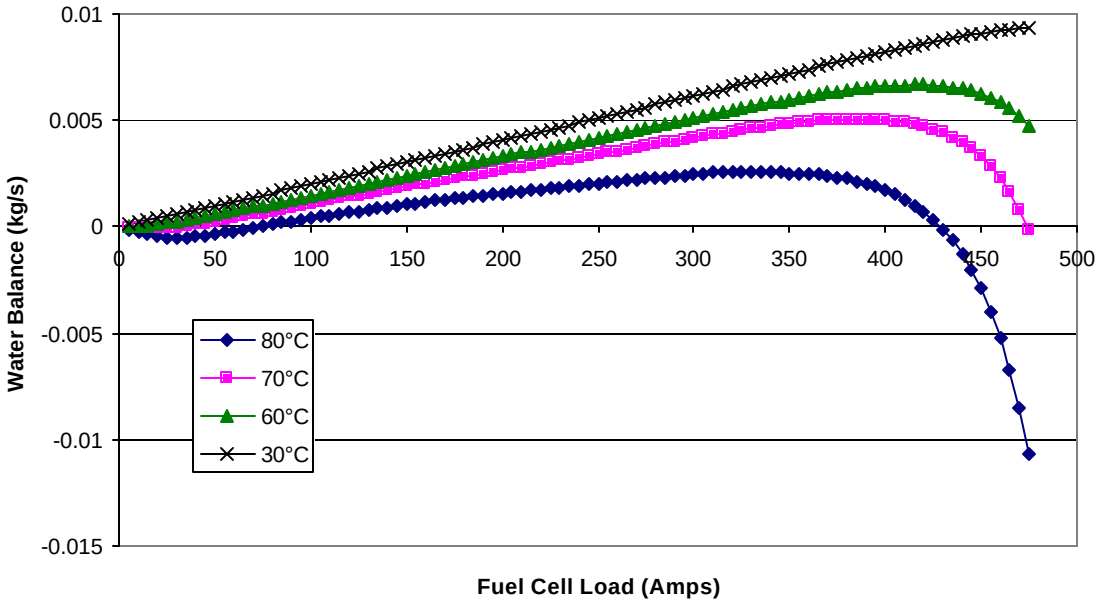

Figure 65: Water balance at different fuel cell temperatures using the twin-screw compressor with an expander

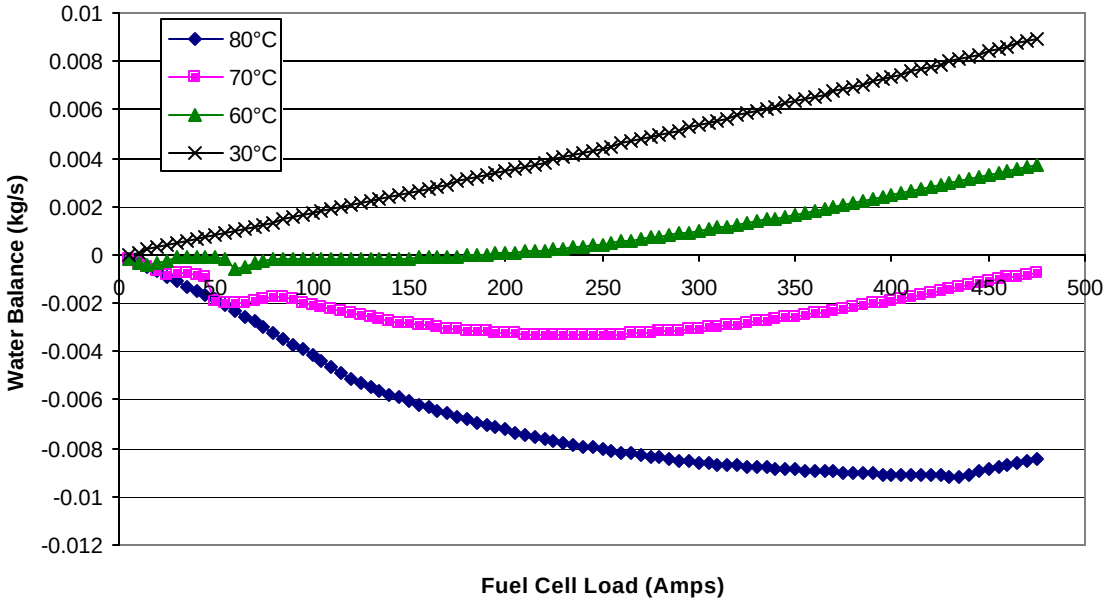

Figure 66: Water balance at different temperatures using the turbocompressor without an expander

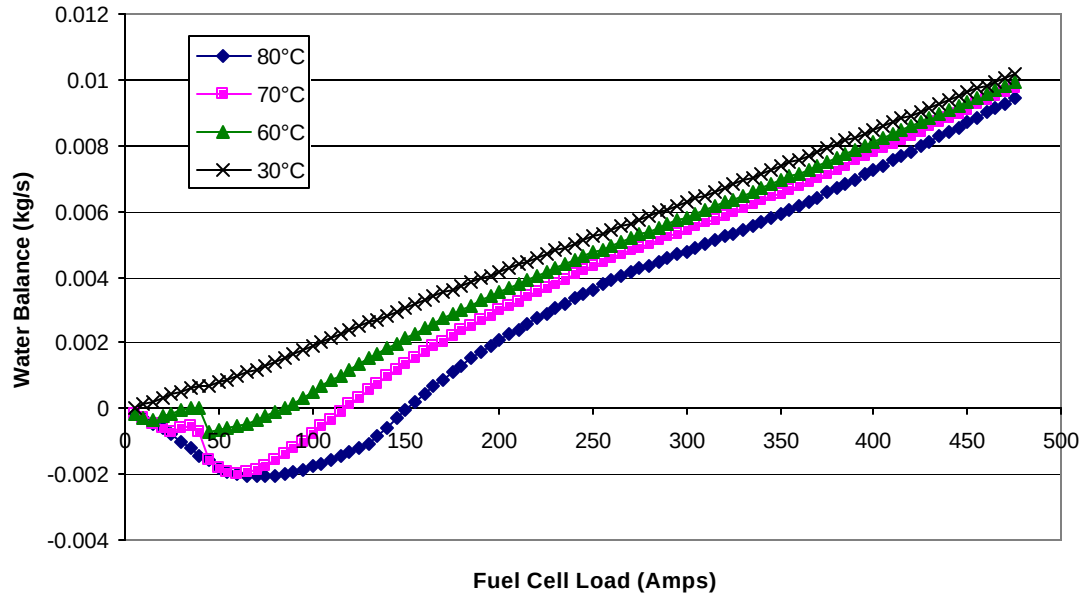

Figure 67: Water balance at different fuel cell temperatures using the turbocompressor with an expander

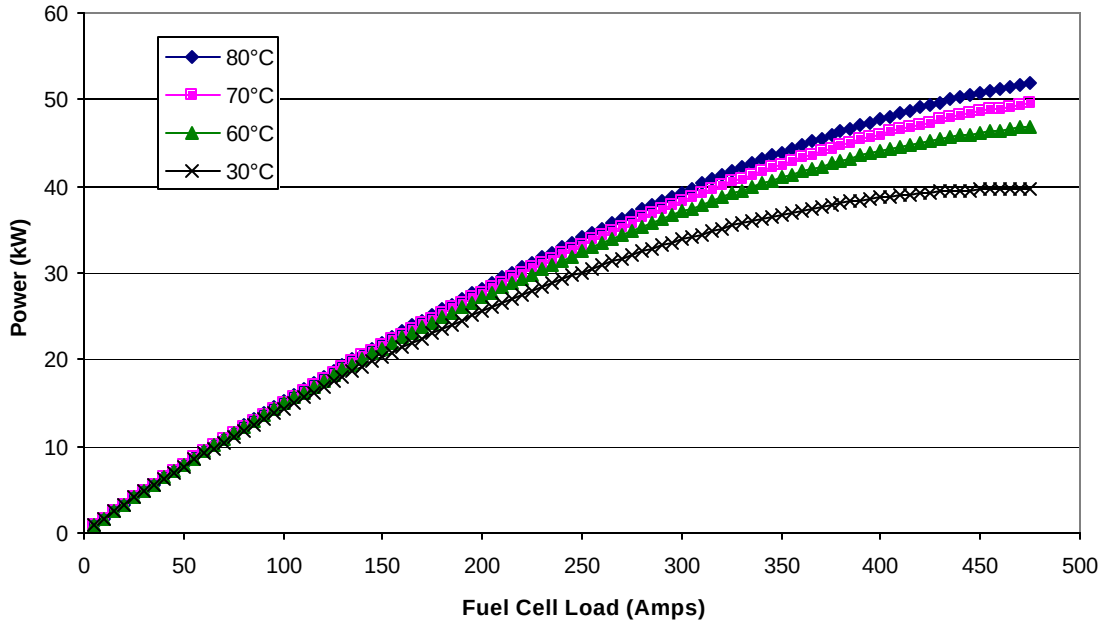

Figure 68: Net system power using the twin-screw compressor without an expander at different fuel cell temperatures

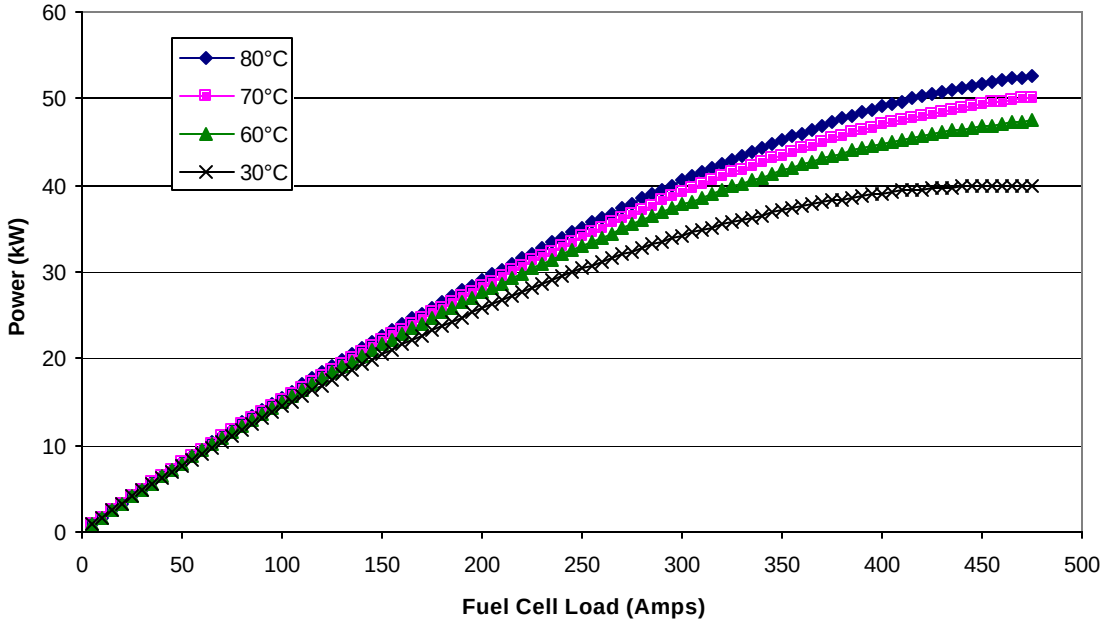

Figure 69: Net system power using the twin-screw compressor with an expander at different fuel cell temperatures

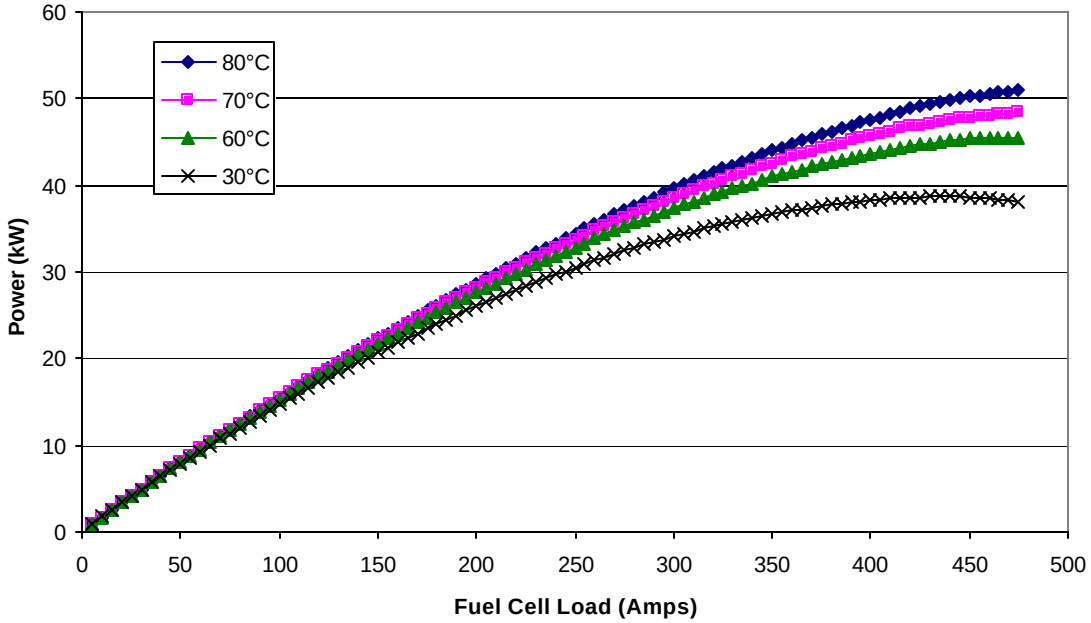

Figure 70: Net system power using the turbocompressor without an expander at different fuel cell temperatures

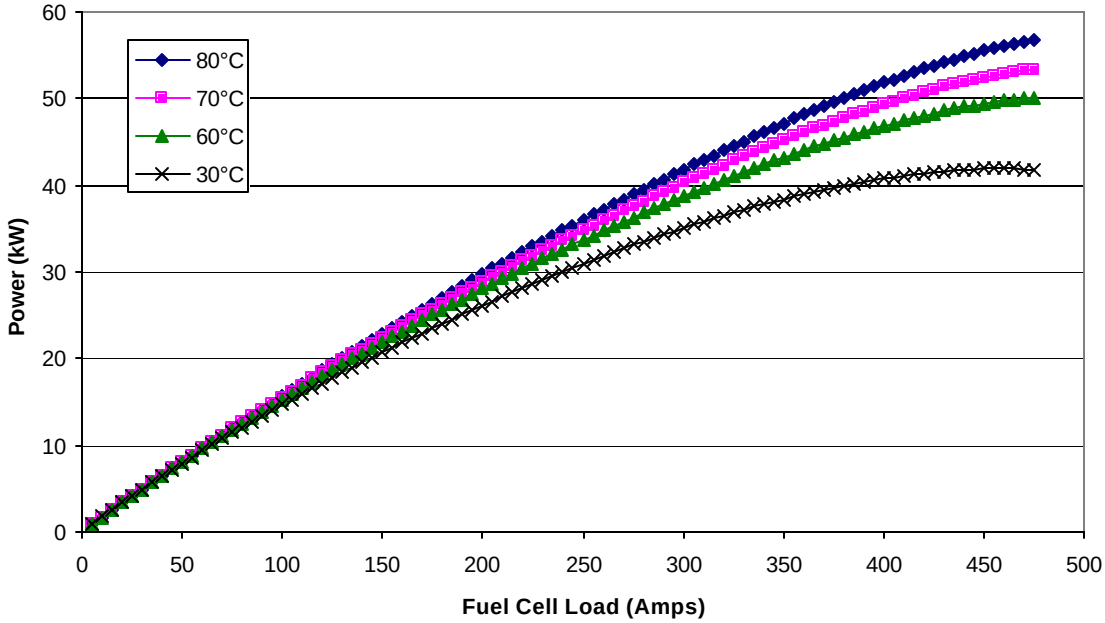

Figure 71: Net system power using the turbocompressor with an expander at different fuel cell temperatures

Higher twin-screw compressor pressure

If a higher operating line were imposed on the system, the water balance would improve. This is possible for the twin-screw compressor, but is not easily implemented for the turbocompressor because of its surge line. To set higher pressures with a turbocompressor would require higher flow rates as well, and would essentially show the results of a minimum airflow rate.

The higher-pressure line for the twin-screw compressor, compared to the original operating lines in Figure 72, is used to show the effects of a higher-pressure line on the system, as shown in Figures 73 through 76. As expected, the water balance at higher pressures is greatly improved from the water balance with the previous operating pressure line. The water balance for the system with an expander is almost completely in the positive region, except for the very lowest fuel cell loads. The drawback is in the efficiency, as shown in Figure 73. The efficiency drops by approximately 3 to 5 percent throughout the range of the fuel cell load without the expander, and about 2 to 3 percent with the expander.

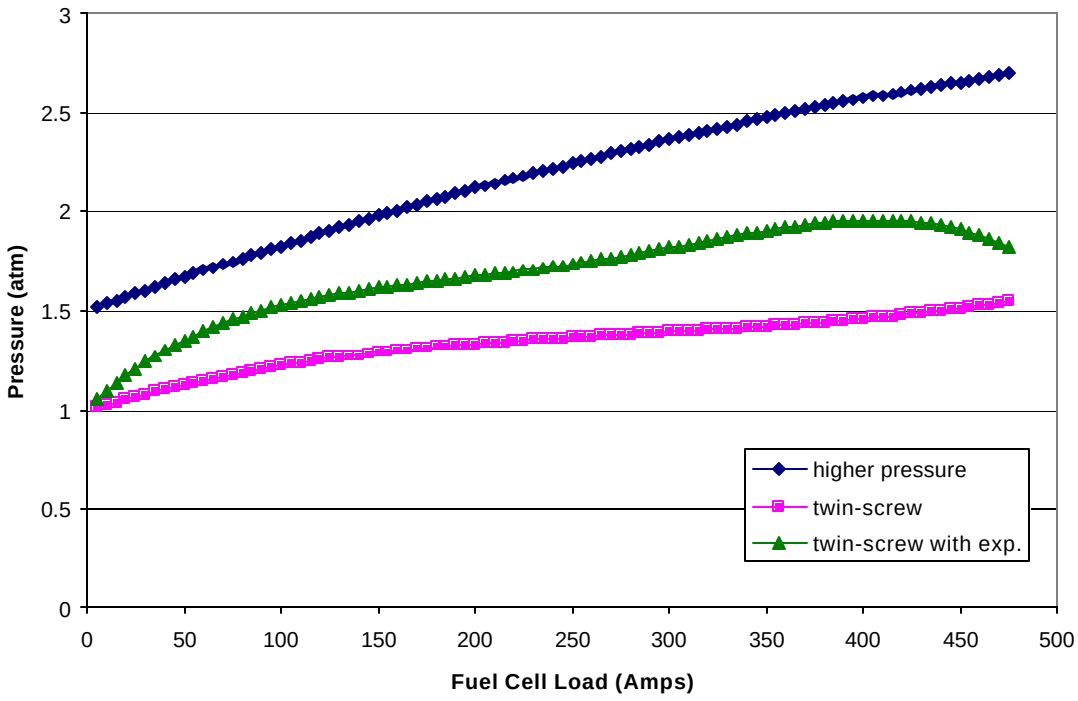

Figure 72: Twin-screw higher pressure operating line compared to highest efficiency curve fits

The benefit of an expander is greatly enhanced when the pressure of the system is increased, as shown in Figures 75 and 76. There is very little difference between the normal pressured systems with and without the expander, but the difference at high fuel cell load for the higher pressure system is approximately 5 kW. This difference is quickly seen in the net system power as well. Though the higher pressure system with an

expander takes about 4 or 5 kW more power to supply air, the benefit to the stack voltage recovers some of this loss, so that the net power is near to the normal pressure systems. Therefore, a better water balance can be achieved with very little loss to the net system power.

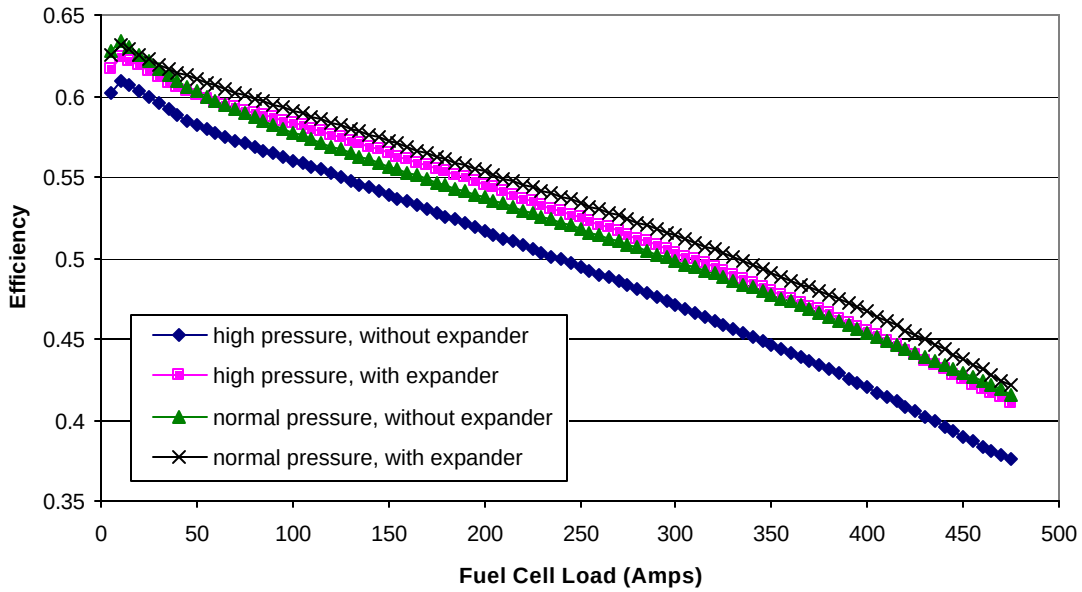

Figure 73: System efficiency at higher pressures than normal for twin-screw compressor

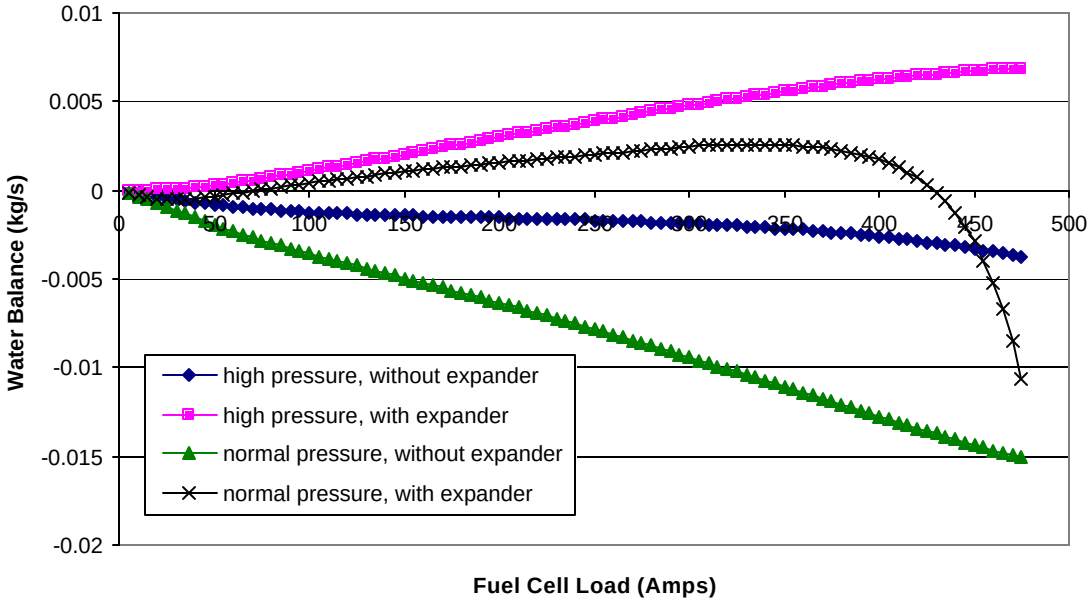

Figure 74: Water balance at higher pressures than normal for twin-screw compressor

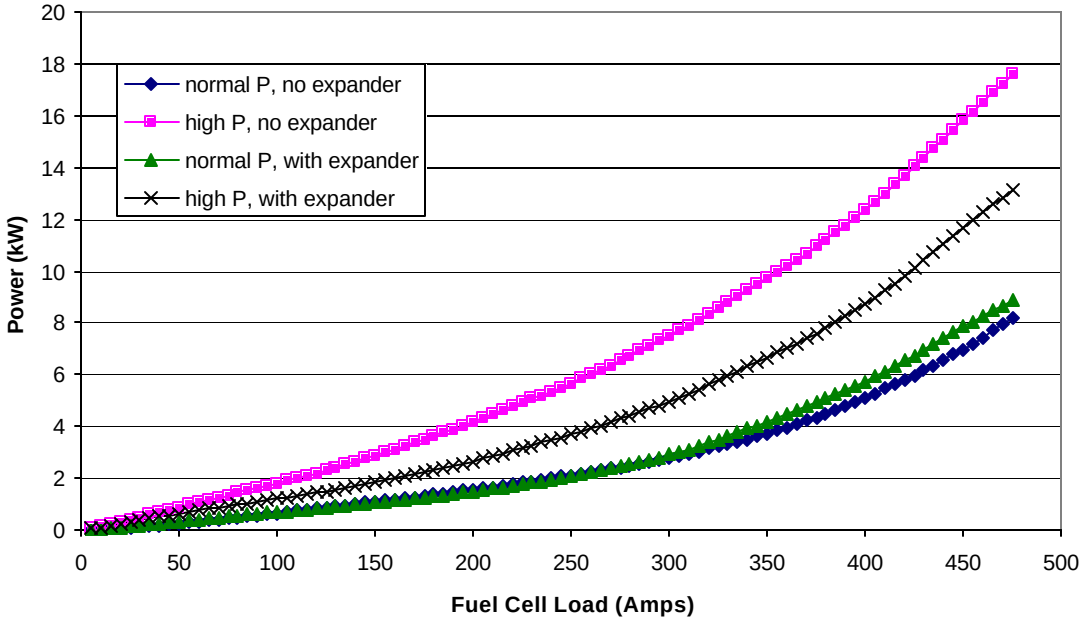

Figure 75: Comparison of compressor power at normal and higher pressures than normal for twin-screw compressor

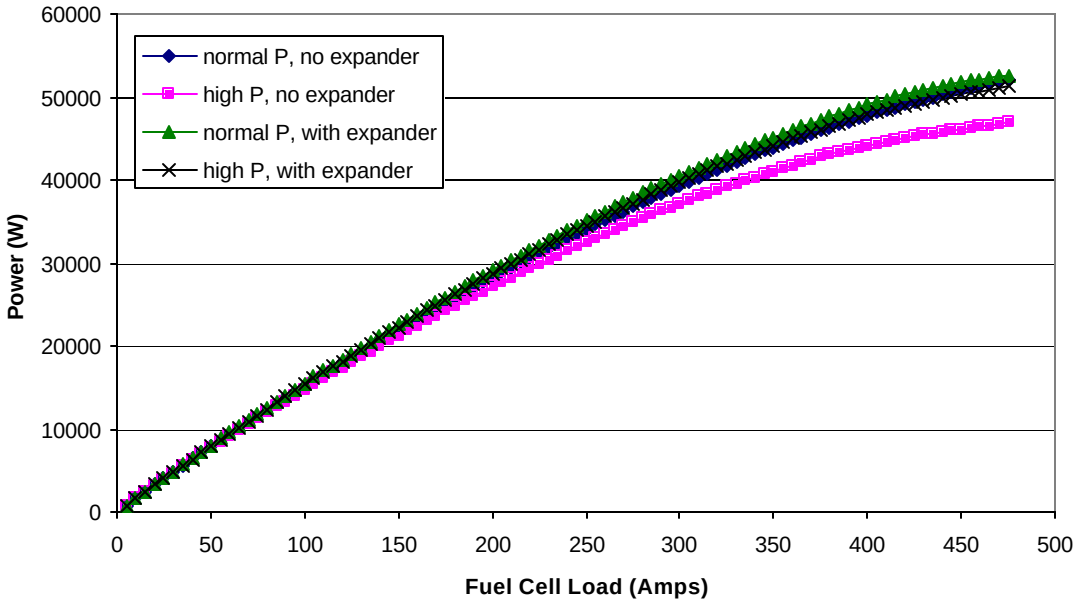

Figure 76: Comparison of net system power at normal and higher pressures than normal for twin-screw compressor

Chapter 6: Conclusions

The twin-screw supercharger and turbocharger are the two technologies that will most likely be used in a fuel cell vehicle because of their size and weight. In ideal conditions, the turbocompressor has about the same efficiency as the twin-screw across the fuel cell load with no expander added. When the expander is added to the system, however, the turbocompressor/expander has higher efficiency than the twin-screw compressor/expander over most of the fuel cell load, except for low fuel cell load where the efficiencies are approximately the same. The approximate maximum efficiency for each system is 65 percent for the turbocompressor systems (with and without expander) and 63 percent for the twin-screw systems (with and without expander). At peak fuel cell load, the efficiencies are as follows:

Turbocompressor:	40.8 % at 51 kW
Turbocompressor with expander:	45.5 % at 56.8 kW
Twin-screw compressor:	41.6 % at 52 kW
Twin-screw compressor with expander:	42.1 % at 52.6 kW

Water balance is a difficulty for both compressor systems when the expander is excluded. With the expander, the twin-screw compressor maintains a positive water balance for the majority of the fuel cell load except for at the lowest and highest fuel cell loads. These exceptions are addressed later by increasing the output pressure of the twin-screw. Water balance issues are a problem for the turbocompressor at low fuel cell loads, and surge line constraints prevent adjustment of the operating line.

Maintaining the temperature of the stack is difficult at low fuel cell load if the only source of energy for heating the stack is the fuel cell waste heat. Mass flow is the primary driver of this problem, so lower mass flows help to maintain a higher stack temperature. The turbocompressor, to maintain efficiency, must produce higher mass flows than is necessary for the twin-screw, and therefore lowers the stack temperature for the turbocompressor system at very low loads. Higher stack pressure has some positive effect by reducing the amount of water evaporation needed to humidify the inlet air.

Minimum mass flow conditions are one way of addressing the potential problem of stack oxygen starvation in fuel cell vehicles. The drawback to this solution is the loss of efficiency at low power conditions. The turbocompressor can produce these minimum mass flows at higher efficiency than the twin-screw compressor. This is primarily due to the particular pressure operating line chosen for the two compressors. Flow rates of about 25 SLPS drop the fuel cell system efficiency down below zero for the twin-screw compressor systems with and without the expander at the lowest fuel cell load. The turbocompressor is capable of flow rates above 30 SLPS at the lowest fuel cell load without dropping below zero. Future work may show that lowering the pressure of the twin-screw compressor at these conditions may improve its efficiency.

Changes in ambient conditions have a small effect on the system performance except when there are hot and humid conditions. In general, hot and/or humid conditions tend to

lower the system efficiency, because the compressor has lower efficiency at high air temperatures and there is less oxygen per mass flow rate at high humidity. When the conditions are both hot and humid, however, water balance significantly increases since a large amount of water is accepted with the inlet air. Efficiency for the twin-screw compressor configurations at the highest fuel cell load decreases by 2 percentage points from the highest to the lowest temperature and humidity conditions. Water balance decreases 7.7 grams per second for the twin-screw compressor alone and decreases 13 grams per second with the expander added. The turbocompressor alone experiences a 3 point efficiency decrease, and with the expander, a 4 point decrease from the highest to the lowest temperature and humidity conditions at the highest fuel cell load. Water balance decreases 8.7 grams per second with the turbocompressor alone and decreases 7.8 grams per second with the expander added.

Stack temperature has a drastic effect on both the efficiency and water balance of the fuel cell system. A change in temperature increases one while decreasing the other. So a change in temperature from 30°C to 80°C results in the following at highest fuel cell load:

Turbocompressor:	+11.1 efficiency	-17.3 kg/s water balance
Turbocompressor with exp:	+12.8 efficiency	-0.8 kg/s water balance
Twin-screw:	+10.7 efficiency	-23.1 kg/s water balance
Twin-screw with exp.:	+10.9 efficiency	-20.0 kg/s water balance

The difference in water balance for the two compressor systems with expanders is due to the difference in operating pressure lines. Future work could show this to be true by modifying the operating lines.

The twin-screw compressor operating line can be modified to improve water balance, but at the expense of low load system efficiency. The turbocompressor could also be modified, but at a section of the fuel cell load that already has a positive water balance. The surge line greatly restricts changes in the operating pressure line at low loads. Changing the pressure for the twin-screw compressor does not significantly change the efficiency at the highest fuel cell load when the expander is included in the system. Without the expander, a decrease of about 4 percentage points results. Water balance increases significantly: 11 grams per second without the expander, and 17.9 grams per second with the expander.

References

Barbir, Frano, Bhaskar Balasubramanian, Jay Neutzler, "Trade-off Design Analysis of Operating Pressure and Temperature in PEM Fuel Cell Systems," ASME AES-Vol. 39, 1999

Cunningham, Joshua M., Myron A. Hoffman, David J. Friedman, "A Comparison of High-Pressure and Low-Pressure Operation of PEM Fuel Cell Systems," SAE Paper 2001-01-0538, SAE International, Warrendale, PA, 2001.

Fronk, Matthew H., David L. Wetter, David A. Masten, Andrew Bosco, "PEM Fuel Cell System Solutions for Transportation," SAE Paper 2000-01-0373, in SAE SP-1505, Fuel Cell Power for Transportation 2000, pp. 103-108.

Pischinger, S., C. Schönfelder, W. Bornscheuer, H. Kindl, A. Wiartalla, "Integrated Air Supply and Humidification Concepts for Fuel Cell Systems," SAE Paper 2001-01-0233, SAE International, Warrendale, PA, 2001.

Roan, V.P., H.F. Creveling, C.A. Garris, B.D. Wilson, "Air System Technology Programs Review Panel: Fuel Cell CEM – Preliminary Findings," ANL Air Management Workshop, Oct. 2000.

Sadler, M., A.J. Stapleton, R.P.G. Heath, N.S. Jackson, "Application of Modeling Techniques to the Design and Development of Fuel Cell Vehicle Systems," SAE Paper 2001-01-0542, SAE International, Warrendale, PA, 2001.

Shapiro, Howard N., Michael J. Moran, Fundamentals of Engineering Thermodynamics, John Wiley & Sons, Inc., 1995

Appendix

Instructions for computer programs

High efficiency programs

The most difficult part of using this program is to find the range of pressure and mass flow in which the highest efficiency lies. Equations for the range of pressure and flow are at the beginning of program. The larger the range, the more likely it is that the correct answer will be found, but also makes the program take longer to finish the solution. Vary the variable "max_pvalues" to change the pressure range size and the variable "max_airvalues" to change the flow rate range size. If the user wants to change the location of the range in the pressure or flow rate, modify "PR_knob" or "smair_knob". This will shift the range up or down according to the sign of the number the user chooses.

The data collection variables to use for finding the highest efficiency are "press_test," which is the list of efficiency values for the various pressure and flow rate values, and "enough_press" which is a test variable to make sure the chosen amount of pressure was enough to overcome the system pressure drop. Multiplying "press_test" by "enough_press" will give efficiency values for pressures that were high enough, and zeros for pressures that were too low. These variables are both three dimensional, and their contents are as follows: press_test(flow rate, pressure, current). The same contents apply to "enough_press." So to find the efficiency using the 2^{nd} flow rate value and the 8^{th} pressure value in their respective ranges for the first current value, this command should be used: press_test(2,8,1)*enough_press(2,8,1). A value of zero would mean that the 8^{th} pressure value was not sufficient to overcome the pressure drop across the fuel cell system.

It is suggested that only one current value at a time be tested for the first few times, just to get used to the system. Each current value will produce a matrix of efficiency values corresponding to the ranges of pressure and flow rate chosen. So a test of 30 pressure values and 20 flow values will give a 30 x 20 matrix of efficiency values for that current value. Use the data collection variables mentioned above to make sure the highest efficiency value is within the range. Once the range is found to be correct, the program has already located and saved the maximum values in the variables "store_flow" and "store_press."

An example: hh = 10; index2 = 15

```
>> presstest(:,:,1).*enough_press(:,:,1)

ans =

   0.6291   0.6289   0.6289   0.6287   0.6286   0.6285   0.6283   0.6282   0.6280   0.6280
   0.6319   0.6318   0.6315   0.6314   0.6312   0.6310   0.6309   0.6308   0.6306   0.6304
   0.6336   0.6335   0.6334   0.6331   0.6329   0.6327   0.6326   0.6323   0.6322   0.6320
   0.6349   0.6347   0.6345   0.6343   0.6340   0.6338   0.6336   0.6334   0.6332   0.6330
   0.6358   0.6355   0.6352   0.6350   0.6348   0.6345   0.6343   0.6340   0.6338   0.6336
   0.6363   0.6361   0.6357   0.6355   0.6352   0.6349   0.6346   0.6344   0.6342   0.6338
   0.6367   0.6363   0.6360   0.6357   0.6354   0.6351   0.6348   0.6345   0.6342   0.6339
   0.6368   0.6365   0.6361   0.6358   0.6354   0.6351   0.6348   0.6344   0.6342   0.6338
   0.6368   0.6364   0.6361   0.6357   0.6354   0.6350   0.6346   0.6343   0.6339   0.6336
   0.6367   0.6363   0.6359   0.6355   0.6351   0.6348   0.6344   0.6340   0.6336   0.6333
   0.6365   0.6361   0.6357   0.6353   0.6348   0.6344   0.6340   0.6336   0.6332   0.6328
   0.6363   0.6358   0.6354   0.6349   0.6345   0.6341   0.6336   0.6332   0.6327   0.6323
   0.6360   0.6355   0.6350   0.6345   0.6341   0.6336   0.6331   0.6327   0.6322   0.6318
   0.6356   0.6351   0.6346   0.6341   0.6336   0.6331   0.6326   0.6321   0.6316   0.6312
   0.6353   0.6347   0.6342   0.6336   0.6331   0.6326   0.6321   0.6316   0.6310   0.6305
```

Increasing flow rate

Increasing pressure

The command given shows all of the pressure and flow rate range values of the efficiency at the first current value. So since index2 = 15 and hh = 10, the matrix that results is a 15x10 matrix. The bolded numbers (bold added) are the highest in the matrix. They lie in the middle of the flow rate range, but at one extreme of the pressure range. To ensure that the efficiency values are the highest possible, a lower pressure range needs to be tested. So after subtracting 0.015 from the pressure equation:

```
>> [presstest(:,:,1).*enough_press(:,:,1)]

ans =

   0.6313   0.6311   0.6310   0.6308   0.6307   0.6306   0.6304   0.6302   0.6302   0.6299
   0.6345   0.6343   0.6342   0.6339   0.6338   0.6336   0.6334   0.6333   0.6331   0.6329
   0.6368   0.6365   0.6363   0.6362   0.6360   0.6357   0.6355   0.6353   0.6351   0.6349
        0   0.6382   0.6380   0.6377   0.6375   0.6373   0.6370   0.6368   0.6365   0.6363
        0   0.6395   0.6392   0.6389   0.6387   0.6384   0.6381   0.6378   0.6376   0.6373
        0        0   0.6402   0.6399   0.6396   0.6393   0.6389   0.6387   0.6384   0.6380
        0        0   0.6409   0.6406   0.6402   0.6399   0.6395   0.6392   0.6389   0.6385
        0        0        0   0.6411   0.6407   0.6403   0.6400   0.6396   0.6392   0.6389
        0        0        0   0.6415   0.6411   0.6406   0.6402   0.6399   0.6395   0.6391
        0        0        0        0   0.6413   0.6409   0.6404   0.6400   0.6396   0.6392
        0        0        0        0   0.6414   0.6410   0.6405   0.6401   0.6396   0.6392
        0        0        0        0        0   0.6410   0.6405   0.6401   0.6396   0.6391
        0        0        0        0        0   0.6410   0.6405   0.6400   0.6395   0.6390
        0        0        0        0        0        0   0.6404   0.6399   0.6393   0.6388
        0        0        0        0        0        0   0.6403   0.6397   0.6392   0.6386
```

Increasing flow rate

Increasing pressure

Here it can be seen that the pressure is reaching the pressure drop value for the system, since there are zeros in the output. The highest value is found in these flow rate and pressure values: 0.6415. If a maximum efficiency value is in the middle of the ranges of pressure and flow rate, as shown above, it is not always guaranteed to be the maximum efficiency overall for that current value. At some current values and especially with the expander, the efficiencies may decrease and then increase with increasing pressure. To be safe, large ranges should be checked.

Fuel Simulator Programs

These programs are much more user friendly, and only require that the inputs be adjusted as desired before running. After the program is run, a set of the output is saved in a tab-delimited file. This set can be changed to include or exclude variables, and is located at the end of the program. Also, a variety of output figures can be displayed after the program is run, but these must be turned on in the input section of the program.

A small section of the program holds the highest efficiency lines, and can be replaced by other lines if further work is done on this system.

Computer Programs

Here are the primary programs for this paper. The first program finds the highest efficiency operating lines for the twin-screw compressor.

PROGRAM 1

```
close all
clear all

% This program is used to find the compressor pressure/air flow curve
that gives the highest system efficiency for the
% twin-screw compressor.  Calculations are done using a variety of
pressures and air flow rates, and a single
% combination of these is output for each current value

% Assumptions/conditions:  constant specific heats, variable stack
temperature but fixed stack humidity, fixed fan
% and compressor motor efficiency, ideal gases, fixed relative humidity
and temperature at entrance to compressor

% Variable suffixes
% 1 -- compressor air inlet
% 2 -- humidifier air inlet
% 3 -- fuel cell air inlet
% 4 -- expander air inlet
% 5 -- radiator coolant inlet
% 6 -- fuel cell coolant inlet
% 7 -- humidifier coolant inlet
```

```
%-------------------------------------------------------------------------
% Constants and System Parameters
%-------------------------------------------------------------------------

% physical constants
cp_air = 1004.5;                % specific heat of air (J/kg-K)
cp_h2o = 4200;                  % specific heat of water (J/kg-K)
cp_vap = 1900;                  % specific heat of vapor (J/kg-K)
cp_H2 = 14400;                  % specific heat of hydrogen (J/kg-K)
e_vap = 2383000;                % approx. energy of vaporization of
water at 50°C (J/kg) (Shapiro 724)
H2_HHV = 141890000;             % higher heating value of hydrogen
(J/kg)
H2_LHV = 119860000;             % lower heating value of hydrogen
(J/kg)
k = 1.4;                        % specific heat ratio of air
Ma = 28.97;                     % molar mass of air (kmol/kg)
MH2 = 2.016;                    % molar mass of hydrogen (kmol/kg)
Mh2o = 18.02;                   % molar mass of water (kmol/kg)
MO2 = 32;                       % molar mass of oxygen (kmol/kg)

% 1050 compressor values
adeff_max = 0.675;              % maximum expected adiabatic efficiency
voleff_max = 0.975;             % maximum expected volumetric
efficiency
temp_max = 96.15;               % maximum expected outlet temperature
increase of compressor (K)
speed_max = 16;                 % maximum speed of compressor
(RPM/1000)
BB = [.115 .121 .135 .149 .157 .162 .167 .169;.172 .187 .204 .216 0.224
.232 .242 .254;.229 .254 .271 .281 .293 .3 .310 .319;.288 .314 .334
.346 .359 .359 .367 .377;.348 .372 .396 .41 .419 .416 .425 .436;.409
.432 .459 .471 .481 .478 .487 .497;.477 .494 .523 .533 .544 .539 .548
.558;.543 .556 .584 .595 .607 .599 .608 .618;.596 .618 .642 .654 .667
.658 .668 .679;.649 .681 .701 .715 .725 .721 .730 .742;.706 .741 .762
.777 .785 .783 .792 .806;.764 .803 .823 .839 .847 .843 .849 .872;.831
.872 .883 .899 .908 .902 .905 .939;.911 .942 .951 .958 .967 .971 .972
1.007;.989 1.012 1.019 1.016 1.026 1.051 1.049 1.076;1.067 1.081 1.087
1.074 1.085 1.131 1.125 1.144]; %speed matrix for Opcon 1050 -- last
modified 2/16/01
CC = [.535 .715 .711 .566 .424 .321 .243 .161;.546 .769 .809 .741 .663
.618 .6 .571;.558 .824 .894 .859 .816 .779 .751 .703;.555 .849 .944
.913 .877 .839 .807 .758;.526 .867 .968 .95 .920 .881 .851 .801;.497
.87 .983 .971 .954 .909 .883 .836;.472 .860 .993 .986 .963 .936 .909
.863;.447 .844 .993 .996 .963 .945 .918 .876;.425 .814 .988 .997 .963
.954 .927 .887;.403 .78 .975 .986 .963 .946 .925 .887;.382 .729 .957
.971 .963 .936 .923 .886;.361 .679 .926 .957 .955 .926 .919 .871;.337
.644 .885 .937 .933 .917 .916 .857;.306 .608 .853 .909 .905 .904 .906
.858;.283 .569 .821 .881 .877 .886 .891 .859;.262 .531 .789 .854 .849
.868 .875 .86]; %adiabatic efficiency matrix for Opcon 1050 -- last
modified 2/17/01
DD = [.688 .617 .526 .47 .437 .416 .397 .386;.756 .682 .617 .58 .558
.541 .521 .498;.823 .747 .698 .672 .647 .632 .612 .592;.877 .798 .756
.73 .705 .69 .672 .653;.905 .846 .8 .772 .754 .738 .721 .701;.931 .878
.828 .806 .791 .771 .756 .74;.944 .896 .849 .832 .815 .802 .788
.773;.957 .911 .869 .852 0.835 .824 0.81 .797;.968 .921 .888 .871 .855
.845 .833 .819;.978 .931 .905 .887 .874 .858 .846 .832;.987 .94 .915
```

```
.897 .888 .869 .86 .845;.995 .948 .925 .907 .898 .881 .874 .851;.993
.945 .933 .917 .908 .892 .889 .857;.975 .942 .933 .926 .918 .894 .893
.86;.962 .94 .933 .936 .927 .884 .887 .863;.951 .938 .933 .946 .937
.875 .88 .867]; %volumetric efficiency matrix for Opcon 1050 -- last
modified 2/17/01

% approx. 60 kW stack (honeywell)
num_cell = 210;                 % number of cells per stack
area_cell = 678;                % cell effective area (cm^2)
maxfan = 250;                   % max power of fan (W)
maxpump = 900;                  % max power of pump (W) -- determined
from Price Pump CD-100, 3600 RPM rating, 4.25" impeller diameter
maxpumpflow = 3.15;             % max mass flow of pump (kg/s) --
determined from Price Pump (see above)
resist = 3.68;                  % flow resistance for fuel cell system
to achieve 0.4 atm drop at max flow (atm-s/kg)
radair_on = 3.63;               % mass flow of air through radiator
with fan on (m^3/s)
radair_off = 0.34;              % mass flow of air through radiator
with fan off (m^3/s)
UA_on = 5100;                   % radiator UA value with fan on
UA_off = 510;                   % radiator UA value with fan off

% input values
Amax = 475;                     % maximum current for fuel cell (A)
Amin = 5;                       % minimum current for fuel cell (A)
Astep = 5;                      % step size for current loop (A)
T1 = 298;                       % temperature entering compressor (K)
T3 = 353;                       % desired temperature of input to stack
(K)
T3_init = T3;                   % record of original T3 value (T3
changes in program)
T4 = T3;                        % output temperature of stack
T4_init = T4;                   % stores original T4 value
T_h2o = T4+5;                   % assumed temperature of water taken
for humidification (stack coolant outlet)
T_h2o_init = T_h2o;             % stores original T_h2o value
P1 = 100000/101325;             % pressure entering compressor (atm)
RH1 = 0.5;                      % relative humidity entering compressor
RH3 = 0.75;                     % relative humidity entering fuel cell
-- use whole percentage points
RH3_init = RH3;                 % record of original RH3 value
RH_H2 = 1;                      % hydrogen humidity entering fuel cell
SR_min = 2;                     % minimum stoichiometric ratio
SR_max = 2.5;                   % maximum stoichiometric ratio
nm = 0.88;                      % combined compressor motor and
inverter efficiency
nfan = 0.65;                    % fan efficiency
max_pvalues = 2;                % maximum number of pressure values
examined for efficiency
max_airvalues = 20;             % maximum number of airflow values
examined for efficiency
PR_knob = -.005;                % adjusts pressure range in examination
smair_knob = -2;                % adjusts airflow range in examination

% minimum parasitics
minpumpflow = 0.2;              % minimum coolant flow (kg/s)
```

```
min_smair = 0;                  % minimum standard mass flow of air
(SLPS)

%-----------------------------------------------------------------------
% Initial Calculations
%-----------------------------------------------------------------------

den_air = Ma*P1*101325/(8314*T1);       % approx. density of dry air
entering compressor (kg/m^3)
den_vap = Mh2o*P1*101325/(8314*T1);     % ideal gas density of water
vapor at input conditions (kg/m^3)
Pg1 = 10^(2.95E-2*(T1-273)-9.18E-5*(T1-273)^2+1.44E-7*(T1-273)^3-2.18);
% inlet saturation pressure (T in K and P_sat in atm)

% this section determines the maximum airflow rate for normalization
purposes
mH2_max = num_cell*475*MH2/(96487*2*1000);                  % kg H2/s
(Fuel Cell Handbook, 8-2)
mO2_max = 0.5*(mH2_max*MO2)/MH2;                            % kg O2/s
(Shapiro, 558, equ. 12.1)
maxair_req = (mO2_max*Ma)/(MO2*0.21);                       % kg dry
air/s required for reaction (ibid, mole fraction of O2 from Shapiro
559)
mair1_max = SR_max*maxair_req;                              % actual kg
dry air/s flowing through compressor
mvap1_max = 0.622*mair1_max*Pg1*RH1/(P1-Pg1*RH1);           % mass flow
of h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
max_smair =(mair1_max/den_air + mvap1_max/den_vap)*1000;    % mass flow
of dry air and vapor(standard l/s)
if max_smair<min_smair                                      % makes
sure maximum is not less than minimum
    max_smair = min_smair;
end

% fuel cell current loop
index = 0;                                                  % place-
keeping variable
for current = Amin:Astep:Amax
    index = index + 1;

    SR = 3.381e-6*(current^2)-6e-4*current+2.009;           % operating
line for air stoichiometric ratio--60 kW
                                                            % equation
taken from a curve fit for the following:
                                                            % current:
300 210 120 30
                                                            % SR:
2.5 2.2 2.0 2.0
    if SR<SR_min                                            % makes
sure stoichiometric ratio stays above two and below the maximum
        SR = SR_min;
    elseif SR>SR_max
        SR = SR_max
    end

    % fuel cell fuel and air consumption values
```

```
    mH2 = num_cell*current*MH2/(96487*2*1000);              % kg H2/s
(Fuel Cell Handbook, 8-2)
    mO2 = 0.5*(mH2*MO2)/MH2;                                % kg O2/s
(Shapiro, 558, equ. 12.1)
    mair_req = (mO2*Ma)/(MO2*0.21);                         % kg dry
air/s required for reaction (ibid, mole fraction of O2 from Shapiro
559)
    mair1 = SR*mair_req;                                    % actual kg
dry air/s flowing through compressor
    mvap1 = 0.622*mair1*Pg1*RH1/(P1-Pg1*RH1);               % mass flow
of h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
    smair =(mair1/den_air + mvap1/den_vap)*1000;            % mass flow
of dry air and vapor(standard l/s)
    if smair<min_smair                                      % makes
sure standard mass flow does not go below minimum
        smairold = smair;
        smair = min_smair;
        mair1 = smair/smairold*mair1;                       % converts
from old mair1 and mvap1 values to new values
        mvap1 = smair/smairold*mvap1;
    end
    smair_orig = smair;                                     % stores
original smair value for reference

    % pressure and flow rate matrices
    P2 = 0.000115*smair^2 + 0.0023*smair + 1.1793;          % estimated
high efficiency operating pressure line
    P2_orig = P2;                                           % stores
original P2 value for reference
    for index2 = 1:max_airvalues                            % fuel cell
air flow loop (changing air flow)
        smairold = smair;
        smair = smair_orig + smair_knob  + index2*0.1;      % modified
airflow rate (kg/s)
        mair1 = smair/smairold*mair1;                       % modified
air intake rate (kg/s)
        mvap1 = smair/smairold*mvap1;                       % modified
vapor intake rate (kg/s)
        hh = 1:max_pvalues;                                 % number of
pressure values examined
        P2 = P2_orig + PR_knob + hh*0.001;                  % fuel cell
pressure cluster
        PR = P2/P1;
        for jj = 1:max(hh)                                  % this loop
raises the fuel cell pressure to at least atmospheric pressure
            if PR(jj)<1                                     % increases
pressure to at least ambient
                PR(jj) = 1;
                P2(jj) = PR(jj)*P1;
            end
        end

        % initializing variables
        for jj = 1:max(hh)
            T3(jj) = T3_init;
            T4(jj) = T4_init;
            T_h2o(jj) = T_h2o_init;
```

```
        end

%-------------------------------------------------------------------------
% Compressor Section
%-------------------------------------------------------------------------

        smair_norm = smair/max_smair;           % normalized air flow
through compressor
        out = 0;                                % initialization of
flag
        index1 = 0;                             % initialization of
index value
        P2_alt = [0 0];                         % initialization of
pressure outside of data range
        smair_alt = [0 0];                      % initialization of air
flow outside of data range
        alt_index = zeros(1,max(hh));           % initialization of
index of which data points are out of data range

        for jj = 1:max(hh)                      % loop for finding
which data points are out of range
            if smair_norm<.0625 | smair_norm>1 | P2(jj)<1.1 |
P2(jj)>2.7  % test if data point is out of data range
                out = 1;                        % this flag shows that
there is a data point in this series that is out of range
                if smair_norm<.0625             % this loop and nested
loops determine where the data point is
                    if P2(jj)<1.1               % in relation to the
known data set
                        index1 = index1 + 1;

[smairz,pz]=linear(smair_norm,P2(jj),0.0625,1.1);  % here, a point is
assigned (smairz) which is linear
                        smair_alt(index1,:) = [0.0625 smairz];
% with the outside data point and a point on
                        P2_alt(index1,:) = [1.1 pz];
% the edge of the known data.  This is used
                        alt_index(index1) = jj;
% later to find extrapolated values for
                    elseif P2(jj)>2.7
% the outside data point
                        index1 = index1 + 1;
                        [smairz,pz] =
linear(smair_norm,P2(jj),0.0625,2.7);
                        smair_alt(index1,:) = [0.0625 smairz];
                        P2_alt(index1,:) = [2.7 pz];
                        alt_index(index1) = jj;
                    else
                        index1 = index1 + 1;
                        [smairz,pz] =
linear(smair_norm,P2(jj),0.0625,P2(jj));
                        smair_alt(index1,:) = [0.0625 smairz];
                        P2_alt(index1,:) = [P2(jj) pz];
                        alt_index(index1) = jj;
                    end
                elseif smair_norm>1
                    if P2(jj)<1.1
```

```
                        index1 = index1 + 1;
                        [smairz,pz] = linear(smair_norm,P2(jj),1,1.1);
                        smair_alt(index1,:) = [1 smairz];
                        P2_alt(index1,:) = [1.1 pz];
                        alt_index(index1) = jj;
                    elseif P2(jj)>2.7
                        index1 = index1 + 1;
                        [smairz,pz] = linear(smair_norm,P2(jj),1,2.7);
                        smair_alt(index1,:) = [1 smairz];
                        P2_alt(index1,:) = [2.7 pz];
                        alt_index(index1) = jj;
                    else
                        index1 = index1 + 1;
                        [smairz,pz] =
linear(smair_norm,P2(jj),1,P2(jj));
                        smair_alt(index1,:) = [1 smairz];
                        P2_alt(index1,:) = [P2(jj) pz];
                        alt_index(index1) = jj;
                    end
                elseif P2(jj)<1.1
                    index1 = index1 + 1;
                    [smairz,pz] =
linear(smair_norm,P2(jj),smair_norm,1.1);
                    smair_alt(index1,:) = [smair_norm smairz];
                    P2_alt(index1,:) = [1.1 pz];
                    alt_index(index1) = jj;
                elseif P2(jj)>2.7
                    index1 = index1 + 1;
                    [smairz,pz] =
linear(smair_norm,P2(jj),smair_norm,2.7);
                    smair_alt(index1,:) = [smair_norm smairz];
                    P2_alt(index1,:) = [2.7 pz];
                    alt_index(index1) = jj;
                end
            end
        end
        index_a = 1;
        index_b = 1;
        P2_ir = zeros(1,max(hh));
        for index5 = 1:max(hh)           % this loop gathers the data
points that are in range
            if alt_index(index_a) == index5     % tests to see if point
is out of range.  If so, the point is skipped
                index_a = index_a + 1;
            else
                P2_ir(index_b) = P2(index5);           %ir = in range
                index_b = index_b + 1;
            end
        end

        % comp_speed = compressor speed (RPM)
        % eff_overall = adiabatic efficiency of compressor
        % temp_rise = temperature rise in compressor above ambient (K)
        % vol_eff = volumetric efficiency of compressor

        if out         % this if statement finds values for those
points outside of the data range
```

```
            % these first four equations find the compressor values for
the two points inside of the data range which are
            % linear with the data point outside of the data range.  A
2D lookup table is used.
            comp_speed_alt = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P2_alt,smair_alt,'cubic');  % compressor speed
values
            eff_overall_alt = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P2_alt,smair_alt,'cubic'); % compressor
adiabatic efficiency values
            temp_rise_alt = interp1([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.084 .237 .4375 .56 .6686 .77 .866 1],P2_alt,'cubic');
%compressor temperature rise values
            vol_eff_alt = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],DD,P2_alt,smair_alt,'cubic');     % compressor
volumetric efficiency values
            % these next four equations extrapolate the compressor
values for the data point outside of the data range from
            % the other two that are within the data range using linear
extrapolation.
            comp_speed_oor =
linear2(P2_alt,smair_alt,comp_speed_alt,P2,smair_norm,alt_index);  %oor
= out of range
            eff_overall_oor =
linear2(P2_alt,smair_alt,eff_overall_alt,P2,smair_norm,alt_index);
            temp_rise_oor = linear3(P2_alt,temp_rise_alt,P2,alt_index);
            vol_eff_oor =
linear2(P2_alt,smair_alt,vol_eff_alt,P2,smair_norm,alt_index);
            if max(P2_ir)>0     %this if statement finds values for the
points that are in range, if there are any that are in range
                comp_speed_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P2_ir,smair_norm,'cubic');
                eff_overall_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P2_ir,smair_norm,'cubic');
                temp_rise_norm = INTERP1([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.084 .237 .4375 .56 .6686 .77 .866 1],P2_ir,'cubic');
                vol_eff_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],DD,P2_ir,smair_norm,'cubic');
                index_a = 1;
                index_b = 1;
                for jj = 1:max(hh)    % this loop combines the out-of-
range and in-range data into single variables
                    if alt_index(index_a) == jj
                        comp_speed(jj) = comp_speed_oor(index_a) *
speed_max;
                        eff_overall(jj) = eff_overall_oor(index_a) *
adeff_max;
                        temp_rise(jj) = temp_rise_oor(index_a) *
temp_max;
                        vol_eff(jj) = vol_eff_oor(index_a) *
voleff_max;
```

```
                    index_a = index_a + 1;
                else
                    comp_speed(jj) = comp_speed_norm(index_b) *
speed_max;
                    eff_overall(jj) = eff_overall_norm(index_b) *
adeff_max;
                    temp_rise(jj) = temp_rise_norm(index_b) *
temp_max;
                    vol_eff(jj) = vol_eff_norm(index_b) *
voleff_max;
                    index_b = index_b + 1;
                end
            end
        else   % if all data points are out of range, these
equations are used to remove the normalization
            comp_speed = comp_speed_oor * speed_max;
            eff_overall = eff_overall_oor * adeff_max;
            temp_rise = temp_rise_oor * temp_max;
            vol_eff = vol_eff_oor * voleff_max;
        end
    else       % if all data points are in range, these equations
are used
        comp_speed_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P2,smair_norm,'cubic');
        eff_overall_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P2,smair_norm,'cubic');
        temp_rise_norm = INTERP1([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.084 .237 .4375 .56 .6686 .77 .866 1],P2,'cubic');
        vol_eff_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],DD,P2,smair_norm,'cubic');
        comp_speed = comp_speed_norm*speed_max;
        eff_overall = eff_overall_norm*adeff_max;
        temp_rise = temp_rise_norm*temp_max;
        vol_eff = vol_eff_norm*voleff_max;
    end
    T2 = T1+temp_rise;                                          %
Compressor outlet temperature (K)
    Pwr_c = ((mair1/Ma + mvap1/Mh2o)*8314*T1*k/(k-1)*((P2/P1).^((k-
1)/k)-1))./(eff_overall*nm);    % Compressor power

    for jj = 1:max(hh)                                  % this loop
takes arrays (i.e. P2, T3, T4, etc.) 1 element at a time
        % approximate fuel cell calculations to make guess at
Q_stack (stack heat rejection) and m_cool (coolant flow rate)
        pO2 = P2(jj)*0.21;                              % partial
pressure of oxygen in dry air (atm)
        J = current/area_cell;                          % current
density (V/cm^2)
        V_guess = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4(jj))*J+0.14*log10(pO2)+(T4(jj)-333)*0.00041*log10(10^3*J);  %
revised Honeywell curve with low temp. effects (5/2/01)
        Pwr_f_guess = V_guess*num_cell*current;         % stack
power (W)
```

```
            Q_guess = mH2*H2_HHV-Pwr_f_guess;               % stack
heat rejection (W)

            % uses a 50gpm pump with max power of 1.2 bHP (CD-100 from
Pricepump)
            m_cool = Q_guess/(10*cp_h2o);                   % coolant
flow giving a 10°C temp diff across stack (kg/s)
            Pwr_pump(jj) = maxpump/maxpumpflow*m_cool;      % linear
power with max determined by Price Pump website (W)
            if Pwr_pump(jj)>maxpump                         % checks to
see if the pump power exceeds the maximum
                Pwr_pump(jj) = maxpump;                     % if the
power does exceed the max, it is set to the maximum power (W)
                m_cool = maxpumpflow;                       % also the
coolant flow for the pump is set to the maximum flow (kg/s)
                'stack will overheat -- reached max of coolant pump'
            elseif m_cool<minpumpflow                       % checks if
the pump coolant flow is below the minimum
                m_cool = minpumpflow;                       % if the
flow is less than the minimum, the flow is set to the minimum
                Pwr_pump(jj) = maxpump/maxpumpflow*m_cool;        %
linear power with max determined by Price Pump website (W)
                T3(jj) = T3_init + Q_guess/(2*m_cool*cp_h2o) - 5;  %
temperature achievable in humidifier air stream (K) (assumes 5° min.
difference btwn coolant and air)
                T_h2o(jj) = T4(jj) + Q_guess/(2*m_cool*cp_h2o);    %
coolant temperature (K)
            end

%-----------------------------------------------------------------------
% Humidifier/Heat Exchanger Section
%-----------------------------------------------------------------------

            P3 = P2(jj);            % no pressure drop over humidifier
            cont = 0;
            while ~cont             % this loop checks to make sure the
humidifier doesn't use more heat than is given by stack
                RH3 = RH3_init;     % initializing RH3

                % hydrogen humidification
                Pg4 = 10^(2.95E-2*(T4(jj)-273)-9.18E-5*(T4(jj)-
273)^2+1.44E-7*(T4(jj)-273)^3-2.18); % water sat. pressure for H2 (atm)
                mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);
% water vapor in H2 (kg/s) (assumes initially dry H2)

                % air humidification
                Pg3 = 10^(2.95E-2*(T3(jj)-273)-9.18E-5*(T3(jj)-
273)^2+1.44E-7*(T3(jj)-273)^3-2.18); % saturation pressure in fuel cell
inlet (atm)
                if T3(jj)==T4(jj)
                    mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);       %
vapor mass flow going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                      %
net water added in humidifier (kg/s)
                    mtot3 = mair1 + mvap3;
% total mass flow into fuel cell (kg/s)
                else  % effects of coolant loop minimum
```

```
                    mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);       %
vapor mass needed to humidify air to correct RH when warmed to fuel
cell temp (kg/s) (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                      %
net water added in humidifier (kg/s)
                    mtot3 = mair1 + mvap3;                          %
total mass flow into fuel cell (kg/s)
                    RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);     %
relative humidity of air at humidifier exit
                    if RH3>1
                        'lower than requested fuel cell inlet air RH'
                        RH3 = 1;
                        mvap3 = 0.622*mair1*Pg3/(P3-Pg3);           %
vapor mass going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;                  %
net water added in humidifier (kg/s)
                        mtot3 = mair1 + mvap3;                      %
total mass flow into fuel cell (kg/s)
                        Pg3_in = 10^(2.95E-2*(T4(jj)-273)-9.18E-
5*(T4(jj)-273)^2+1.44E-7*(T4(jj)-273)^3-2.18);  % saturation pressure
entering fuel cell (atm)
                        RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);   % relative humidity of air
after warming up in fuel cell
                    end
                end
                Q_needed = mair1*cp_air*(T3(jj)-T2(jj)) +
mvap1*cp_vap*(T3(jj)-T2(jj)) + mh2o_addh*e_vap +
mh2o_addh*cp_vap*(T3(jj)-T_h2o(jj));  % total heat needed to warm mass
flow into fuel cell (W)

                while Q_needed<Q_guess & T4(jj)<T4_init             %
raises the temp of stack if heat is available
                    RH3 = RH3_init;                                 %
initializing RH3
                    T4(jj) = T4(jj) + .08;                          %
increases operating temp of fuel cell (K)
                    if T4(jj)>T4_init                               %
makes sure T4 doesn't go over the desired operating temp
                        T4(jj) = T4_init;                           %
sets T4 to the initial desired fuel cell temperature (K)
                    end
                    T3(jj) = T4(jj) + Q_guess/(2*m_cool*cp_h2o) - 5;
% temperature achievable in humidifier air stream (K) (assumes 5° min.
difference btwn coolant and air)
                    T_h2o(jj) = T4(jj) + Q_guess/(2*m_cool*cp_h2o);
% water temp same as stack coolant exit temp (K)
                    Pg3 = 10^(2.95E-2*(T3(jj)-273)-9.18E-5*(T3(jj)-
273)^2+1.44E-7*(T3(jj)-273)^3-2.18);    % saturation pressure in fuel
cell inlet (atm)
                    Pg4 = 10^(2.95E-2*(T4(jj)-273)-9.18E-5*(T4(jj)-
273)^2+1.44E-7*(T4(jj)-273)^3-2.18);    % saturation pressure in fuel
cell inlet (atm)
                    mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 -
Pg4*RH_H2);% water vapor in H2 (assumes initially dry H2 and can make
it to fc temp)
                    if T3(jj)==T4(jj)
```

```
                        mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);
% vapor mass going into fuel cell (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;
% net water added in humidifier (kg/s)
                        mtot3 = mair1 + mvap3;
% total mass flow into fuel cell (kg/s)
                    else    % effects of coolant loop minimum
                        mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);
% vapor mass needed to humidify air to correct RH when warmed to fuel
cell temp (kg/s) (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;
% net water added in humidifier (kg/s)
                        mtot3 = mair1 + mvap3;
% total mass flow into fuel cell (kg/s)
                        RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);
% relative humidity of air at humidifier exit
                        if RH3>1
                            'lower than requested fuel cell inlet air
RH'
                            RH3 = 1;
                            mvap3 = 0.622*mair1*Pg3/(P3-Pg3);
% vapor mass going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                            mh2o_addh = mvap3 - mvap1;
% net water added in humidifier (kg/s)
                            mtot3 = mair1 + mvap3;
% total mass flow into fuel cell (kg/s)
                            Pg3_in = 10^(2.95E-2*(T4(jj)-273)-9.18E-
5*(T4(jj)-273)^2+1.44E-7*(T4(jj)-273)^3-2.18); % saturation pressure in
fuel cell inlet (atm)
                            RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  % relative humidity of air
after warming up in fuel cell
                        end
                    end
                    Q_needed = mair1*cp_air*(T3(jj)-T2(jj)) +
mvap1*cp_vap*(T3(jj)-T2(jj)) + mh2o_addh*e_vap +
mh2o_addh*cp_vap*(T3(jj)-T_h2o(jj)); % total heat needed in humidifier
(W)
                end
                temp_mod = 0;                   % flag to show
temperature modification
                while Q_needed>Q_guess          % lowers stack
temperature to allow humidifier to heat to temperature
                    temp_mod = 1;
                    RH3 = RH3_init;             % initializing RH3
                    T4(jj) = T4(jj) - .08;      % incremental decrease
in stack ave temperature
                    T3(jj) = T4(jj) + Q_guess/(2*m_cool*cp_h2o) - 5;
% temperature achievable in humidifier air stream (K) (assumes 5° min.
difference btwn coolant and air)
                    T_h2o(jj) = T4(jj) + Q_guess/(2*m_cool*cp_h2o);
% water temp same as stack coolant exit temp (K)
                    Pg3 = 10^(2.95E-2*(T3(jj)-273)-9.18E-5*(T3(jj)-
273)^2+1.44E-7*(T3(jj)-273)^3-2.18);    % saturation pressure in fuel
cell inlet (atm)
```

```
                    Pg4 = 10^(2.95E-2*(T4(jj)-273)-9.18E-5*(T4(jj)-
273)^2+1.44E-7*(T4(jj)-273)^3-2.18);    % saturation pressure in fuel
cell inlet (atm)
                    mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 -
Pg4*RH_H2);    % water vapor in H2 (kg/s) (assumes initially dry H2 and
can make it to fc temp)
                    if T3(jj)==T4(jj)
                        mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);
% vapor mass going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;
% net water added in humidifier (kg/s)
                        mtot3 = mair1 + mvap3;
% total mass flow into fuel cell (kg/s)
                    else  % effects of coolant loop minimum
                        mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);
% vapor mass needed to humidify air to correct RH when warmed to fuel
cell temp (kg/s) (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;
% net water added in humidifier (kg/s)
                        mtot3 = mair1 + mvap3;
% total mass flow into fuel cell (kg/s)
                        RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);
% relative humidity of air at humidifier exit
                        if RH3>1
                            'lower than requested fuel cell inlet air
RH'
                            RH3 = 1;
                            mvap3 = 0.622*mair1*Pg3/(P3-Pg3);
% vapor mass going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                            mh2o_addh = mvap3 - mvap1;
% net water added in humidifier (kg/s)
                            mtot3 = mair1 + mvap3;
% total mass flow into fuel cell (kg/s)
                            Pg3_in = 10^(2.95E-2*(T4(jj)-273)-9.18E-
5*(T4(jj)-273)^2+1.44E-7*(T4(jj)-273)^3-2.18); % saturation pressure of
air before warming up in fuel cell (atm)
                            RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  % relative humidity of air
after warming up in fuel cell
                        end
                    end
                    Q_needed = mair1*cp_air*(T3(jj)-T2(jj)) +
mvap1*cp_vap*(T3(jj)-T2(jj)) + mh2o_addh*e_vap +
mh2o_addh*cp_vap*(T3(jj)-T_h2o(jj)); %total heat needed to warm mass
flow into fuel cell (W)
                    % Q_needed shows amount of heat needed to warm air
stream to T3
                end
                Q_hum = Q_needed;

%-----------------------------------------------------------------------
% Fuel Cell Section
%-----------------------------------------------------------------------

                Pdrop(jj) = resist*(mair1*den_air + (mtot3-
mair1)*den_vap); % pressure loss across system (atm)
```

```
                P4(jj) = P3 - Pdrop(jj);                                % 
pressure at exit of fuel cell (atm)
                PO2_3 = (P3 - RH3*Pg3)*0.21;                            % 
partial pressure of oxygen at fuel cell inlet (atm)
                mO2_used = (num_cell*current*MO2)/(96487*4000);         % 
mass flow of oxygen consumed (kg/s)
                mair4(jj) = mair1 - mO2_used;                           % 
total mass flow of dry air at fuel cell exit (kg/s)
                O2_ratio4 = (mair1*0.21-mO2_used)/mair4(jj);            % 
ratio of oxygen to dry air at fuel cell exit
                mvapmax = 0.622*mair1*Pg4/(P4(jj)-Pg4);                 % 
total vapor at 100% RH at fuel cell outlet (kg/s)
                mvap_fc = mvapmax - mvap3;                              % 
total vapor acquired by airstream from fuel cell water production 
(kg/s)
                mh2o_prod = mH2*Mh2o/MH2;                               % 
stack water production (kg/s)
                h2o_used = 0;                                           % 
flag to show if all of the water in the fuel cell was evaporated in air
                if mvap_fc>mh2o_prod                                    % 
checks to see if fuel cell air would accept more water than the fuel 
cell produces
                    h2o_used = 1;
                    mvap_fc = mh2o_prod;   % if production of water is 
less than total that the air can accept for 100% humidity, then air 
will only take total production of water
                    RH4 = ((mvap3+mvap_fc)*(P3-
Pdrop(jj)))/((0.622*mair1+mvap3+mvap_fc)*Pg4); % relative humidity of 
air at fuel cell outlet
                else
                    RH4 = 1;
                    h2o_used = 0;
                end
                PO2_4 = (P4(jj) - RH4*Pg4)*O2_ratio4;                   % 
partial pressure of oxygen at fuel cell exit (atm)
                pO2 = (PO2_3 + PO2_4)/2;                                % 
average partial pressure of oxygen (atm)
                J = current/area_cell;                                  % 
current density (A/cm^2)
                V = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4(jj))*J+0.14*log10(pO2)+(T4(jj)-333)*0.00041*log10(10^3*J);
%revised Honeywell curve with temp. effects (V) (5/2/01)
                if V <0.5
                    'data for this current level is bad'
                end
                Vstack = V * num_cell;                                  % 
stack voltage (V)
                Pwr_f(jj) = Vstack*current;                             % 
stack power (W)
                mvap4(jj) = mvap3 + mvap_fc;                            % 
total water leaving fuel cell (kg/s)
                n_stack = Pwr_f(jj)/(mH2*H2_LHV);                       % 
stack efficiency
                Q_stack = mH2*(1 - mvap_fc/mh2o_prod)*H2_HHV + 
mH2*(mvap_fc/mh2o_prod)*H2_LHV - Pwr_f(jj);  % stack heat rejection (W)
                mh2o_net = mh2o_prod - mvap_fc;                         % 
water recoverable as liquid at fuel cell outlet (kg/s)
```

```
                if ((Q_guess/Q_stack)>1.01|(Q_guess/Q_stack)<.99)   %
compares actual stack heat rejection with guess
                    if (Q_guess/Q_stack)<.99                        %
this only allows an increase of stack temp if there is more heat
available from stack
                        T4(jj) = T4(jj)+.05;
                        if T4(jj) > T4_init
                            T4(jj) = T4_init;
                        end
                        T3(jj) = T4(jj);
                        T_h2o(jj) = T4(jj)+5;
                    end
                    Q_guess = Q_stack;
% old stack heat rejection becomes new guess
                    cont = 0;
                    % another adjustment of flow rate and check for
exceeding the flow/power limits
                    m_cool = Q_guess/(10*cp_h2o);
% coolant flow giving a 10°C temp diff across stack (kg/s)
                    Pwr_pump(jj) = maxpump/maxpumpflow*m_cool;
% linear power with max determined by Price Pump website (W)
                    if Pwr_pump(jj)>maxpump
                        Pwr_pump(jj) = maxpump;
                        m_cool = maxpumpflow;
                    elseif m_cool<minpumpflow
                        m_cool = minpumpflow;
                        Pwr_pump(jj) = maxpump/maxpumpflow*m_cool;
% linear power with max determined by Price Pump website (W)
                        T_h2o(jj) = T4(jj) + Q_guess/(2*m_cool*cp_h2o);
% stack coolant exit temp (K)
                        T3(jj) = T4(jj) + Q_guess/(2*m_cool*cp_h2o) -
5; % max temperature achievable in humidifier air stream (K) (assumes
5° min. difference btwn coolant and air)
                    end
                else
                    cont = 1;
                end
            end %while (in humidifier)
            press_check(jj) = 1;        % '1' stands for good, that is,
enough pressure to overcome pressure drop over system
            if (P2(jj)-Pdrop(jj))<P1    % checks for enough pressure to
overcome pressure drop over system
                press_check(jj) = 0;    % '0' stands for bad, that is,
not enough pressure to overcome pressure drop over system
            end

%-----------------------------------------------------------------------
% Radiator Section
%-----------------------------------------------------------------------

            % the NTU method is used for the radiator calculations
            T5 = T4(jj) - Q_hum/(m_cool*cp_h2o);    % temperature of
coolant entering radiator (K)
            if T5 < T1
                'Warning!  Coolant temp. entering radiator is lower
than ambient'
            end
```

```
            Q_rad = Q_stack - Q_hum;                % heat that must be
rejected by radiator (W)

            % radiator/air properties with fan on
            m_on = radair_on*den_air;
            C_on = m_on*cp_air;
            C_h2o = m_cool*cp_h2o;
            C1 = [C_on C_h2o];
            Cr_on = min(C1)/max(C1);
            NTU_on = UA_on/min(C1);
            eff_on = 1-exp((1/Cr_on)*NTU_on^.22*(exp(-
Cr_on*NTU_on^.78)-1));
            Qmax_on = min(C1)*(T5-T1);
            Q_on = eff_on*Qmax_on;

            % radiator/air properties with fan off
            m_off = radair_off*den_air;
            C_off = m_off*cp_air;
            C_h2o = m_cool*cp_h2o;
            C2 = [C_off C_h2o];
            Cr_off = min(C2)/max(C2);
            NTU_off = UA_off/min(C2);
            eff_off = 1-exp((1/Cr_off)*NTU_off^.22*(exp(-
Cr_off*NTU_off^.78)-1));
            Qmax_off = min(C2)*(T5-T1);
            Q_off = eff_off*Qmax_off;

            bypass_pcnt = 0;
            fan_timeon = (Q_rad-Q_off)/(Q_on-Q_off);    % percent of
operating time with fan on
            if fan_timeon < 0                           % tests if fan
can be turned off and bypass valve activated
                bypass_pcnt = 1 - Q_rad/Q_off;          % percent of
flow diverted through bypass valve
                fan_timeon = 0;
            elseif fan_timeon > 1                       % limits fan to
100% time on
                fan_timeon = 1;
            end
            avePwr_fan(jj) = (fan_timeon*maxfan)/nfan;  % fan power (W)
            Q_out = Q_on*fan_timeon + Q_off*(1-fan_timeon)*(1-
bypass_pcnt);    % heat rejected from radiator (W)
            T6 = T5 - Q_out/(m_cool*cp_h2o);            % energy
balance, temp of coolant entering fuel cell (K)
            T7 = T6 + Q_stack/(m_cool*cp_h2o);          % energy
balance, temp of coolant exiting fuel cell and entering humidifier (K)
        end % arrays (pressure)

%-------------------------------------------------------------------------
% Expander Section
%-------------------------------------------------------------------------

        smair_exp =(mair4/den_air + mvap4/den_vap)*1000;    % mass flow
of dry air and vapor (standard l/s)
        smair_exp_norm = smair_exp/max_smair;               %
normalization of air going through expander
```

```
        out_e = 0;                                              %
initialization of flag
        index3 = 0;                                             %
initialization of index value
        P4_alt = [0 0];                                         %
initialization of pressure outside of data range
        smair_exp_alt = [0 0];                                  %
initialization of air flow outside of data range
        alt_exp_index = zeros(1,max(hh));                       %
initialization of index of which data points are out of data range

        for jj = 1:max(hh)                                      % this loop
determines what input points are out of the data range
            if smair_exp_norm(jj)<.0625 | smair_exp_norm(jj)>1 |
P4(jj)<1.1 | P4(jj)>2.7
                out_e = 1;                                      % this
section is similar to compressor section; see compressor for details
                if smair_exp_norm(jj)<.0625
                    if P4(jj)<1.1
                        index3 = index3 + 1;
                        [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),0.0625,1.1);
                        smair_exp_alt(index3,:) = [.0625 smairz];
                        P4_alt(index3,:) = [1.1 pz];
                        alt_exp_index(index3) = jj;
                    elseif P4(jj)>2.7
                        index3 = index3 + 1;
                        [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),0.0625,2.7);
                        smair_exp_alt(index3,:) = [.0625 smairz];
                        P4_alt(index3,:) = [2.7 pz];
                        alt_exp_index(index3) = jj;
                    else
                        index3 = index3 + 1;
                        [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),0.0625,P4(jj));
                        smair_exp_alt(index3,:) = [.0625 smairz];
                        P4_alt(index3,:) = [P4(jj) pz];
                        alt_exp_index(index3) = jj;
                    end
                elseif smair_exp_norm(jj)>1
                    if P4(jj)<1.1
                        index3 = index3 + 1;
                        [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),1,1.1);
                        smair_exp_alt(index3,:) = [1 smairz];
                        P4_alt(index3,:) = [1.1 pz];
                        alt_exp_index(index3) = jj;
                    elseif P4(jj)>2.7
                        index3 = index3 + 1;
                        [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),1,2.7);
                        smair_exp_alt(index3,:) = [1 smairz];
                        P4_alt(index3,:) = [2.7 pz];
                        alt_exp_index(index3) = jj;
                    else
                        index3 = index3 + 1;
```

```
                        [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),1,P4(jj));
                        smair_exp_alt(index3,:) = [1 smairz];
                        P4_alt(index3,:) = [P4(jj) pz];
                        alt_exp_index(index3) = jj;
                    end
                elseif P4(jj)<1.1
                    index3 = index3 + 1;
                    [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),smair_exp_norm(jj),1.1);
                    smair_exp_alt(index3,:) = [smair_exp_norm(jj)
smairz];
                    P4_alt(index3,:) = [1.1 pz];
                    alt_exp_index(index3) = jj;
                elseif P4(jj)>2.7
                    index3 = index3 + 1;
                    [smairz,pz] =
linear(smair_exp_norm(jj),P4(jj),smair_exp_norm(jj),2.7);
                    smair_exp_alt(index3,:) = [smair_exp_norm(jj)
smairz];
                    P4_alt(index3,:) = [2.7 pz];
                    alt_exp_index(index3) = jj;
                end
            end
        end
        index_c = 1;
        index_d = 1;
        P4_ir = zeros(1,max(hh));
        for index6 = 1:max(hh)           % this loop gathers the data
points that are in range
            if alt_exp_index(index_c) == index6
                index_c = index_c + 1;
            else
                P4_ir(index_d) = P4(index6);           %ir = in range
                smair_exp_norm_ir(index_d) = smair_exp_norm(index6);
                index_d = index_d + 1;
            end
        end

        % ad_exp = adiabatic efficiency of expander
        % exp_speed = speed of expander (RPM)

        if out_e        % this if statement finds values for those
points outside of the data range
            ad_exp_alt = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4 2.7],[.0625
.125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75 .8125 .875
.9375 1],CC,P4_alt,smair_exp_alt,'cubic');
            exp_speed_alt = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P4_alt,smair_exp_alt,'cubic');
            ad_exp_oor =
linear4(P4_alt,smair_exp_alt,ad_exp_alt,P4,smair_exp_norm,alt_exp_index
);
            exp_speed_oor =
linear4(P4_alt,smair_exp_alt,exp_speed_alt,P4,smair_exp_norm,alt_exp_in
dex);  %oor = out of range
            clear ad_exp_alt exp_speed_alt
```

```
            if max(P4_ir)>0     %this loop finds values for the points
in range using bicubic interpolation
                ad_exp_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P4_ir,smair_exp_norm_ir','cubic');
                exp_speed_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P4_ir,smair_exp_norm_ir','cubic');
                ad_exp_ir = diag(ad_exp_norm)';
                exp_speed_ir = diag(exp_speed_norm)';
                index_c = 1;
                index_d = 1;
                for jj = 1:max(hh)
                    if alt_exp_index(index_c) == jj
                        exp_speed(jj) = exp_speed_oor(index_c) *
speed_max;
                        ad_exp(jj) = ad_exp_oor(index_c) * adeff_max;
                        index_c = index_c + 1;
                    else
                        exp_speed(jj) = exp_speed_ir(index_d) *
speed_max;
                        ad_exp(jj) = ad_exp_ir(index_d) * adeff_max;
                        index_d = index_d + 1;
                    end
                end
            else   % if all data points are out of range, these
equations are used
                exp_speed = exp_speed_oor * speed_max;
                ad_exp = ad_exp_oor * adeff_max;
            end
        else       % if all the data points are in range, these
equations are used
            ad_exp_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P4,smair_exp_norm','cubic');
            exp_speed_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P4,smair_exp_norm','cubic');
            ad_exp = (diag(ad_exp_norm) * adeff_max)';
            exp_speed = (diag(exp_speed_norm)*speed_max)';
        end
        Pwr_e = ((mair4/Ma + mvap4/Mh2o)*8314.*T4*k./(1-
k).*((P1./P4).^((k-1)/k)-1)).*ad_exp;   % expander output power (W)
        Tout = T4 - T4.*ad_exp.*(1-(P1./P4).^((k-1)/k));
% expander output temperature (K)
        Pg_out = 10.^(2.95E-2*(Tout-273)-9.18E-5*(Tout-273).^2+1.44E-
7*(Tout-273).^3-2.18);     % sat. pressure at fuel cell outlet (atm)
        mvap_out = 0.622*mair4*RH4.*Pg_out./(P1-RH4*Pg_out);
                                % mass flow of vapor leaving expander
        mh2o_rec = mvap4 - mvap_out;
                                        % mass flow of liquid water
recoverable from expander outlet
        for jj = max(hh)
            if Pwr_e(jj) <0
                Pwr_e(jj) = 0;
            end
            if mh2o_rec(jj) < 0
```

```
                mh2o_rec(jj) = 0;
            end
        end

%-------------------------------------------------------------------------
% System Output and Plots
%-------------------------------------------------------------------------

        Pwr_netwexp = Pwr_e + Pwr_f - avePwr_fan - Pwr_c - Pwr_pump;
% system power with an expander (W)
        n_syswexp = Pwr_netwexp/(mH2*H2_LHV);
% system efficiency with an expander
        Pwr_net = Pwr_f - avePwr_fan - Pwr_c - Pwr_pump;
% system power without an expander (W)
        n_sys = Pwr_net/(mH2*H2_LHV);
% system efficiency without an expander
        mh2o_bal = mh2o_net + mh2o_rec - mh2o_addh - mvap_H2;
% system water balance (kg/s)

        prtest(hh,index2) = n_sys';             % adiabatic efficiency
without expander
        prtest_exp(hh,index2) = n_syswexp';     % adiabatic efficiency
with expander
        comp_eff(hh,index2) = eff_overall';     % compressor efficiency
        press_log(hh,index2) = P2';             % compressor output
pressure (atm)
        press_drop(hh,index2) = Pdrop';         % fuel cell pressure
drop (atm)
        flo_log(hh,index2) = smair';            % mass flow rate
through compressor (kg/s)
        enuf_press(hh,index2) = press_check';   % checks to see if
enough pressure is supplied to overcome pressure drop
        esmair(hh,index2) = smair_exp';         % standard mass flow
rate exiting the expander (slps)
        fc_outtemp(hh,index2) = T4';            % outlet temperature of
the fuel cell (K)
        fc_outpress(hh,index2) = P4';           % outlet pressure of
the fuel cell (atm)
        e_eff(hh,index2) = ad_exp';             % expander efficiency
        c_temp(hh,index2) = temp_rise';         % temperature rise from
inlet to outlet of compressor (K)
        c_speed(hh,index2) = comp_speed';       % compressor speed
(RPM)
        v_eff(hh,index2) = vol_eff';            % compressor volumetric
efficiency
        POW_e(hh,index2) = Pwr_e';              % expander power (W)
    end   %for loop (changes index2--flow rate)
    presstest(:,:,index) = prtest';             % adiabatic efficiency
without expander
    presstest_exp(:,:,index) = prtest_exp';     % adiabatic efficiency
with expander
    c_eff(:,:,index) = comp_eff';               % compressor efficiency
    pressure_log(:,:,index) = press_log';       % compressor output
pressure (atm)
    pressr_drop(:,:,index) = press_drop';       % fuel cell pressure
drop (atm)
```

```
    flow_log(:,:,index) = flo_log';             % mass flow rate
through compressor (kg/s)
    enough_press(:,:,index) = enuf_press';      % checks to see if
enough pressure is supplied to overcome pressure drop
    expander_smair(:,:,index) = esmair';        % standard mass flow
rate exiting the expander (slps)
    temp4(:,:,index) = fc_outtemp';             % outlet temperature of
the fuel cell (K)
    press4(:,:,index) = fc_outpress';           % outlet pressure of
the fuel cell (atm)
    expander_eff(:,:,index) = e_eff';           % expander efficiency
    c_temprise(:,:,index) = c_temp';            % temperature rise from
inlet to outlet of compressor (K)
    compressor_speed(:,:,index) = c_speed';     % compressor speed
(RPM)
    volumetric_eff(:,:,index) = v_eff';         % compressor volumetric
efficiency
    power_exp(:,:,index) = POW_e;               % expander power (W)
end %current

% these two for loops find the highest efficiency of the compressor
with and without the expander, as well as the mass flow rate
% (exiting compressor) and the pressure (entering fuel cell) at the
highest efficiency
for x = 1:index
    [Y,I] = max(max(presstest_exp(:,:,x).*enough_press(:,:,x)));
    [store_eff(x),i] = max(presstest_exp(:,I,x).*enough_press(:,I,x));
    store_flow(x) = flow_log(i,I,x);
    store_press(x) = max(pressure_log(i,I,x));
end

for x = 1:index
    [Y,I] = max(max(presstest(:,:,x).*enough_press(:,:,x)));
    [store_eff_exp(x),i] = max(presstest(:,I,x).*enough_press(:,I,x));
    store_flow_exp(x) = flow_log(i,I,x);
    store_press_exp(x) = max(pressure_log(i,I,x));
end
```

This program simulates the fuel cell system with the twin-screw compressor

PROGRAM 2

```
close all
clear all

% This program simulates a fuel cell system using a curve fit to the
highest efficiency pressure line found in another program
```

```
% Assumptions/conditions:  constant specific heats, variable stack
temperature but fixed stack humidity, fixed fan
% and compressor motor efficiency, ideal gases, fixed relative humidity
and temperature at entrance to compressor

% Variable suffixes
% 1 -- compressor air inlet
% 2 -- humidifier air inlet
% 3 -- fuel cell air inlet
% 4 -- expander air inlet
% 5 -- radiator coolant inlet
% 6 -- fuel cell coolant inlet
% 7 -- humidifier coolant inlet

%-------------------------------------------------------------------------
% Constants and System Parameters
%-------------------------------------------------------------------------

% physical constants
cp_air = 1004.5;                % specific heat of air (J/kg-K)
cp_h2o = 4200;                  % specific heat of water (J/kg-K)
cp_vap = 1900;                  % specific heat of vapor (J/kg-K)
cp_H2 = 14400;                  % specific heat of hydrogen (J/kg-K)
e_vap = 2383000;                % approx. energy of vaporization of
water at 50°C (J/kg) (Shapiro 724)
H2_HHV = 141890000;             % higher heating value of hydrogen
(J/kg)
H2_LHV = 119860000;             % lower heating value of hydrogen
(J/kg)
k = 1.4;                        % specific heat ratio of air
Ma = 28.97;                     % molar mass of air (kmol/kg)
MH2 = 2.016;                    % molar mass of hydrogen (kmol/kg)
Mh2o = 18.02;                   % molar mass of water (kmol/kg)
MO2 = 32;                       % molar mass of oxygen (kmol/kg)

% 1050 compressor values
adeff_max = 0.675;              % maximum expected adiabatic efficiency
voleff_max = 0.975;             % maximum expected volumetric
efficiency
temp_max = 96.15;               % maximum expected outlet temperature
increase of compressor (K)
speed_max = 16;                 % maximum speed of compressor
(RPM/1000)
BB = [.115 .121 .135 .149 .157 .162 .167 .169;.172 .187 .204 .216 0.224
.232 .242 .254;.229 .254 .271 .281 .293 .3 .310 .319;.288 .314 .334
.346 .359 .359 .367 .377;.348 .372 .396 .41 .419 .416 .425 .436;.409
.432 .459 .471 .481 .478 .487 .497;.477 .494 .523 .533 .544 .539 .548
.558;.543 .556 .584 .595 .607 .599 .608 .618;.596 .618 .642 .654 .667
.658 .668 .679;.649 .681 .701 .715 .725 .721 .730 .742;.706 .741 .762
.777 .785 .783 .792 .806;.764 .803 .823 .839 .847 .843 .849 .872;.831
.872 .883 .899 .908 .902 .905 .939;.911 .942 .951 .958 .967 .971 .972
1.007;.989 1.012 1.019 1.016 1.026 1.051 1.049 1.076;1.067 1.081 1.087
1.074 1.085 1.131 1.125 1.144]; %speed matrix for Opcon 1050 -- last
modified 2/16/01
CC = [.535 .715 .711 .566 .424 .321 .243 .161;.546 .769 .809 .741 .663
.618 .6 .571;.558 .824 .894 .859 .816 .779 .751 .703;.555 .849 .944
.913 .877 .839 .807 .758;.526 .867 .968 .95 .920 .881 .851 .801;.497
```

```
.87 .983 .971 .954 .909 .883 .836;.472 .860 .993 .986 .963 .936 .909
.863;.447 .844 .993 .996 .963 .945 .918 .876;.425 .814 .988 .997 .963
.954 .927 .887;.403 .78 .975 .986 .963 .946 .925 .887;.382 .729 .957
.971 .963 .936 .923 .886;.361 .679 .926 .957 .955 .926 .919 .871;.337
.644 .885 .937 .933 .917 .916 .857;.306 .608 .853 .909 .905 .904 .906
.858;.283 .569 .821 .881 .877 .886 .891 .859;.262 .531 .789 .854 .849
.868 .875 .86]; %adiabatic efficiency matrix for Opcon 1050 -- last
modified 2/17/01
DD = [.688 .617 .526 .47 .437 .416 .397 .386;.756 .682 .617 .58 .558
.541 .521 .498;.823 .747 .698 .672 .647 .632 .612 .592;.877 .798 .756
.73 .705 .69 .672 .653;.905 .846 .8 .772 .754 .738 .721 .701;.931 .878
.828 .806 .791 .771 .756 .74;.944 .896 .849 .832 .815 .802 .788
.773;.957 .911 .869 .852 0.835 .824 0.81 .797;.968 .921 .888 .871 .855
.845 .833 .819;.978 .931 .905 .887 .874 .858 .846 .832;.987 .94 .915
.897 .888 .869 .86 .845;.995 .948 .925 .907 .898 .881 .874 .851;.993
.945 .933 .917 .908 .892 .889 .857;.975 .942 .933 .926 .918 .894 .893
.86;.962 .94 .933 .936 .927 .884 .887 .863;.951 .938 .933 .946 .937
.875 .88 .867]; %volumetric efficiency matrix for Opcon 1050 -- last
modified 2/17/01

% approx. 60 kW stack (honeywell)
num_cell = 210;                 % number of cells per stack
area_cell = 678;                % cell effective area (cm^2)
maxfan = 250;                   % max power of fan (W)
maxpump = 900;                  % max power of pump (W) -- determined
from Price Pump CD-100, 3600 RPM rating, 4.25" impeller diameter
maxpumpflow = 3.15;             % max mass flow of pump (kg/s) --
determined from Price Pump (see above)
resist = 3.68;                  % flow resistance for fuel cell system
to achieve 0.4 atm drop at max flow (atm-s/kg)
radair_on = 3.63;               % mass flow of air through radiator
with fan on (m^3/s)
radair_off = 0.34;              % mass flow of air through radiator
with fan off (m^3/s)
UA_on = 5100;                   % radiator UA value with fan on
UA_off = 510;                   % radiator UA value with fan off

% input values
Amax = 475;                     % maximum current for fuel cell (A)
Amin = 5;                       % minimum current for fuel cell (A)
Astep = 5;                      % step size for current loop (A)
T1 = 298;                       % temperature entering compressor (K)
T3 = 353;                       % desired temperature of input to stack
(K)
T3_init = T3;                   % record of original T3 value (T3
changes in program)
T4 = T3;                        % output temperature of stack
T4_init = T4;                   % stores original T4 value
T_h2o = T4+5;                   % assumed temperature of water taken
for humidification (stack coolant outlet)
T_h2o_init = T_h2o;             % stores original T_h2o value
P1 = 100000/101325;             % pressure entering compressor (atm)
RH1 = 0.5;                      % relative humidity entering compressor
RH3 = 0.75;                     % relative humidity entering fuel cell
-- use whole percentage points
RH3_init = RH3;                 % record of original RH3 value
RH_H2 = 1;                      % hydrogen humidity entering fuel cell
```

```
SR_min = 2;                     % minimum stoichiometric ratio
SR_max = 2.5;                   % maximum stoichiometric ratio
nm = 0.88;                      % combined compressor motor and
inverter efficiency
nfan = 0.65;                    % fan efficiency
expnd = 0;                      % 1 if an expander is desired, 0 if not
hi_p = 0;                       % 2 if high pressure line, 1 if mid
pressure line, 0 if normal pressure line
                                % no mid pressure line without expander
ploton = 0;                     % 1 if plots are desired, 0 if no plots
are desired

% minimum parasitics
minpumpflow = 0.2;              % minimum coolant flow (kg/s)
min_smair = 0;                  % minimum standard mass flow of air
(SLPS)

%-------------------------------------------------------------------------
% Initial Calculations
%-------------------------------------------------------------------------

den_air = Ma*P1*101325/(8314*T1);       % approx. density of dry air
entering compressor (kg/m^3)
den_vap = Mh2o*P1*101325/(8314*T1);     % ideal gas density of water
vapor at input conditions (kg/m^3)
Pg1 = 10^(2.95E-2*(T1-273)-9.18E-5*(T1-273)^2+1.44E-7*(T1-273)^3-2.18);
% inlet saturation pressure (T in K and P_sat in atm)

% this section determines the maximum airflow rate for normalization
purposes
mH2_max = num_cell*475*MH2/(96487*2*1000);                  % kg H2/s
(Fuel Cell Handbook, 8-2)
mO2_max = 0.5*(mH2_max*MO2)/MH2;                            % kg O2/s
(Shapiro, 558, equ. 12.1)
maxair_req = (mO2_max*Ma)/(MO2*0.21);                       % kg dry
air/s required for reaction (ibid, mole fraction of O2 from Shapiro
559)
mair1_max = SR_max*maxair_req;                              % actual kg
dry air/s flowing through compressor
mvap1_max = 0.622*mair1_max*Pg1*RH1/(P1-Pg1*RH1);           % mass flow
of h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
max_smair =(mair1_max/den_air + mvap1_max/den_vap)*1000;    % mass flow
of dry air and vapor(standard l/s)
if max_smair<min_smair                                      % makes
sure maximum is not less than minimum
    max_smair = min_smair;
end

% fuel cell current loop
index = 0;                                                  % place-
keeping variable
for current = Amin:Astep:Amax
    index = index + 1;

% highest efficiency pressure and mass flow lines
%-----------------------------------------------
```

```
    % P2_stack = [0.9949 1.0005 1.0042 1.0091 1.012 1.0161 1.0192
1.0225 1.0258 1.0292 1.0327 1.0354 1.0391 1.0419 1.0449 1.0489 1.052
1.0562 1.0595 1.0639 1.0675 1.0711 1.0748 1.0786 1.0825 1.0865 1.0906
1.0938 1.0981 1.2955 1.304 1.3105 1.3154 1.3194 1.3236 1.3268 1.3303
1.3328 1.3366 1.3394 1.3425 1.3456 1.349 1.3515 1.3542 1.3571 1.3601
1.3623 1.3647 1.3663 1.3681 1.3691 1.3694 1.3708 1.3714 1.3723 1.3744
1.3768 1.3793 1.3832 1.3863 1.3906 1.3852 1.3861 1.3893 1.3928 1.3976
1.4046 1.413 1.4227 1.4348 1.4462 1.4579 1.468 1.4764 1.4843 1.4885
1.4921 1.4911 1.4875 1.4824 1.4757 1.4704 1.4696 1.4713 1.4755 1.4821
1.4883 1.4949 1.5031 1.5109 1.5202 1.5291 1.5376 1.5457];    %without
expander
    % P2_stack = [0.9949 1.0005 1.0042 1.0091 1.012 1.0161 1.0192
1.0225 1.0258 1.4072 1.4177 1.4284 1.4391 1.4509 1.4619 1.4739 1.482
1.4942 1.5065 1.5179 1.5285 1.5391 1.5488 1.5576 1.5665 1.5735 1.5796
1.5858 1.5921 1.5985 1.604 1.6115 1.6184 1.6274 1.6356 1.6448 1.6543
1.6638 1.6726 1.6814 1.6895 1.6986 1.707 1.6985 1.7092 1.7201 1.7311
1.7413 1.7517 1.7603 1.7681 1.7741 1.7804 1.7848 1.7884 1.7923 1.7954
1.7998 1.8043 1.8082 1.8133 1.8196 1.8262 1.8341 1.8423 1.8518 1.8636
1.8756 1.889 1.9047 1.9198 1.9322 1.9419 1.949 1.9534 1.9573 1.9585
1.9581 1.9571 1.9535 1.9604 1.9557 1.9514 1.9486 1.9463 1.9435 1.9401
1.9333 1.9239 1.9111 1.8969 1.8822 1.8681 2.1566 2.1717];     %with
expander
    % smair_stack = [1.4549 2.5079 3.1594 4.0115 4.5644 5.2173 5.7701
6.323 6.8759 7.4288 7.9816 8.4345 9.0874 9.5403 9.9931 10.646 11.1989
11.8518 12.4046 13.1575 13.7104 14.3633 15.0161 15.669 16.3219 16.9748
17.6277 18.2805 18.9334 19.5863 20.2392 20.892 21.567 22.2501 22.9368
23.6275 24.3221 25.0208 25.724 26.4316 27.1438 27.8609 28.583 29.3103
30.0429 30.7809 31.5247 32.2742 33.0297 33.7914 34.5594 35.3339 36.1151
36.9031 37.698 38.5001 39.3095 40.1264 40.9509 41.7832 42.6235 43.472
44.3287 45.194 46.0678 46.9505 47.8421 48.7429 49.6529 50.5725 51.5016
52.4406 53.3895 54.3485 55.3179 56.2977 57.2881 58.2893 59.3014 60.3247
61.3593 62.4053 63.4629 64.5323 65.6137 66.7072 67.8129 68.9311 70.0619
71.2055 72.362 73.5317 74.7146 75.9109 77.1209]; %without expander
    % smair_stack = [1.4549 2.5079 3.1594 4.0115 4.5644 5.2173 5.7701
6.323 6.8759 6.5288 7.1816 7.8345 8.4874 9.1403 9.7931 10.546 11.6989
12.4518 13.1046 13.7575 14.3104 14.8633 15.4161 15.869 16.3219 16.9748
17.6277 18.2805 18.9334 19.5863 20.2392 20.892 21.567 22.2501 22.9368
23.6275 24.3221 25.0208 25.724 26.4316 27.1438 27.8609 28.583 29.3103
30.0429 30.7809 31.5247 32.2742 33.0297 33.7914 34.5594 35.3339 36.1151
36.9031 37.698 38.5001 39.3095 40.1264 40.9509 41.7832 42.6235 43.472
44.3287 45.194 46.0678 46.9505 47.8421 48.7429 49.6529 50.5725 51.5016
52.4406 53.3895 54.3485 55.3179 56.2977 57.2881 58.2893 59.3014 60.3247
61.3593 62.4053 63.4629 64.5323 65.6137 66.7072 67.8129 68.9311 70.0619
71.2055 72.362 73.5317 74.7146 75.9109 77.1209]; %with expander
    % smair = smair_stack(current/5);                        % standard
mass flow rate of air (slps)
    % P2 = P2_stack(current/5);                              %
compressor pressure (atm)
    % hum_ratio = 0.622*Pg1*RH1/(P1-Pg1*RH1);                % ambient
humidity ratio
    % mair1 =smair/(1000/den_air + 1000*hum_ratio/den_vap);  % mass
flow rate of air (kg/s)
    % mvap1 = (smair/1000 - mair1/den_air)*den_vap;          % mass
flow rate of vapor (kg/s)
    % mH2 = num_cell*current*MH2/(96487*2*1000);             % kg H2/s
(Fuel Cell Handbook, 8-2)
%-----------------------------------------------
```

```
% curve fit of highest efficiency pressure lines
%-----------------------------------------------
    SR = 3.381e-6*(current^2)-6e-4*current+2.009;           % operating
line for air stoichiometric ratio--60 kW
                                                            % equation
taken from a curve fit for the following:
                                                            % current:
300 210 120 30
                                                            % SR:
2.5 2.2 2.0 2.0
    if SR<SR_min                                            % makes
sure stoichiometric ratio stays above two and below the maximum
        SR = SR_min;
    elseif SR>SR_max
        SR = SR_max
    end

    % fuel cell fuel and air consumption values
    mH2 = num_cell*current*MH2/(96487*2*1000);              % kg H2/s
(Fuel Cell Handbook, 8-2)
    mO2 = 0.5*(mH2*MO2)/MH2;                                % kg O2/s
(Shapiro, 558, equ. 12.1)
    mair_req = (mO2*Ma)/(MO2*0.21);                         % kg dry
air/s required for reaction (ibid, mole fraction of O2 from Shapiro
559)
    mair1 = SR*mair_req;                                    % actual kg
dry air/s flowing through compressor
    mvap1 = 0.622*mair1*Pg1*RH1/(P1-Pg1*RH1);               % mass flow
of h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
    smair =(mair1/den_air + mvap1/den_vap)*1000;            % mass flow
of dry air and vapor(standard l/s)
    if smair<min_smair                                      % makes
sure standard mass flow does not go below minimum
        smairold = smair;
        smair = min_smair;
        mair1 = smair/smairold*mair1;                       % converts
from old mair1 and mvap1 values to new values
        mvap1 = smair/smairold*mvap1;
    end

    % choosing a pressure line for compressor
    if expnd
        if hi_p==2          % elevated pressure line (high) with
expander (atm)
            P2 = -2.0517*(current/1000)^2 + 3.5*(current/1000) + 1.5;
        elseif hi_p==1      % elevated pressure line (mid) with
expander (atm)
            P2 = -2.1712*(current/1000)^3 + 6.3328*(current/1000)^2 -
1.8011*(current/1000) + 1.9;
        else                % pressure line with expander (atm)
            P2 = -162.68*(current/1000)^4 + 160.48*(current/1000)^3 -
55.517*(current/1000)^2 + 9.3188*(current/1000) + 1.0066;
        end
    elseif hi_p==2          % elevated pressure line without expander
(atm)
        P2 = -2.0517*(current/1000)^2 + 3.5*(current/1000) + 1.5;
    else                    % pressure line without expander (atm)
```

```
        P2 = 9.5323*(current/1000)^3 - 8.3697*(current/1000)^2 +
2.9814*(current/1000) + 0.9981;
    end

    % initializing variables
    T3 = T3_init;
    T4 = T4_init;
    T_h2o = T_h2o_init;

    p_continue = 0;
    while ~p_continue       % this loop checks to make sure the
pressure of the compressor at least matches the pressure drop across
the system

%-----------------------------------------------------------------------
% Compressor Section
%-----------------------------------------------------------------------

        smair_norm = smair/max_smair;           % normalized air flow
through compressor
        out = 0;                                % initialization of
flag
        P2_alt = [0 0];                         % initialization of
pressure outside of data range
        smair_alt = [0 0];                      % initialization of air
flow outside of data range
        if smair_norm<.0625 | smair_norm>1 | P2<1.1 | P2>2.7  % test if
data point is out of data range
            out = 1;                            % this flag shows that
there is a data point in this series that is out of range
            if smair_norm<.0625                 % this loop and nested
loops determine where the data point is in relation to the known data
set
                if P2<1.1
                    [smairz,pz]=linear(smair_norm,P2,0.0625,1.1);   %
here, a point is assigned (smairz) which is linear with the outside
                    smair_alt = [0.0625 smairz];                    %
data point and a point on the edge of the known data.  This is used
                    P2_alt = [1.1 pz];                              %
later to find extrapolated values for the outside data point
                elseif P2>2.7
                    [smairz,pz] = linear(smair_norm,P2,0.0625,2.7);
                    smair_alt = [0.0625 smairz];
                    P2_alt = [2.7 pz];
                else
                    [smairz,pz] = linear(smair_norm,P2,0.0625,P2);
                    smair_alt = [0.0625 smairz];
                    P2_alt = [P2 pz];
                end
            elseif smair_norm>1
                if P2<1.1
                    [smairz,pz] = linear(smair_norm,P2,1,1.1);
                    smair_alt = [1 smairz];
                    P2_alt = [1.1 pz];
                elseif P2>2.7
                    [smairz,pz] = linear(smair_norm,P2,1,2.7);
                    smair_alt = [1 smairz];
```

```
                P2_alt = [2.7 pz];
            else
                [smairz,pz] = linear(smair_norm,P2,1,P2);
                smair_alt = [1 smairz];
                P2_alt= [P2 pz];
            end
        elseif P2<1.1
            [smairz,pz] = linear(smair_norm,P2,smair_norm,1.1);
            smair_alt = [smair_norm smairz];
            P2_alt = [1.1 pz];
        elseif P2>2.7
            [smairz,pz] = linear(smair_norm,P2,smair_norm,2.7);
            smair_alt = [smair_norm smairz];
            P2_alt = [2.7 pz];
        end
    end

    % comp_speed = compressor speed (RPM)
    % eff_overall = adiabatic efficiency of compressor
    % temp_rise = temperature rise in compressor above ambient (K)
    % vol_eff = volumetric efficiency of compressor

    if out         % this if statement finds values for those
points outside of the data range
        % these first seven equations find the compressor values
for the two points inside of the data range which are
        % linear with the data point outside of the data range.  A
2D lookup table is used.
        comp_speed_alt(1) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P2_alt(1),smair_alt(1),'cubic');  % compressor
speed values
        comp_speed_alt(2) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P2_alt(2),smair_alt(2),'cubic');  % compressor
speed values
        eff_overall_alt(1) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P2_alt(1),smair_alt(1),'cubic'); % compressor
adiabatic efficiency values
        eff_overall_alt(2) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P2_alt(2),smair_alt(2),'cubic'); % compressor
adiabatic efficiency values
        temp_rise_alt = interp1([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.084 .237 .4375 .56 .6686 .77 .866 1],P2_alt,'cubic');
%compressor temperature rise values
        vol_eff_alt(1) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],DD,P2_alt(1),smair_alt(1),'cubic');     %
compressor volumetric efficiency values
        vol_eff_alt(2) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],DD,P2_alt(2),smair_alt(2),'cubic');     %
compressor volumetric efficiency values
        % these next four equations extrapolate the compressor
values for the data point outside of the data range from
```

```
            % the other two that are within the data range using linear
extrapolation.
            comp_speed_oor =
linear22(P2_alt,smair_alt,comp_speed_alt,P2,smair_norm);  %oor = out of
range
            eff_overall_oor =
linear22(P2_alt,smair_alt,eff_overall_alt,P2,smair_norm);
            temp_rise_oor = linear32(P2_alt,temp_rise_alt,P2);
            vol_eff_oor =
linear22(P2_alt,smair_alt,vol_eff_alt,P2,smair_norm);
            % these equations are used to remove the normalization
            comp_speed = comp_speed_oor * speed_max;
            eff_overall = eff_overall_oor * adeff_max;
            temp_rise = temp_rise_oor * temp_max;
            vol_eff = vol_eff_oor * voleff_max;
        else    % if all data points are in range, these equations are
used
            comp_speed_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P2,smair_norm,'cubic');
            eff_overall_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],CC,P2,smair_norm,'cubic');
            temp_rise_norm = INTERP1([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.084 .237 .4375 .56 .6686 .77 .866 1],P2,'cubic');
            vol_eff_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],DD,P2,smair_norm,'cubic');
            comp_speed = comp_speed_norm*speed_max;
            eff_overall = eff_overall_norm*adeff_max;
            temp_rise = temp_rise_norm*temp_max;
            vol_eff = vol_eff_norm*voleff_max;
        end
        T2 = T1+temp_rise;                          % Compressor outlet
temperature (K)
        Pwr_c = ((mair1/Ma + mvap1/Mh2o)*8314*T1*k/(k-1)*((P2/P1).^((k-
1)/k)-1))./(eff_overall*nm);      % Compressor power (W)

        % approximate fuel cell calculations to make guess at Q_stack
(stack heat rejection) and m_cool (coolant flow rate)
        pO2 = P2*0.21;                              % partial pressure
of oxygen in dry air (atm)
        J = current/area_cell;                      % current density
(V/cm^2)
        V_guess = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4)*J+0.14*log10(pO2)+(T4-333)*0.00041*log10(10^3*J);   % revised
Honeywell curve with low temp. effects (5/2/01)
        Pwr_f_guess = V_guess*num_cell*current;     % stack power (W)
        Q_guess = mH2*H2_HHV-Pwr_f_guess;           % stack heat
rejection (W)

        % use of a 50gpm pump with max power of 1.2 bHP (CD-100 from
Pricepump)
        m_cool = Q_guess/(10*cp_h2o);               % coolant flow
giving a 10°C temp diff across stack (kg/s)
        Pwr_pump = maxpump/maxpumpflow*m_cool;      % linear power with
max determined by Price Pump website (W)
```

```
        if Pwr_pump>maxpump                         % checks to see if
the pump power exceeds the maximum
            Pwr_pump = maxpump;                     % if the power does
exceed the max, it is set to the maximum power (W)
            m_cool = maxpumpflow;                   % also the coolant
flow for the pump is set to the maximum flow (kg/s)
            'stack will overheat -- reached max of coolant pump'
        elseif m_cool<minpumpflow                      % checks if the
pump coolant flow is below the minimum
            m_cool = minpumpflow;                   % if the flow is
less than the minimum, the flow is set to the minimum (kg/s)
            Pwr_pump = maxpump/maxpumpflow*m_cool;     % linear power
with max determined by Price Pump website (W)
            T3 = T3_init + Q_guess/(2*m_cool*cp_h2o) - 5;   %
temperature achievable in humidifier air stream (assumes 5° min.
difference btwn coolant and air)
            T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);    % coolant
temperature (K)
        end

%-------------------------------------------------------------------------
% Humidifier/Heat Exchanger Section
%-------------------------------------------------------------------------

        P3 = P2;                % no pressure drop over humidifier
        cont = 0;
        while ~cont             % this loop checks to make sure the
humidifier doesn't use more heat than is given by stack
            RH3 = RH3_init;     % initializing RH3

            % hydrogen humidification
            Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-273)^2+1.44E-7*(T4-
273)^3-2.18); % water sat. pressure for H2 (atm)
            mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);    %
water vapor in H2 (kg/s) (assumes initially dry H2)

            % air humidification
            Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-273)^2+1.44E-7*(T3-
273)^3-2.18);    % saturation pressure in fuel cell inlet (T in K and
P_sat in atm)
            if T3==T4
                mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);       % vapor
mass flow going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                mh2o_addh = mvap3 - mvap1;                      % net
water added in humidifier (kg/s)
                mtot3 = mair1 + mvap3;                          % total
mass flow into fuel cell (kg/s)
            else  % effects of coolant loop minimum
                mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);       % vapor
mass needed to humidify air to correct RH when warmed to fuel cell temp
(kg/s) (Shapiro, 582, equ 12.43)
                mh2o_addh = mvap3 - mvap1;                      % net
water added in humidifier (kg/s)
                mtot3 = mair1 + mvap3;                          % total
mass flow into fuel cell (kg/s)
                RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);     %
relative humidity of air at humidifier exit
```

```
                if RH3>1
                    'lower than requested fuel cell inlet air RH'
                    RH3 = 1;
                    mvap3 = 0.622*mair1*Pg3/(P3-Pg3);           % vapor
mass going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                  % net
water added in humidifier (kg/s)
                    mtot3 = mair1 + mvap3;                      % total
mass flow into fuel cell (kg/s)
                    Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);  % saturation pressure entering fuel
cell (atm)
                    RH3_in = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);
% relative humidity of air after warming up in fuel cell
                end
            end
            Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-T2) +
mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o);  % total heat needed to
warm mass flow into fuel cell

            while Q_needed<Q_guess & T4<T4_init                 %
raises the temp of stack if heat is available
                RH3 = RH3_init;                                 %
initializing RH3
                T4 = T4 + .08;                                  %
increases operating temp of fuel cell (K)
                if T4>T4_init                                   % makes
sure T4 doesn't go over the desired operating temp
                    T4 = T4_init;                               % sets
T4 to the initial desired fuel cell temperature (K)
                end
                T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;            %
temperature achievable in humidifier air stream (assumes 5° min.
difference btwn coolant and air)
                T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);             %
water temp same as stack coolant exit temp
                Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-273)^2+1.44E-
7*(T3-273)^3-2.18);    % saturation pressure in fuel cell inlet (atm)
                Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-273)^2+1.44E-
7*(T4-273)^3-2.18);    % saturation pressure in fuel cell inlet (atm)
                mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);%
water vapor in H2 (assumes initially dry H2 and can make it to fc temp)
                if T3==T4
                    mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);       %
vapor mass going into fuel cell (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                      %
net water added in humidifier (kg/s)
                    mtot3 = mair1 + mvap3;                          %
total mass flow into fuel cell (kg/s)
                else    % effects of coolant loop minimum
                    mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);       %
vapor mass needed to humidify air to correct RH when warmed to fuel
cell temp (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                      %
net water added in humidifier (kg/s)
                    mtot3 = mair1 + mvap3;                          %
total mass flow into fuel cell (kg/s)
```

```
                    RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);       %
relative humidity of air at humidifier exit
                    if RH3>1
                        'lower than requested fuel cell inlet air RH'
                        RH3 = 1;
                        mvap3 = 0.622*mair1*Pg3/(P3-Pg3);           %
vapor mass going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;                  %
net water added in humidifier (kg/s)
                        mtot3 = mair1 + mvap3;                      %
total mass flow into fuel cell (kg/s)
                        Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18); % saturation pressure in fuel cell
inlet (atm)
                        RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  % relative humidity of air
after warming up in fuel cell
                    end
                end
                Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-T2)
+ mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o); % total heat needed in
humidifier (W)
            end
            temp_mod = 0;                   % flag to show temperature
modification
            while Q_needed>Q_guess          % lowers stack temperature
to allow humidifier to heat to temperature
                temp_mod = 1;
                RH3 = RH3_init;             % initializing RH3
                T4 = T4 - .08;              % incremental decrease in
stack ave temperature
                T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;        %
temperature achievable in humidifier air stream (K) (assumes 5° min.
difference btwn coolant and air)
                T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);         % water
temp same as stack coolant exit temp (K)
                Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-273)^2+1.44E-
7*(T3-273)^3-2.18); % saturation pressure in fuel cell inlet (atm)
                Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-273)^2+1.44E-
7*(T4-273)^3-2.18); % saturation pressure in fuel cell inlet (atm)
                mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);
% water vapor in H2 (kg/s) (assumes initially dry H2 and can make it to
fc temp)
                if T3==T4
                    mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);   % vapor
mass going into fuel cell (kg/s) (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                  % net
water added in humidifier (kg/s)
                    mtot3 = mair1 + mvap3;                      % total
mass flow into fuel cell (kg/s)
                else    % effects of coolant loop minimum
                    mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);   % vapor
mass needed to humidify air to correct RH when warmed to fuel cell temp
(kg/s) (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                  % net
water added in humidifier (kg/s)
```

```
                    mtot3 = mair1 + mvap3;                      % total
mass flow into fuel cell (kg/s)
                    RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3); %
relative humidity of air at humidifier exit
                    if RH3>1
                        'lower than requested fuel cell inlet air RH'
                        RH3 = 1;
                        mvap3 = 0.622*mair1*Pg3/(P3-Pg3);       % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;              % net
water added in humidifier (kg/s)
                        mtot3 = mair1 + mvap3;                  % total
mass flow into fuel cell (kg/s)
                        Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);  % saturation pressure of air before
warming up in fuel cell (atm)
                        RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  % relative humidity of air
after warming up in fuel cell
                    end
                end
                Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-T2)
+ mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o); % total heat needed in
humidifier (W)
                % Q_needed shows amount of heat needed to warm air
stream to T3
            end
            Q_hum = Q_needed;

%-----------------------------------------------------------------------
% Fuel Cell Section
%-----------------------------------------------------------------------

            Pdrop = resist*(mair1*den_air + (mtot3-mair1)*den_vap); %
pressure loss across system (atm)
            P4 = P3 - Pdrop;                                % pressure
at exit of fuel cell (atm)
            PO2_3 = (P3 - RH3*Pg3)*0.21;                    % partial
pressure of oxygen at fuel cell inlet (atm)
            mO2_used = (num_cell*current*MO2)/(96487*4000); % mass flow
of oxygen consumed (kg/s)
            mair4 = mair1 - mO2_used;                       % total
mass flow of dry air at fuel cell exit (kg/s)
            O2_ratio4 = (mair1*0.21-mO2_used)/mair4;        % ratio of
oxygen to dry air at fuel cell exit
            mvapmax = 0.622*mair1*Pg4/(P4-Pg4);             % total
vapor at 100% RH at fuel cell outlet (kg/s)
            mvap_fc = mvapmax - mvap3;                      % total
vapor acquired by airstream from fuel cell water production (kg/s)
            mh2o_prod = mH2*Mh2o/MH2;                       % stack
water production (kg/s)
            h2o_used = 0;                                   % flag to
show if all of the water in the fuel cell was evaporated in air
            if mvap_fc>mh2o_prod                            % checks to
see if fuel cell air would accept more water than the fuel cell
produces
                h2o_used = 1;
```

```
                mvap_fc = mh2o_prod;    % if production of water is
less than total that the air can accept for 100% humidity, then air
will only take total production of water
                RH4 = ((mvap3+mvap_fc)*(P3-
Pdrop))/((0.622*mair1+mvap3+mvap_fc)*Pg4);   % relative humidity of air
at fuel cell outlet
            else
                RH4 = 1;
                h2o_used = 0;
            end
            PO2_4 = (P4 - RH4*Pg4)*O2_ratio4;               % partial
pressure of oxygen at fuel cell exit (atm)
            pO2 = (PO2_3 + PO2_4)/2;                        % average
partial pressure of oxygen (atm)
            J = current/area_cell;                          % current
density (V/cm^2)
            V = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4)*J+0.14*log10(pO2)+(T4-333)*0.00041*log10(10^3*J); % revised
Honeywell curve with temp. effects (V) (5/2/01)
            if V <0.5
                'data for', current, 'Amp current level is bad'
            end
            Vstack = V * num_cell;                          % stack
voltage (V)
            Pwr_f = Vstack*current;                         % stack
power (W)
            mvap4 = mvap3 + mvap_fc;                        % total
water leaving fuel cell (kg/s)
            n_stack = Pwr_f/(mH2*H2_LHV);                   % stack
efficiency
            Q_stack = mH2*(1 - mvap_fc/mh2o_prod)*H2_HHV +
mH2*(mvap_fc/mh2o_prod)*H2_LHV - Pwr_f;  % stack heat rejection (W)
            mh2o_net = mh2o_prod - mvap_fc;                 % water
recoverable as liquid at fuel cell outlet (kg/s)
            if ((Q_guess/Q_stack)>1.01|(Q_guess/Q_stack)<.99)   %
compares actual stack heat rejection with guess
                if (Q_guess/Q_stack)<.99                    % this only
allows the initialization of temp if there is more heat available from
stack
                    T4 = T4+.05;
                    if T4 > T4_init
                        T4 = T4_init;
                    end
                    T3 = T4;
                    T_h2o = T4+5;
                end
                Q_guess = Q_stack;                              % old
stack heat rejection becomes new guess
                cont = 0;
                % another adjustment of flow rate and check for
exceeding the flow/power limits
                m_cool = Q_guess/(10*cp_h2o);                   %
coolant flow giving a 10°C temp diff across stack (kg/s)
                Pwr_pump = maxpump/maxpumpflow*m_cool;          %
linear power with max determined by Price Pump website (W)
                if Pwr_pump>maxpump
                    Pwr_pump = maxpump;
```

```
                    m_cool = maxpumpflow;
                elseif m_cool<minpumpflow
                    m_cool = minpumpflow;
                    Pwr_pump = maxpump/maxpumpflow*m_cool;      %
linear power with max determined by Price Pump website (W)
                    T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);     % stack
coolant exit temp (K)
                    T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;    % max
temperature achievable in humidifier air stream (K) (assumes 5° min.
difference btwn coolant and air)
                end
            else
                cont = 1;
            end
        end %while (in humidifier)
        if (P2-Pdrop)<P1            % this loop checks if there is
enough pressure to overcome pressure drop
            p_continue = 0;
            P2 = P1 + Pdrop + .001; % new pressure adds pressure drop
plus a small amount extra to ambient
            PR = P2/P1;
        else
            p_continue = 1;
        end
    end %while -- pressure check (before compressor)

%-------------------------------------------------------------------------
% Radiator Section
%-------------------------------------------------------------------------

    % the NTU method is used for the radiator calculations
    T5 = T4 - Q_hum/(m_cool*cp_h2o);    % temperature of coolant
entering radiator
    if T5 < T1
       'Warning!  Coolant temp. entering radiator is lower than
ambient'
    end
    Q_rad = Q_stack - Q_hum;            % heat that must be rejected by
radiator (W)

    %--air properties with fan on
    m_on = radair_on*den_air;
    C_on = m_on*cp_air;
    C_h2o = m_cool*cp_h2o;
    C1 = [C_on C_h2o];
    Cr_on = min(C1)/max(C1);
    NTU_on = UA_on/min(C1);
    eff_on = 1-exp((1/Cr_on)*NTU_on^.22*(exp(-Cr_on*NTU_on^.78)-1));
    Qmax_on = min(C1)*(T5-T1);
    Q_on = eff_on*Qmax_on;

    %--air properties with fan off
    m_off = radair_off*den_air;
    C_off = m_off*cp_air;
    C_h2o = m_cool*cp_h2o;
    C2 = [C_off C_h2o];
    Cr_off = min(C2)/max(C2);
```

```
    NTU_off = UA_off/min(C2);
    eff_off = 1-exp((1/Cr_off)*NTU_off^.22*(exp(-Cr_off*NTU_off^.78)-
1));
    Qmax_off = min(C2)*(T5-T1);
    Q_off = eff_off*Qmax_off;

    bypass_pcnt = 0;
    fan_timeon = (Q_rad-Q_off)/(Q_on-Q_off);    % percent of operating
time with fan on
    if fan_timeon < 0                           % tests if fan can be
turned off and bypass valve activated
        bypass_pcnt = 1 - Q_rad/Q_off;          % percent of flow
diverted through bypass valve
        fan_timeon = 0;
    elseif fan_timeon > 1                       % limits fan to 100%
time on
        fan_timeon = 1;
    end
    avePwr_fan = (fan_timeon*maxfan)/nfan;      % fan power (W)
    Q_out = Q_on*fan_timeon + Q_off*(1-fan_timeon)*(1-bypass_pcnt);
% heat rejected from radiator
    T6 = T5 - Q_out/(m_cool*cp_h2o);    % energy balance, temp of
coolant entering fuel cell
    T7 = T6 + Q_stack/(m_cool*cp_h2o);  % energy balance, temp of
coolant exiting fuel cell and entering humidifier

%-------------------------------------------------------------------------
% Expander Section
%-------------------------------------------------------------------------

    smair_exp =(mair4/den_air + mvap4/den_vap)*1000;    % mass flow of
dry air and vapor (standard l/s)
    smair_exp_norm = smair_exp/max_smair;               % normalization
of air going through expander
    out_e = 0;                                          %
initialization of flag
    P4_alt = [0 0];                                     %
initialization of pressure outside of data range
    smair_exp_alt = [0 0];                              %
initialization of air flow outside of data range

    if smair_exp_norm<.0625 | smair_exp_norm>1 | P4<1.1 | P4>2.7
        out_e = 1;              % this section is similar to
compressor section; see compressor for details
        if smair_exp_norm<.0625
            if P4<1.1
                [smairz,pz] = linear(smair_exp_norm,P4,0.0625,1.1);
                smair_exp_alt = [.0625 smairz];
                P4_alt = [1.1 pz];
            elseif P4>2.7
                [smairz,pz] = linear(smair_exp_norm,P4,0.0625,2.7);
                smair_exp_alt = [.0625 smairz];
                P4_alt = [2.7 pz];
            else
                [smairz,pz] = linear(smair_exp_norm,P4,0.0625,P4);
                smair_exp_alt = [.0625 smairz];
                P4_alt = [P4 pz];
```

```
            end
        elseif smair_exp_norm>1
            if P4<1.1
                [smairz,pz] = linear(smair_exp_norm,P4,1,1.1);
                smair_exp_alt = [1 smairz];
                P4_alt = [1.1 pz];
            elseif P4>2.7
                [smairz,pz] = linear(smair_exp_norm,P4,1,2.7);
                smair_exp_alt = [1 smairz];
                P4_alt = [2.7 pz];
            else
                [smairz,pz] = linear(smair_exp_norm,P4,1,P4);
                smair_exp_alt = [1 smairz];
                P4_alt = [P4 pz];
            end
        elseif P4<1.1
            [smairz,pz] = linear(smair_exp_norm,P4,smair_exp_norm,1.1);
            smair_exp_alt = [smair_exp_norm smairz];
            P4_alt = [1.1 pz];
        elseif P4>2.7
            [smairz,pz] = linear(smair_exp_norm,P4,smair_exp_norm,2.7);
            smair_exp_alt = [smair_exp_norm smairz];
            P4_alt = [2.7 pz];
        end
    end
    if out_e
        ad_exp_alt(1) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4 2.7],[.0625
.125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75 .8125 .875
.9375 1],CC,P4_alt(1),smair_exp_alt(1),'cubic');
        ad_exp_alt(2) = interp2([1.1 1.3 1.6 1.8 2 2.2 2.4 2.7],[.0625
.125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75 .8125 .875
.9375 1],CC,P4_alt(2),smair_exp_alt(2),'cubic');
        exp_speed_alt(1) = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P4_alt(1),smair_exp_alt(1),'cubic');
        exp_speed_alt(2) = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4
2.7],[.0625 .125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75
.8125 .875 .9375 1],BB,P4_alt(2),smair_exp_alt(2),'cubic');
        ad_exp_oor =
linear22(P4_alt,smair_exp_alt,ad_exp_alt,P4,smair_exp_norm);
        exp_speed_oor =
linear22(P4_alt,smair_exp_alt,exp_speed_alt,P4,smair_exp_norm);  %oor =
out of range
        ad_exp = ad_exp_oor * adeff_max;
        exp_speed = exp_speed_oor * speed_max;
    else
        ad_exp_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4 2.7],[.0625
.125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75 .8125 .875
.9375 1],CC,P4,smair_exp_norm','cubic');
        exp_speed_norm = INTERP2([1.1 1.3 1.6 1.8 2 2.2 2.4 2.7],[.0625
.125 .1875 .25 .3125 .375 .4375 .5 .5625 .625 .6875 .75 .8125 .875
.9375 1],BB,P4,smair_exp_norm','cubic');
        ad_exp = (ad_exp_norm * adeff_max)';
        exp_speed = (exp_speed_norm*speed_max)';
    end
    Pwr_e = ((mair4/Ma + mvap4/Mh2o)*8314.*T4*k./(1-k).*((P1./P4).^((k-
1)/k)-1)).*ad_exp;
```

```
    if Pwr_e <0
        Pwr_e = 0;
    end

    if expnd
        % water recovery with expander
        Tout = T4 - T4.*ad_exp.*(1-(P1./P4).^((k-1)/k));  %Hill and
Peterson -- p 379, eq. 8.12
        Pg_out = 10.^(2.95E-2*(Tout-273)-9.18E-5*(Tout-273).^2+1.44E-
7*(Tout-273).^3-2.18);    %sat. pressure at fuel cell outlet (T in K
and Pg in atm)
        RH_out = P1*mvap4/((0.622*mair4 + mvap4)*Pg_out);   % Moran and
Shapiro -- p 582, eq. 12.43
        if RH_out>1
            mvap_out = 0.622*mair4*Pg_out/(P1 - Pg_out);  %assumes Tout
remains the same
            mh2o_rec = mvap4 - mvap_out;
        else
            mh2o_rec = 0;
        end
        if mh2o_rec < 0
            mh2o_rec = 0;
        end

    else
        % water recovery without expander
        mh2o_rec = 0;
    end
% ---------------------------------------------------------------------
% system output section
% ---------------------------------------------------------------------

    Pwr_netwexp = Pwr_e + Pwr_f - avePwr_fan - Pwr_c - Pwr_pump;  %
system power with an expander
    n_syswexp = Pwr_netwexp/(mH2*H2_LHV);                         %
system efficiency with an expander
    Pwr_net = Pwr_f - avePwr_fan - Pwr_c - Pwr_pump;              %
system power without an expander
    n_sys = Pwr_net/(mH2*H2_LHV);                                 %
system efficiency without an expander
    mh2o_bal = mh2o_net + mh2o_rec - mh2o_addh;                   %
system water balance with expander

    current_plot(index) = current;          % Current range for plots
(A)
    stack_pwr(index) = Pwr_f;               % Fuel cell stack power (W)
    stack_eff(index) = n_stack;             % Fuel cell stack
efficiency
    stack_V(index) = Vstack;                % Fuel cell stack voltage
(V)
    stack_T(index) = T3-273;                % Fuel cell stack
temperature (°C)
    stack_Q(index) = Q_stack;               % Fuel cell stack heat
rejection (W)
    net_power(index) = Pwr_net;             % Net fuel cell system
power (W)
```

```
    net_powerwexp(index) = Pwr_netwexp;     % Net fuel cell system
power with expander (W)
    net_eff(index) = n_sys;                 % Net fuel cell system
adiabatic efficiency
    net_effwexp(index) = n_syswexp;         % Net fuel cell system
adiabatic efficiency with expander
    compressor_power(index) = Pwr_c;        % Compressor power (W)
    compressor_eff(index) = eff_overall;    % Compressor adiabatic
efficiency
    compressor_veff(index) = vol_eff;       % Compressor volumetric
efficiency
    compressor_speed(index) = comp_speed;   % Compressor speed (RPM)
    compressor_Trise(index) = temp_rise;    % Compressor temperature
rise from ambient (°C)
    compressor_pressure(index) = P2;        % Compressor pressure (atm)
    expand_pwr(index) = Pwr_e;              % Expander power extracted
(W)
    expander_speed(index) = exp_speed;      % Expander speed (RPM)
    pressure_drop(index) = Pdrop;           % Pressure drop over stack
(atm)
    H2O_bal(index) = mh2o_bal;              % Water balance (kg/s)
    addedwater_air(index) = mh2o_addh;      % Water added to air (kg/s)
    addedwater_H2(index) = mvap_H2;         % Water added to hydrogen
(kg/s)
    RH4_log(index) = RH4;                   % Relative humidity of air
at fuel cell exit
    mtotal4(index) = (mvap4 + mair4)*1000;  % Total mass flow out of
fuel cell (kg/s)
    smair_in(index) = smair;                % Standard mass flow of
air/water at inlet (SLPS)
    fan_ontime(index)= fan_timeon;          % Percent of fan run time
    parasitics(index) = Pwr_c + Pwr_pump + avePwr_fan;  %Total
parasitic power (W)
    H2O_produced(index) = mh2o_prod;        % Total water produced in
fuel cell (kg/s)
end % current

if expnd         % Tab delimited files
    Z = zeros(length(current_plot),1);
    M= [current_plot' stack_pwr' stack_eff' stack_V' stack_T' stack_Q'
net_powerwexp' net_effwexp' compressor_power' compressor_eff'
compressor_speed' compressor_Trise' compressor_pressure' expand_pwr'
pressure_drop' H2O_bal' addedwater_air' addedwater_H2' RH4_log'
parasitics'];
    dlmwrite('twinout.dat', M,'\t');

else
    Z = zeros(length(current_plot),1);
    M= [current_plot' stack_pwr' stack_eff' stack_V' stack_T' stack_Q'
net_power' net_eff' compressor_power' compressor_eff' compressor_speed'
compressor_Trise' compressor_pressure' pressure_drop' H2O_bal'
addedwater_air' addedwater_H2' RH4_log' parasitics'];
    dlmwrite('twinout.dat', M,'\t');
end

% plotted results
if ploton
```

```
figure(1)
plot(current_plot,stack_pwr/1000)
title('Figure 1: Stack Power')
xlabel('Current (A)')
ylabel('Power (kW)')
grid

figure(2)
plot(current_plot,stack_eff)
title('Figure 2: Stack Efficiency')
xlabel('Current (A)')
ylabel('Efficiency')
grid

figure(3)
plot(current_plot,stack_Q/1000)
title('Figure 3: Stack Heat Rejection')
xlabel('Current (A)')
ylabel('Heat Rejected (kW)')
grid

figure(4)
plot(current_plot,stack_V)
title('Figure 4: Stack Voltage')
xlabel('Current (A)')
ylabel('Voltage (V)')
grid

figure(5)
plot(current_plot,net_powerwexp/1000,current_plot,net_power/1000)
title('Figure 5: Net System Power')
xlabel('Current (A)')
ylabel('Power (kW)')
legend('with expander', 'no expander',0)
grid

figure(6)
plot(current_plot,H2O_bal)
title('Figure 6: Net Water Production')
xlabel('Current (A)')
ylabel('Water (kg/s)')
grid

figure(7)
plot(current_plot,net_effwexp,current_plot,net_eff)
title('Figure 7: System Efficiency')
xlabel('Current (A)')
ylabel('Efficiency')
legend('with expander', 'no expander',0)
grid

figure(8)
plot(current_plot,fan_ontime*100)
title('Figure 8: Percentage of Time that the Fan is On')
xlabel('Current (A)')
ylabel('Percent')
grid
```

```
figure(9)
plot(current_plot, expand_pwr/1000)
title('Figure 9: Expander Power')
xlabel('Current (A)')
ylabel('Power (kW)')
grid

figure(10)
plot(current_plot,compressor_power)
title('Figure 10: Power Required from Compressor')
xlabel('Current (A)')
ylabel('Power (W)')
grid

figure(11)
plot(current_plot, compressor_speed*1000, current_plot,
expander_speed*1000)
title('Figure 11:  Compressor and Expander Speed');
xlabel('Current (A)')
ylabel('speed (rpm)')
legend('compressor','expander',0)
grid

figure(12)
plot(current_plot, compressor_Trise)
title('Figure 12:  Temperature Rise in Compressor')
xlabel('Current (A)')
ylabel('Degrees (°C)')
grid

figure(13)
plot(current_plot,compressor_pressure)
title('Figure 13: Output Pressure of Compressor')
xlabel('Current (A)')
ylabel('Pressure (atm)')
grid

figure(14)
plot(current_plot, compressor_veff, current_plot, compressor_eff)
title('Figure 14:  Volumetric and Adiabatic Efficiency of Compressor');
xlabel('Current (A)')
ylabel('Efficiency')
grid

figure(15)
plot(current_plot,pressure_drop)
title('Figure 15: Pressure Drop Across System')
xlabel('Current (A)')
ylabel('Pressure (atm)')
grid

figure(16)
plot(current_plot,parasitics/1000,current_plot,stack_pwr/1000)
title('Figure 16: Parasitic Power Compared to Fuel Cell Power')
xlabel('Current (A)')
ylabel('Power (kW)')
```

```
legend('Parasitic','Fuel Cell',0)
grid

figure(17)
plot(net_powerwexp/1000, stack_T)
title('Figure 17:  Stack Temperature')
xlabel('Total System Power (kW)')
ylabel('Temperature (°C)')
grid

figure(18)
plot(current_plot, smair_in/.8445, current_plot, mtotal4)
title('Figure 18: Moist Air Mass Flow');
xlabel('Current (A)')
ylabel('mass flow (g/s)')
legend('inlet air','outlet air',0)
grid

figure(19)
plot(current_plot, RH4_log)
title('Figure 19:  Fuel Cell Exit Humidity');
xlabel('Current (A)')
ylabel('Relative Humidity')
grid

figure(20)
plot(current_plot,addedwater_air*1000, current_plot,
addedwater_H2*1000)
title('Figure 20:  Water Added to Air and Hydrogen Streams');
xlabel('Current (A)')
ylabel('water (g/s)')
legend('Air Stream','H2 Stream',0)
grid
end
```

This program finds the highest efficiency operating lines for the turbocompressor

PROGRAM 3

```
close all
clear all

% This program is used to find the compressor pressure/air flow curve
that gives the highest system efficiency for the
% turbocompressor.  Calculations are done using a variety of pressures
and air flow rates, and a single
% combination of these is output for each current value

% Assumptions/conditions:  constant specific heats, variable stack
temperature but fixed stack humidity, fixed fan
```

```
% and compressor motor efficiency, ideal gases, fixed relative humidity
and temperature at entrance to compressor

% Variable suffixes
% 1 -- compressor air inlet
% 2 -- humidifier air inlet
% 3 -- fuel cell air inlet
% 4 -- expander air inlet
% 5 -- radiator coolant inlet
% 6 -- fuel cell coolant inlet
% 7 -- humidifier coolant inlet

%-------------------------------------------------------------------------
% Constants and System Parameters
%-------------------------------------------------------------------------

% physical constants
cp_air = 1004.5;                % specific heat of air (J/kg-K)
cp_h2o = 4200;                  % specific heat of water (J/kg-K)
cp_vap = 1900;                  % specific heat of vapor (J/kg-K)
cp_H2 = 14400;                  % specific heat of hydrogen (J/kg-K)
e_vap = 2383000;                % approx. energy of vaporization of
water at 50°C (J/kg) (Shapiro 724)
H2_HHV = 141890000;             % higher heating value of hydrogen
(J/kg)
H2_LHV = 119860000;             % lower heating value of hydrogen
(J/kg)
k = 1.4;                        % specific heat ratio of air
Ma = 28.97;                     % molar mass of air (kmol/kg)
MH2 = 2.016;                    % molar mass of hydrogen (kmol/kg)
Mh2o = 18.02;                   % molar mass of water (kmol/kg)
MO2 = 32;                       % molar mass of oxygen (kmol/kg)

%turbomachine constants

s105K = [0.791 0.854 0.924 0.975 1.034 1.068 1.072;            %
normalized mass flow (SLPS)
   0.997 1 0.983 0.957 0.915 0.849 0.772;                      %
normalized pressure ratio
   0.876 0.896 0.922 0.935 0.926 0.841 0.8;                    %
normalized efficiency
   1 0 0 0 0 0 0];                                             %
normalized speed

s100K = [0.757 0.812 0.872 0.928 0.979 1.017 1.034 1.04 1.042;  %
normalized mass flow (SLPS)
   0.935 0.929 0.92 0.897 0.863 0.809 0.741 0.678 0.595;       %
normalized pressure ratio
   0.913 0.936 0.957 0.962 0.95 0.902 0.852 0.807 0.699;       %
normalized efficiency
   0.952 0 0 0 0 0 0 0 0];                                     %
normalized speed

s90K = [0.658 0.709 0.757 0.804 0.85 0.897 0.932 0.945 0.954;   %
normalized mass flow (SLPS)
   0.781 0.778 0.766 0.749 0.727 0.695 0.65 0.595 0.53;        %
normalized pressure ratio
```

```
   0.95 0.977 0.984 0.986 0.974 0.945 0.889 0.812 0.705;        %
normalized efficiency
   0.857 0 0 0 0 0 0 0 0];                                      %
normalized speed

s80K = [0.551 0.59 0.632 0.671 0.709 0.753 0.782 0.804 0.82;    %
normalized mass flow (SLPS)
   0.655 0.652 0.641 0.63 0.615 0.593 0.567 0.524 0.473;        %
normalized pressure ratio
   0.97 0.987 0.998 1 0.987 0.956 0.902 0.824 0.719;            %
normalized efficiency
   0.762 0 0 0 0 0 0 0 0];                                      %
normalized speed

s70K = [0.458 0.491 0.525 0.555 0.59  0.62 0.624 0.668 0.688;   %
normalized mass flow (SLPS)
   0.544 0.541 0.536 0.527 0.516 0.501 0.499 0.456 0.425;       %
normalized pressure ratio
   0.981 0.991 0.995 0.993 0.981 0.952 0.945 0.838 0.715;       %
normalized efficiency
   0.667 0 0 0 0 0 0 0 0];                                      %
normalized speed

s60K = [0.39 0.418 0.445 0.475 0.509 0.53 0.534 0.572 0.59;     %
normalized mass flow (SLPS)
   0.467 0.464 0.459 0.453 0.444 0.439 0.436 0.407 0.385;       %
normalized pressure ratio
   0.981 0.994 0.998 0.994 0.981 0.956 0.946 0.841 0.719;       %
normalized efficiency
   0.571 0 0 0 0 0 0 0 0];                                      %
normalized speed

s50K = [0.325 0.35 0.368 0.393 0.424 0.443 0.445 0.475 0.491;   %
normalized mass flow (SLPS)
   0.407 0.405 0.402 0.399 0.396 0.388 0.385 0.37 0.356;        %
normalized pressure ratio
   0.981 0.991 0.996 0.995 0.981 0.956 0.95 0.841 0.719;        %
normalized efficiency
   0.476 0 0 0 0 0 0 0 0];                                      %
normalized speed

s40K = [0.261 0.282 0.299 0.317 0.333 0.35 0.354 0.38 0.397;    %
normalized mass flow (SLPS)
   0.359 0.356 0.356 0.353 0.35 0.348 0.346 0.339 0.325;        %
normalized pressure ratio
   0.981 0.991 0.996 0.995 0.981 0.956 0.946 0.841 0.719;       %
normalized efficiency
   0.381 0 0 0 0 0 0 0 0];                                      %
normalized speed

s30K = [0.195 0.207 0.222 0.236 0.253 0.268 0.272 0.287 0.295;  %
normalized mass flow (SLPS)
   0.325 0.325 0.325 0.325 0.322 0.316 0.315 0.311 0.302;       %
normalized pressure ratio
   0.981 0.991 0.996 0.995 0.981 0.956 0.95 0.841 0.717;        %
normalized efficiency
```

```
   0.286 0 0 0 0 0 0 0 0];                                          %
normalized speed

s20K = [0.131 0.14 0.146 0.154 0.162 0.171 0.174 0.187 0.195;   %
normalized mass flow (SLPS)
   0.299 0.299 0.299 0.299 0.298 0.296 0.296 0.291 0.288;       %
normalized pressure ratio
   0.981 0.991 0.995 0.995 0.981 0.952 0.946 0.838 0.715;       %
normalized efficiency
   0.191 0 0 0 0 0 0 0 0];                                          %
normalized speed

s10K = [0.056 0.061 0.066 0.071 0.076 0.081 0.082 0.097 0.103;  %
normalized mass flow (SLPS)
   0.293 0.293 0.293 0.293 0.293 0.291 0.291 0.291 0.288;       %
normalized pressure ratio
   0.981 0.987 0.995 0.995 0.981 0.956 0.95 0.841 0.715;        %
normalized efficiency
   0.095 0 0 0 0 0 0 0 0];                                          %
normalized speed

% approx. 60 kW stack (honeywell)
num_cell = 210;                 % number of cells per stack
area_cell = 678;                % cell effective area (cm^2)
maxfan = 250;                   % max power of fan (W)
maxpump = 900;                  % max power of pump (W) -- determined
from Price Pump CD-100, 3600 RPM rating
maxpumpflow = 3.15;             % max mass flow of pump (kg/s) --
determined from Price Pump (see above)
resist = 3.68;                  % flow resistance for fuel cell system
to achieve 0.4 atm drop at max flow (atm-s/kg)
radair_on = 3.63;               % mass flow of air through radiator
with fan on (m^3/s)
radair_off = 0.34;              % mass flow of air through radiator
with fan off (m^3/s)
UA_on = 5100;                   % radiator UA value with fan on
UA_off = 510;                   % radiator UA value with fan off

% input values
Amax = 475;                     % maximum current for fuel cell (A)
Amin = 5;                       % minimum current for fuel cell (A)
Astep = 5;                      % step size for current loop (A)
T1 = 298;                       % temperature entering compressor (K)
T3 = 353;                       % desired temperature of input to stack
(K)
T3_init = T3;                   % record of original T3 value (T3
changes in program)
T4 = T3;                        % output temperature of stack
T4_init = T4;                   % stores original T4 value
T_h2o = T4+5;                   % assumed temperature of water taken
for humidification (stack coolant outlet)
T_h2o_init = T_h2o;             % stores original T_h2o value
P1 = 100000/101325;             % pressure entering compressor (atm)
RH1 = 0.5;                      % relative humidity entering compressor
RH3 = 0.75;                     % relative humidity entering fuel cell
-- use whole percentage points
RH3_init = RH3;                 % record of original RH3 value
```

```
RH_H2 = 1;                      % hydrogen humidity entering fuel cell
SR_min = 2;                     % minimum stoichiometric ratio
SR_max = 2.5;                   % maximum stoichiometric ratio
nm = 0.88;                      % combined compressor motor and
inverter efficiency
nfan = 0.65;                    % fan efficiency
ne = 0.81;                      % assumed efficiency of expander
max_pvalues = 100;              % maximum number of pressure values
examined for efficiency
max_airvalues = 20;             % maximum number of airflow values
examined for efficiency
PR_knob = -.005;                % adjusts pressure range in examination
smair_knob = -2;                % adjusts airflow range in examination

% minimum parasitics
minpumpflow = 0.2;              % minimum coolant flow (kg/s)
min_smair = 0;                  % minimum standard mass flow of air
(SLPS)

%-----------------------------------------------------------------------
% Initial Calculations
%-----------------------------------------------------------------------

den_air = Ma*P1*101325/(8314*T1);       % approx. density of dry air
entering compressor (kg/m^3)
den_vap = Mh2o*P1*101325/(8314*T1);     % ideal gas density of water
vapor at input conditions (kg/m^3)
Pg1 = 10^(2.95E-2*(T1-273)-9.18E-5*(T1-273)^2+1.44E-7*(T1-273)^3-2.18);
% inlet saturation pressure (T in K and P_sat in atm)

% this section determines the maximum airflow rate for normalization
purposes
mH2_max = num_cell*475*MH2/(96487*2*1000);                  % kg H2/s
(Fuel Cell Handbook, 8-2)
mO2_max = 0.5*(mH2_max*MO2)/MH2;                            % kg O2/s
(Shapiro, 558, equ. 12.1)
maxair_req = (mO2_max*Ma)/(MO2*0.21);                       % kg dry
air/s required for reaction (ibid, mole fraction of O2 from Shapiro
559)
mair1_max = SR_max*maxair_req;                              % actual kg
dry air/s flowing through compressor
mvap1_max = 0.622*mair1_max*Pg1*RH1/(P1-Pg1*RH1);           % mass flow
of h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
max_smair =(mair1_max/den_air + mvap1_max/den_vap)*1000;    % mass flow
of dry air and vapor(standard l/s)
if max_smair<min_smair                                      % makes
sure maximum is not less than minimum
    max_smair = min_smair;
end
PR_opmax = -3E-06*max_smair^3 + 0.0007*max_smair^2 - 0.0064*max_smair +
1.0382;     % max operating press. for turbocompressor--60 kW

% fuel cell current loop
index = 0;                                          % place-keeping
variable
for current = Amin:Astep:Amax
    index = index + 1;
```

```
    SR = 3.381e-6*(current^2)-6e-4*current+2.009;     % operating line
for air stoichiometric ratio--60 kW
                                                      % equation taken
from a curve fit for the following:
                                                      % current:  300
210 120 30
                                                      % SR:       2.5
2.2 2.0 2.0
    if SR<SR_min
        SR = SR_min;
    elseif SR>SR_max
        SR = SR_max
    end

    for hh = 1:max_pvalues    % loop for finding highest efficiency
operating pressure line
        mH2 = num_cell*current*MH2/(96487*2*1000);        % kg H2/s
(Fuel Cell Handbook, 8-2)
        mO2 = 0.5*(mH2*MO2)/MH2;                          % kg O2/s
(Shapiro, 558, equ. 12.1)
        mair_req = (mO2*Ma)/(MO2*0.21);                   % kg dry
air/s required for reaction (ibid, mole fraction of O2 from Shapiro
559)
        mair1 = SR*mair_req;                              % actual kg
dry air/s flowing through compressor
        mvap1 = 0.622*mair1*Pg1*RH1/(P1-Pg1*RH1);         % mass flow
of h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
        smair =(mair1/den_air + mvap1/den_vap)*1000;      % mass flow
of dry air and vapor(standard l/s)
        if smair<min_smair                                % makes sure
standard mass flow does not go below minimum
            smairold = smair;
            smair = min_smair;
            mair1 = smair/smairold*mair1;                 % converts
from old mair1 and mvap1 values to new values
            mvap1 = smair/smairold*mvap1;
        end
        smair_orig = smair;                               % stores
original smair value for reference

        for index2 = 1:max_airvalues                      % loop for
finding highest efficiency airflow rate
            smairold = smair;
            smair = smair_orig + smair_knob + index2*0.1; % standard
mass flow (SLPS)
            mair1 = smair/smairold*mair1;
            mvap1 = smair/smairold*mvap1;

            % The operating pressure line is estimated below with
PR_stack, and later modified
            PR_stack = [1.015 1.015 1.015 1.015 1.015 1.015 1.0158
1.0178 1.0198 1.0218 1.0238 1.0258 1.0278 1.0298 1.0318 1.0338 1.0529
1.0647 1.0766 1.0884 1.1003 1.1121 1.0747 1.0822 1.0901 1.0985 1.1073
1.1165 1.1262 1.1362 1.1468 1.1577 1.1694 1.1818 1.1453 1.1586 1.1725
1.187 1.1526 1.2175 1.2336 1.2503 1.2675 1.2854 1.3038 1.2735 1.2932
1.3629 1.3345 1.3561 1.3783 1.4013 1.3755 1.3999 1.4249 1.4506 1.4771
```

```
1.5044 1.5324 1.5118 1.5414 1.5717 1.6029 1.6349 1.6678 1.7015 1.736
1.7221 1.7585 1.7958 1.7846 1.8238 1.8639 1.9049 1.947 1.99 2.0341
2.0298 2.0759 2.0737 2.0233 2.0232 2.0243 2.0264 2.079 2.0834 2.0889
2.1449 2.1527 2.211 2.2704 2.2817 2.2942 2.1104 2.1253];
            PR = PR_stack(current/5);               % each pressure
value is chosen according to current
            P2 = PR*P1;                             % base compressor
pressure (will be modified later)
            pressure_act = PR;                      % pressure is
stored
            PR = pressure_act + PR_knob + hh*0.005; % modified pressure
            P2 = PR*P1;
            if PR<1                                 % increases
pressure to at least ambient
                PR = 1;
                P2 = PR*P1;
            end

            % initializing variables
            T3 = T3_init;
            T4 = T4_init;
            T_h2o = T_h2o_init;

            % approximate fuel cell calculations to make guess at
Q_stack and m_cool
            pO2 = P2*0.21;                          % partial pressure
of oxygen in dry air
            J = current/area_cell;                  % current density
            V_guess = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4)*J+0.14*log10(pO2)+(T4-333)*0.00041*log10(10^3*J);   % revised
Honeywell curve with low temp. effects (5/2/01)
            Pwr_f_guess = V_guess*num_cell*current; % stack power (W)
            Q_guess = mH2*H2_HHV-Pwr_f_guess;       % initial fuel cell
heat rejection value (W)

            % use of a 50gpm pump with max power of 1.2 bHP (CD-100
from Pricepump)
            m_cool = Q_guess/(10*cp_h2o);           % coolant flow
giving a 10°C temp diff across stack
            Pwr_pump = maxpump/maxpumpflow*m_cool;  % linear power with
max determined by Price Pump website

            if Pwr_pump>maxpump                     % checks to see if
the pump power exceeds the maximum
                Pwr_pump = maxpump;                 % if the power does
exceed the max, it is set to the maximum (kg/s)
                m_cool = maxpumpflow;               % coolant flow is
also set to maximum (kg/s)
                'stack will overheat -- reached max of coolant pump'
            elseif m_cool<minpumpflow
                m_cool = minpumpflow;
                Pwr_pump = maxpump/maxpumpflow*m_cool;    % linear
power with max determined by Price Pump website
                T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;  % temperature
achievable in humidifier air stream (assumes 5° min. difference btwn
coolant and air)
```

```
                T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);   % stack exit
coolant temp
            end
%-------------------------------------------------------------------------
% Compressor Section
%-------------------------------------------------------------------------

            % normalize smair based on itself to place in range of
compressor
            smair_norm = smair/max_smair;

            % normalizing values for compressor data
            max_flow = 76;            % normalized for high efficiency
(SLPS)
            max_PR = 3.51;
            max_eff = 0.796;
            max_speed = 105000;
            min_speed = 10000;

            PR_norm = PR/max_PR;      % scaled based on compressor
maximum

            % first check to make sure the input point is below surge
line
            surge = 0;
            if smair_norm <= s20K(1,1)
                P_surge = s20K(2,1) - (s20K(2,1)-s10K(2,1))/(s20K(1,1)-
s10K(1,1))*(s20K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s20K(1,1)) & (smair_norm<=s30K(1,1))
                P_surge = s30K(2,1) - (s30K(2,1)-s20K(2,1))/(s30K(1,1)-
s20K(1,1))*(s30K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s30K(1,1)) & (smair_norm<=s40K(1,1))
                P_surge = s40K(2,1) - (s40K(2,1)-s30K(2,1))/(s40K(1,1)-
s30K(1,1))*(s40K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s40K(1,1)) & (smair_norm<=s50K(1,1))
                P_surge = s50K(2,1) - (s50K(2,1)-s40K(2,1))/(s50K(1,1)-
s40K(1,1))*(s50K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s50K(1,1)) & (smair_norm<=s60K(1,1))
                P_surge = s60K(2,1) - (s60K(2,1)-s50K(2,1))/(s60K(1,1)-
s50K(1,1))*(s60K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s60K(1,1)) & (smair_norm<=s70K(1,1))
```

```
                P_surge = s70K(2,1) - (s70K(2,1)-s60K(2,1))/(s70K(1,1)-
s60K(1,1))*(s70K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s70K(1,1)) & (smair_norm<=s80K(1,1))
                P_surge = s80K(2,1) - (s80K(2,1)-s70K(2,1))/(s80K(1,1)-
s70K(1,1))*(s80K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s80K(1,1)) & (smair_norm<=s90K(1,1))
                P_surge = s90K(2,1) - (s90K(2,1)-s80K(2,1))/(s90K(1,1)-
s80K(1,1))*(s90K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif (smair_norm>s90K(1,1)) & (smair_norm<=s100K(1,1))
                P_surge = s100K(2,1) - (s100K(2,1)-
s90K(2,1))/(s100K(1,1)-s90K(1,1))*(s100K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            elseif smair_norm>s100K(1,1)
                P_surge = s105K(2,1) - (s105K(2,1)-
s100K(2,1))/(s105K(1,1)-s100K(1,1))*(s105K(1,1)-smair_norm);
                if PR_norm > P_surge
                    surge = 1;
                end
            end

            if surge         % this will place the operating point
directly on the surge line at the requested pressure
                smairold = smair_norm;
                if PR_norm <= s20K(2,1)
                    smair_norm = s20K(1,1) - (s20K(1,1)-
s10K(1,1))/(s20K(2,1)-s10K(2,1))*(s20K(2,1)-PR_norm);
                elseif (PR_norm>s20K(2,1)) & (PR_norm<=s30K(2,1))
                    smair_norm = s30K(1,1) - (s30K(1,1)-
s20K(1,1))/(s30K(2,1)-s20K(2,1))*(s30K(2,1)-PR_norm);
                elseif (PR_norm>s30K(2,1)) & (PR_norm<=s40K(2,1))
                    smair_norm = s40K(1,1) - (s40K(1,1)-
s30K(1,1))/(s40K(2,1)-s30K(2,1))*(s40K(2,1)-PR_norm);
                elseif (PR_norm>s40K(2,1)) & (PR_norm<=s50K(2,1))
                    smair_norm = s50K(1,1) - (s50K(1,1)-
s40K(1,1))/(s50K(2,1)-s40K(2,1))*(s50K(2,1)-PR_norm);
                elseif (PR_norm>s50K(2,1)) & (PR_norm<=s60K(2,1))
                    smair_norm = s60K(1,1) - (s60K(1,1)-
s50K(1,1))/(s60K(2,1)-s50K(2,1))*(s60K(2,1)-PR_norm);
                elseif (PR_norm>s60K(2,1)) & (PR_norm<=s70K(2,1))
                    smair_norm = s70K(1,1) - (s70K(1,1)-
s60K(1,1))/(s70K(2,1)-s60K(2,1))*(s70K(2,1)-PR_norm);
                elseif (PR_norm>s70K(2,1)) & (PR_norm<=s80K(2,1))
                    smair_norm = s80K(1,1) - (s80K(1,1)-
s70K(1,1))/(s80K(2,1)-s70K(2,1))*(s80K(2,1)-PR_norm);
                elseif (PR_norm>s80K(2,1)) & (PR_norm<=s90K(2,1))
```

```
                    smair_norm = s90K(1,1) - (s90K(1,1)-
s80K(1,1))/(s90K(2,1)-s80K(2,1))*(s90K(2,1)-PR_norm);
                elseif (PR_norm>s90K(2,1)) & (PR_norm<=s100K(2,1))
                    smair_norm = s100K(1,1) - (s100K(1,1)-
s90K(1,1))/(s100K(2,1)-s90K(2,1))*(s100K(2,1)-PR_norm);
                elseif PR_norm>s100K(2,1)
                    smair_norm = s105K(1,1) - (s105K(1,1)-
s100K(1,1))/(s105K(2,1)-s100K(2,1))*(s105K(2,1)-PR_norm);
                end
                mair1 = smair_norm/smairold*mair1;
                mvap1 = smair_norm/smairold*mvap1;
                smair = smair_norm*max_smair;
                if PR_norm>0.997
                    PR_norm = 0.997;
                end
            end

            % flags
            check = 0;      % indicates if input point is within smair
range of speed line
            done = 0;       % indicates if top and bottom speed lines
have been found
            range = 0;      % indicates if the input point was ever in
a smair range
            special1 = 0;   % indicates a special case (operating point
is just below lowest map line)
            special2 = 0;   % indicates a special case (operating point
is left of lowest map line)

% This section determines where the input operating point is on the
compressor map

            if
and(smair_norm<=max(s10K(1,:)),smair_norm>=min(s10K(1,:)))   % if smair
is between max and min of s10K flow values
                x = 1;
                while ~check
                    A1 = s10K(1,x);
                    A2 = s10K(1,x+1);
                    if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s10K
                        check = 1;   % allow to get out of while loop
                        range = 1;
                        P = s10K(2,x+1) - (s10K(2,x+1) - s10K(2,x))/(A2
- A1)*(A2 - smair_norm);  % interp. of pressure on s10K at smair
                        if P>=PR_norm
                            done = 1;
                            special1 = 1;
                        end
                    else
                        x = x + 1;
                    end
                end
            elseif smair_norm<min(s10K(1,:))
                x = 1;
                done = 1;
                special2 = 1;
```

```
            end

            check = 0;
            if and(~done,
and(smair_norm<=max(s20K(1,:)),smair_norm>=min(s20K(1,:))))   % if
smair is between max and min of s20K flow values
                x = 1;
                while ~check
                    A1 = s20K(1,x);
                    A2 = s20K(1,x+1);
                    if and(smair_norm<=A2, smair_norm>=A1)   % if smair
is between two selected flows in s20K
                        check = 1;      % allow to get out of while
loop
                        range = 1;
                        P = s20K(2,x+1) - (s20K(2,x+1) - s20K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s20K at smair
                        if P>=PR_norm
                            done = 1;
                            top = s20K;
                            bot = s10K;
                        end
                    else
                        x = x + 1;
                    end
                end
            elseif and(range, ~done)
                top = s20K;
                bot = s10K;
                done = 1;
            elseif and(smair_norm<min(s20K(1,:)), ~done)
                P = s20K(2,2) - (s20K(2,2) - s20K(2,1))/(s20K(1,2) -
s20K(1,1))*(s20K(1,2) - smair_norm);  %interp. of pressure on s20K at
smair
                done = 1;
                top = s20K;
                bot = s10K;
            end

            check = 0;
            if and(~done,
and(smair_norm<=max(s30K(1,:)),smair_norm>=min(s30K(1,:))))  % if smair
is between max and min of s30K flow values
                x = 1;
                while ~check
                    A1 = s30K(1,x);
                    A2 = s30K(1,x+1);
                    if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s30K
                        check = 1;  % allow to get out of while loop
                        range = 1;
                        P = s30K(2,x+1) - (s30K(2,x+1) - s30K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s30K at smair
                        if P>=PR_norm
                            done = 1;
                            top = s30K;
                            bot = s20K;
```

```
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s30K;
            bot = s20K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s40K(1,:)),smair_norm>=min(s40K(1,:))))  % if smair
is between max and min of s40K flow values
            x = 1;
            while ~check
                A1 = s40K(1,x);
                A2 = s40K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s40K
                    check = 1;  % allow to get out of while loop
                    range = 1;
                    P = s40K(2,x+1) - (s40K(2,x+1) - s40K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s40K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s40K;
                        bot = s30K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s40K;
            bot = s30K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s50K(1,:)),smair_norm>=min(s50K(1,:))))  % if smair
is between max and min of s50K flow values
            x = 1;
            while ~check
                A1 = s50K(1,x);
                A2 = s50K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s50K
                    check = 1;  % allow to get out of while loop
                    range = 1;
                    P = s50K(2,x+1) - (s50K(2,x+1) - s50K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s50K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s50K;
```

```
                        bot = s40K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s50K;
            bot = s40K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s60K(1,:)),smair_norm>=min(s60K(1,:))))  % if smair
is between max and min of s60K flow values
            x = 1;
            while ~check
                A1 = s60K(1,x);
                A2 = s60K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s60K
                    check = 1;  % allow to get out of while loop
                    range = 1;
                    P = s60K(2,x+1) - (s60K(2,x+1) - s60K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s60K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s60K;
                        bot = s50K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s60K;
            bot = s50K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s70K(1,:)),smair_norm>=min(s70K(1,:))))  % if smair
is between max and min of s70K flow values
            x = 1;
            while ~check
                A1 = s70K(1,x);
                A2 = s70K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s70K
                    check = 1;  % allow to get out of while loop
                    range = 1;
                    P = s70K(2,x+1) - (s70K(2,x+1) - s70K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s70K at smair
                    if P>=PR_norm
                        done = 1;
```

```
                        top = s70K;
                        bot = s60K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s70K;
            bot = s60K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s80K(1,:)),smair_norm>=min(s80K(1,:))))  % if smair
is between max and min of s80K flow values
            x = 1;
            while ~check
                A1 = s80K(1,x);
                A2 = s80K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s80K
                    check = 1;  % allow to get out of while loop
                    range = 1;
                    P = s80K(2,x+1) - (s80K(2,x+1) - s80K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s80K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s80K;
                        bot = s70K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s80K;
            bot = s70K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s90K(1,:)),smair_norm>=min(s90K(1,:))))  % if smair
is between max and min of s90K flow values
            x = 1;
            while ~check
                A1 = s90K(1,x);
                A2 = s90K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s90K
                    check = 1;  % allow to get out of while loop
                    range = 1;
                    P = s90K(2,x+1) - (s90K(2,x+1) - s90K(2,x))/(A2
- A1)*(A2 - smair_norm);  %interp. of pressure on s90K at smair
                    if P>=PR_norm
```

```
                            done = 1;
                            top = s90K;
                            bot = s80K;
                        end
                    else
                        x = x + 1;
                    end
                end
            elseif and(range, ~done)
                top = s90K;
                bot = s80K;
                done = 1;
            end

            check = 0;
            if and(~done,
and(smair_norm<=max(s100K(1,:)),smair_norm>=min(s100K(1,:))))  % if
smair is between max and min of s100K flow values
                x = 1;
                while ~check
                    A1 = s100K(1,x);
                    A2 = s100K(1,x+1);
                    if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s100K
                        check = 1;  % allow to get out of while loop
                        range = 1;
                        P = s100K(2,x+1) - (s100K(2,x+1) -
s100K(2,x))/(A2 - A1)*(A2 - smair_norm);  %interp. of pressure on s100K
at smair
                        if P>=PR_norm
                            done = 1;
                            top = s100K;
                            bot = s90K;
                        end
                    else
                        x = x + 1;
                    end
                end
            elseif and(range, ~done)
                top = s100K;
                bot = s90K;
                done = 1;
            end

            check = 0;
            if and(~done,
and(smair_norm<=max(s105K(1,:)),smair_norm>=min(s105K(1,:))))  % if
smair is between max and min of s105K flow values
                x = 1;
                while ~check  % finds points that the input point lies
between
                    A1 = s105K(1,x);
                    A2 = s105K(1,x+1);
                    if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s105K
                        check = 1;  % allow to get out of while loop
```

```
                        P = s105K(2,x+1) - (s105K(2,x+1) -
s105K(2,x))/(A2 - A1)*(A2 - smair_norm);  %interp. of pressure on s105K
at smair
                        if P>=PR_norm
                            done = 1;
                            top = s105K;
                            bot = s100K;
                        end
                    else
                        x = x + 1;
                    end
                end
            elseif and(range, ~done)
                top = s105K;
                bot = s100K;
                done = 1;
            end

            while ~done                     % op. point is beyond
highest speed line on map, so pressure will be lowered until acceptable
                PR_norm = PR_norm-0.01;    % lowers pressure
                P2 = PR_norm*max_PR*P1;
                check = 0;
                x = 1;
                while ~check                % finds points that the
input point lies between
                    A1 = s105K(1,x);
                    A2 = s105K(1,x+1);
                    if and(smair_norm<=A2, smair_norm>=A1)  % if smair
is between two selected flows in s105K
                        check = 1;          % allow to get out of while
loop
                        P = s105K(2,x+1) - (s105K(2,x+1) -
s105K(2,x))/(A2 - A1)*(A2 - smair_norm);  % interp. of pressure on
s105K at smair
                        if P>=PR_norm
                            done = 1;
                            top = s105K;
                            bot = s100K;
                        end
                    else
                        x = x + 1;
                    end
                end
            end

% This section defines two special cases and gives the speed,
efficiency, and changes in operating point

            if special1                             % this is the special
case if the input point is below the 10k speed line but smair range
                comp_speed = min_speed;             % compressor speed is
set to at least minimum speed
                PR_norm = s10K(2,x+1) - (s10K(2,x+1) - s10K(2,x))/(A2 -
A1)*(A2 - smair_norm);   % pressure is raised to pressure on 10k map
line at smair
                P2 = PR_norm*max_PR*P1;             % adjustment to P2
```

```
                eff_overall = (s10K(3,x+1) - (s10K(3,x+1) -
s10K(3,x))/(A2 - A1)*(A2 - smair_norm))*max_eff;  % interp. of
efficiency on s10K at smair

            elseif special2  % this is the special case if the input
point is below the 10k speed line and less than smair range
                comp_speed = min_speed;          % compressor speed is
set to at least minimum speed
                smair_bak = smair;               % storing old smair
value
                smair = s10K(1,1)*max_smair;     % smair is set to
lowest compressor map smair
                mair1 = smair/smair_bak*mair1;   % mair1 is updated
                mvap1 = smair/smair_bak*mvap1;   % mvap1 is updated
                P2 = s10K(2,1)*max_PR*P1;        % pressure is set to
pressure at new smair
                eff_overall = s10K(3,1)*max_eff; % efficiency is set to
efficiency at new smair
            else

% This section finds the two closest points to the input point on the
top speed line
% (point1 = closest, point2 = 2nd closest)

                dist1 = sqrt((PR_norm - top(2,1))^2 + (smair_norm -
top(1,1))^2);
                dist2 = sqrt((PR_norm - top(2,2))^2 + (smair_norm -
top(1,2))^2);
                point1 = 1;
                point2 = 2;
                if dist2<dist1                  % sets shortest
distance at dist1
                    dist3 = dist1;
                    dist1 = dist2;
                    dist2 = dist3;
                    point1 = 2;
                    point2 = 1;
                end
                for z=3:length(top)
                    dist3 = sqrt((PR_norm - top(2,z))^2 + (smair_norm -
top(1,z))^2);
                    if dist3<dist2
                        if dist3<dist1
                            dist2 = dist1;
                            point2 = point1;
                            dist1 = dist3;
                            point1 = z;
                        else
                            dist2 = dist3;
                            point2 = z;
                        end
                    end
                end

                if PR_norm==P  % this is the special case if the input
point is directly on one of the speed lines
                    comp_speed = top(4,1)*max_speed;
```

```
                    % find percent distance between points, and
translate that into efficiency
                    sec1 = sqrt((top(1,point1)-smair_norm)^2 +
(top(2,point1)-PR_norm)^2);
                    sec2 = sqrt((top(1,point2)-smair_norm)^2 +
(top(2,point2)-PR_norm)^2);
                    pcnt = sec1/(sec1 + sec2);  % percent of distance
that input point is from point1 to point2 on top speed line
                    eff_overall = (top(3,point1) + pcnt*(top(3,point2)
- top(3,point1)))*max_eff;
                else
                    % This section finds the point on the top speed
line that, if connected to the input point, would make a perpendicular
line
                    % (x = smair_norm, y = PR_norm)

                    check = 0;
                    u = ((smair_norm-top(1,point1))*(top(1,point2)-
top(1,point1))+(PR_norm-top(2,point1))*(top(2,point2)-
top(2,point1)))/((sqrt((top(1,point2)-top(1,point1))^2+(top(2,point2)-
top(2,point1))^2))^2);
                    smairtop = top(1,point1) + u*(top(1,point2) -
top(1,point1));
                    PRtop = top(2,point1) + u*(top(2,point2) -
top(2,point1));

                    % This section finds the point on the bottom speed
line through which the perpendicular to the top speed line goes
                    % (1 = input, 2 = point for perpendicular, 3 and 4
= points on lower line)
                    % first, special end cases
                    % beginning of bottom speed line
                    ua = ((bot(1,2)-bot(1,1))*(PR_norm - bot(2,1)) -
(bot(2,2)-bot(2,1))*(smair_norm-bot(1,1)))/((bot(2,2)-
bot(2,1))*(smairtop-smair_norm) - (bot(1,2)-bot(1,1))*(PRtop -
PR_norm));
                    smairbot = smair_norm + ua*(smairtop - smair_norm);
                    PRbot = PR_norm + ua*(PRtop - PR_norm);
                    if smairbot<=bot(1,2)
                        check = 1;
                    end
                    % middle of bottom speed line
                    x = 1;
                    while ~check    & x<8
                        x = x + 1;
                        ua = ((bot(1,x+1)-bot(1,x))*(PR_norm -
bot(2,x)) - (bot(2,x+1)-bot(2,x))*(smair_norm-bot(1,x)))/((bot(2,x+1)-
bot(2,x))*(smairtop-smair_norm) - (bot(1,x+1)-bot(1,x))*(PRtop -
PR_norm));
                        smairbot = smair_norm + ua*(smairtop -
smair_norm);
                        PRbot = PR_norm + ua*(PRtop - PR_norm);
                        if and(smairbot>=bot(1,x),smairbot<=bot(1,x+1))
                            check = 1;
                        end
                    end
                    % end of bottom speed line
```

```
                    if ~check
                        ua = (((bot(1,9))-bot(1,8))*(PR_norm -
bot(2,8)) - (bot(2,9)-bot(2,8))*(smair_norm-bot(1,8)))/((bot(2,9)-
bot(2,8))*(smairtop-smair_norm) - (bot(1,9)-bot(1,8))*(PRtop -
PR_norm));
                        smairbot = smair_norm + ua*(smairtop -
smair_norm);
                        PRbot = PR_norm + ua*(PRtop - PR_norm);
                        if smairbot>=bot(1,8)
                            check = 1;
                        end
                    end
                    % This section finds the compressor speed

                    d1 = sqrt((smairtop-smair_norm)^2 + (PRtop -
PR_norm)^2);
                    d2 = sqrt((smairbot-smair_norm)^2 + (PRbot -
PR_norm)^2);
                    pcnt = d1/(d1 + d2);
                    comp_speed = (top(4,1) + pcnt*(bot(4,1) -
top(4,1)))*max_speed;

                    % This section develops the efficiency curve for
the given speed and finds the efficiency on that curve

                    eff1 = top(3,1) + pcnt*(bot(3,1) - top(3,1));
                    smair1 = top(1,1) + pcnt*(bot(1,1) - top(1,1));
                    check = 0;
                    x = 1;
                    if top(4,1)~=1
                        while and(~check, x<=8)
                            x = x + 1;
                            eff2 = eff1;
                            smair2 = smair1;
                            eff1 = top(3,x) + pcnt*(bot(3,x) -
top(3,x));
                            smair1 = top(1,x) + pcnt*(bot(1,x) -
top(1,x));
                            if smair_norm<=smair1
                                % now find percent difference of smair
between two points on efficiency curve, and use that to find eff.
                                check = 1;
                                pcnt2 = (smair2 - smair_norm)/(smair2 -
smair1);
                                eff_overall = (eff2 + pcnt2*(eff1 -
eff2))*max_eff;
                            end
                        end
                    else
                        while and(~check, x<=6)   % for the special
case where s105K is used as "top" (because of its fewer data points)
                            x = x + 1;
                            eff2 = eff1;
                            smair2 = smair1;
                            eff1 = top(3,x) + pcnt*(bot(3,x) -
top(3,x));
```

```
                            smair1 = top(1,x) + pcnt*(bot(1,x) -
top(1,x));
                            if smair_norm<=smair1
                                % now find percent difference of smair
between two points on efficiency curve, and use that to find eff.
                                check = 1;
                                pcnt2 = (smair2 - smair_norm)/(smair2 -
smair1);
                                eff_overall = (eff2 + pcnt2*(eff1 -
eff2))*max_eff;
                            end
                        end
                    end
                    if ~check  % takes into account if input point is
beyond smair range of other points on efficiency curve
                        pcnt2 = (smair2 - smair_norm)/(smair2 -
smair1);
                        eff_overall = (eff2 + pcnt2*(eff1 -
eff2))*max_eff;
                    end
                end
            end
            % stagnation temps and pressures are not used due to
negligible difference to static values
            T2 = T1*((P2/P1)^((k-1)/k) - 1)/eff_overall + T1;  %
temperature of outlet
            temp_rise = T2 - T1;
            Pwr_c = ((mair1/Ma + mvap1/Mh2o)*8314*T1*k/(k-
1)*((P2/P1)^((k-1)/k)-1))/(eff_overall*nm);  % compressor power
            vol_eff = 0;  % undefined for turbomachine

%-----------------------------------------------------------------------
% Humidifier/Heat Exchanger Section
%-----------------------------------------------------------------------
            P3 = P2;            % no pressure drop over humidifier

            cont = 0;
            while ~cont         % this loop checks to make sure the
humidifier doesn't use more heat than is given by stack
                RH3 = RH3_init; % initializing RH3
                % hydrogen humidification
                Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-273)^2+1.44E-
7*(T4-273)^3-2.18);  % saturation pressure in fuel cell inlet for
hydrogen (T in K and P_sat in atm)
                mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);  %
water vapor in H2 (assumes initially dry H2 and can make it to fc temp)
                % air humidification
                Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-273)^2+1.44E-
7*(T3-273)^3-2.18); % water sat. pressure for H2
                if T3==T4
                    mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);   % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;         % net water
added in humidifier
                    mtot3 = mair1 + mvap3;             % total mass
flow into fuel cell
                else  % effects of coolant loop minimum
```

```
                    mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);     %
vapor mass needed to humidify air to correct RH when warmed to fuel
cell temp (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;         % net water
added in humidifier
                    mtot3 = mair1 + mvap3;             % total mass
flow into fuel cell
                    RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);  %
relative humidity of air at humidifier exit
                    if RH3>1
                        'lower than requested fuel cell inlet air RH'
                        RH3 = 1;
                        mvap3 = 0.622*mair1*Pg3/(P3-Pg3);  % vapor mass
going into fuel cell (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;     % net water
added in humidifier
                        mtot3 = mair1 + mvap3;
                        Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);
                        RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  %relative humidity of air
after warming up in fuel cell
                    end
                end
                Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-T2)
+ mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o);  %total heat needed to
warm mass flow into fuel cell

                while Q_needed<Q_guess & T4<T4_init    % raises the
temp of stack if heat is available
                    RH3 = RH3_init;   % initializing RH3
                    T4 = T4 + .1;
                    if T4>T4_init
                        T4 = T4_init;
                    end
                    T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;  %
temperature achievable in humidifier air stream (assumes 5° min.
difference btwn coolant and air)
                    T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);   % water
temp same as stack coolant exit temp
                    Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-
273)^2+1.44E-7*(T3-273)^3-2.18);    % saturation pressure in fuel cell
inlet (T in K and P_sat in atm)
                    Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);    % saturation pressure in fuel cell
inlet (T in K and P_sat in atm)
                    mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 -
Pg4*RH_H2);  % water vapor in H2 (assumes initially dry H2 and can make
it to fc temp)
                    if T3==T4
                        mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);  %
vapor mass going into fuel cell (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;   % net water added
in humidifier
                        mtot3 = mair1 + mvap3;
                    else      %effects of coolant loop minimum
```

```
                        mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);  %
vapor mass needed to humidify air to correct RH when warmed to fuel
cell temp (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;                 %
net water added in humidifier
                        mtot3 = mair1 + mvap3;
                        RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);  %
relative humidity of air at humidifier exit
                        if RH3>1
                            'lower than requested fuel cell inlet air
RH'
                            RH3 = 1;
                            mvap3 = 0.622*mair1*Pg3/(P3-Pg3);  % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                            mh2o_addh = mvap3 - mvap1;         % net
water added in humidifier
                            mtot3 = mair1 + mvap3;
                            Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);
                            RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  %relative humidity of air
after warming up in fuel cell
                        end
                    end
                    Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-
T2) + mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o);
                end

                temp_mod = 0;          % flag to show temperature
modification
                while Q_needed>Q_guess % lowers stack temperature to
allow humidifier to heat to temperature
                    temp_mod = 1;
                    RH3 = RH3_init;    % initializing RH3
                    T4 = T4 - .1;      % incremental decrease in stack
ave temperature
                    T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;     %
temperature achievable in humidifier air stream (assumes 5° min.
difference btwn coolant and air)
                    T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);         %
water temp same as stack coolant exit temp
                    Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-
273)^2+1.44E-7*(T3-273)^3-2.18);    % saturation pressure in fuel cell
inlet (T in K and P_sat in atm)
                    Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);    % saturation pressure in fuel cell
inlet (T in K and P_sat in atm)
                    mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 -
Pg4*RH_H2); % water vapor in H2 (assumes initially dry H2 and can make
it to fc temp)
                    if T3==T4
                        mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);   %
vapor mass going into fuel cell (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;         %net water
added in humidifier
                        mtot3 = mair1 + mvap3;
```

```
                    else                          % effects of coolant
loop minimum
                        mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);
% vapor mass needed to humidify air to correct RH when warmed to fuel
cell temp (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;         % net water
added in humidifier
                        mtot3 = mair1 + mvap3;
                        RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);
      %relative humidity of air at humidifier exit
                        if RH3>1
                            'lower than requested fuel cell inlet air
RH'
                            RH3 = 1;
                            mvap3 = 0.622*mair1*Pg3/(P3-Pg3);
%vapor mass going into fuel cell (Shapiro, 582, equ 12.43)
                            mh2o_addh = mvap3 - mvap1;         %net
water added in humidifier
                            mtot3 = mair1 + mvap3;
                            Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);
                            RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  %relative humidity of air
after warming up in fuel cell
                        end
                    end
                    Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-
T2) + mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o);
                    % Q_needed shows amount of heat needed to warm air
stream to T3
                end
                Q_hum = Q_needed;

%-----------------------------------------------------------------------
% Fuel Cell Section
%-----------------------------------------------------------------------
                Pdrop = resist*(mair1*den_air + (mtot3-mair1)*den_vap);
% pressure loss across system
                P4 = P3 - Pdrop;
% pressure at exit of fuel cell
                PO2_3 = (P3 - RH3*Pg3)*0.21;
% partial pressure of oxygen at fuel cell inlet
                mO2_used = (num_cell*current*MO2)/(96487*4000);
% mass flow of oxygen consumed (kg/s)
                mair4 = mair1 - mO2_used;
% total mass flow of dry air at fuel cell exit
                O2_ratio4 = (mair1*0.21-mO2_used)/mair4;
% ratio of oxygen to dry air at fuel cell exit
                mvapmax = 0.622*mair1*Pg4/(P4-Pg4);
% total vapor at 100% RH at fuel cell outlet
                mvap_fc = mvapmax - mvap3;
% total vapor acquired by airstream from fuel cell water production
                mh2o_prod = mH2*Mh2o/MH2;
% stack water production (kg/s)
                h2o_used = 0;
                if mvap_fc>mh2o_prod
                    h2o_used = 1;
```

```
                    mvap_fc = mh2o_prod;
% if production of water is less than total that the air can accept for
100% humidity, then air will only take total production of water
                    RH4 = ((mvap3+mvap_fc)*(P3-
Pdrop))/((0.622*mair1+mvap3+mvap_fc)*Pg4);      %relative humidity of
air at fuel cell outlet
                else
                    RH4 = 1;
                    h2o_used = 0;
                end
                PO2_4 = (P4 - RH4*Pg4)*O2_ratio4;  % partial pressure
of oxygen at fuel cell exit
                pO2 = (PO2_3 + PO2_4)/2;           % average partial
pressure of oxygen
                J = current/area_cell;             % current density
                V = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4)*J+0.14*log10(pO2)+(T4-333)*0.00041*log10(10^3*J);
      %revised Honeywell curve with temp. effects (5/2/01)
                if V <0.5
                    'data for this current level is bad'
                end
                Vstack = V * num_cell;             % stack voltage (V)
                Pwr_f = Vstack*current;            % stack power (W)
                mvap4 = mvap3 + mvap_fc;
                n_stack = Pwr_f/(mH2*H2_LHV);      % stack efficiency
                Q_stack = mH2*(1 - mvap_fc/mh2o_prod)*H2_HHV +
mH2*(mvap_fc/mh2o_prod)*H2_LHV - Pwr_f;               %stack heat
rejection (W)
                mh2o_net = mh2o_prod - mvap_fc; % water recoverable as
liquid at fuel cell outlet

                if ((Q_guess/Q_stack)>1.01|(Q_guess/Q_stack)<.99)   %
compares actual stack heat rejection with guess
                    if (Q_guess/Q_stack)<.99   % this only allows the
initialization of temp if there is more heat available from stack
                        T4 = T4+.1;
                        if T4 > T4_init
                            T4 = T4_init;
                        end
                        T3 = T4;
                        T_h2o = T4+5;
                    end
                    Q_guess = Q_stack;
                    cont = 0;
                    m_cool = Q_guess/(10*cp_h2o);           % coolant
flow giving a 10°C temp diff across stack
                    Pwr_pump = maxpump/maxpumpflow*m_cool;  % linear
power with max determined by Price Pump website
                    if Pwr_pump>maxpump
                        Pwr_pump = maxpump;
                        m_cool = maxpumpflow;
                    elseif m_cool<minpumpflow
                        m_cool = minpumpflow;
                        Pwr_pump = maxpump/maxpumpflow*m_cool;  %
linear power with max determined by Price Pump website
                        T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o); % stack
coolant exit temp
```

```
                        T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;% max
temperature achievable in humidifier air stream (assumes 5° min.
difference btwn coolant and air)
                    end
                else
                    cont = 1;
                end
            end %while (in humidifier)

            press_check = 1;    % '1' stands for good, that is, enough
pressure to overcome pressure drop over system
            if (P2-Pdrop)<P1    % checks for enough pressure to
overcome pressure drop over system
                press_check = 0;% '0' stands for bad, that is, not
enough pressure
            end

%-----------------------------------------------------------------------
% Radiator Section
%-----------------------------------------------------------------------
            % the NTU method is used for the radiator calculations
            T5 = T4 - Q_hum/(m_cool*cp_h2o);   % temperature of coolant
entering radiator
            if T5 < T1
                'Warning!  Coolant temp. entering radiator is lower
than ambient'
            end
            Q_rad = Q_stack - Q_hum;

            % air properties with fan on
            m_on = radair_on*den_air;
            C_on = m_on*cp_air;
            C_h2o = m_cool*cp_h2o;
            C1 = [C_on C_h2o];
            Cr_on = min(C1)/max(C1);
            NTU_on = UA_on/min(C1);
            eff_on = 1-exp((1/Cr_on)*NTU_on^.22*(exp(-
Cr_on*NTU_on^.78)-1));
            Qmax_on = min(C1)*(T5-T1);
            Q_on = eff_on*Qmax_on;

            % air properties with fan off
            m_off = radair_off*den_air;
            C_off = m_off*cp_air;
            C_h2o = m_cool*cp_h2o;
            C2 = [C_off C_h2o];
            Cr_off = min(C2)/max(C2);
            NTU_off = UA_off/min(C2);
            eff_off = 1-exp((1/Cr_off)*NTU_off^.22*(exp(-
Cr_off*NTU_off^.78)-1));
            Qmax_off = min(C2)*(T5-T1);
            Q_off = eff_off*Qmax_off;

            bypass_pcnt = 0;
            fan_timeon = (Q_rad-Q_off)/(Q_on-Q_off);
            if fan_timeon < 0
                bypass_pcnt = 1 - Q_rad/Q_off;
```

```
            fan_timeon = 0;
        elseif fan_timeon > 1
            fan_timeon = 1;
        end
        avePwr_fan = (fan_timeon*maxfan)/nfan;

        Q_out = Q_on*fan_timeon + Q_off*(1-fan_timeon)*(1-
bypass_pcnt);    % heat rejected from radiator
        T6 = T5 - Q_out/(m_cool*cp_h2o);    % energy balance, temp
of coolant entering fuel cell
        T7 = T6 + Q_stack/(m_cool*cp_h2o);  % energy balance, temp
of coolant exiting fuel cell and entering humidifier

%-------------------------------------------------------------------------
% Expander Section
%-------------------------------------------------------------------------

        % stagnation temps and pressures are not used due to
negligible difference to static values

        Pwr_e = ((mair4/Ma + mvap4/Mh2o)*8314*T4*k/(1-
k)*((P1/P4)^((k-1)/k)-1))*ne;
        Tout = T4 - T4*ne*(1-(P1/P4)^((k-1)/k));
        Pg_out = 10^(2.95E-2*(Tout-273)-9.18E-5*(Tout-273)^2+1.44E-
7*(Tout-273)^3-2.18);  % sat. pressure at fuel cell outlet (T in K and
Pg in atm)
        mvap_out = 0.622*mair4*RH4*Pg_out/(P1-RH4*Pg_out);   % mass
flow of vapor leaving expander
        mh2o_rec = mvap4 - mvap_out;   % mass flow of liquid water
recoverable from expander outlet
        if Pwr_e <0
            Pwr_e = 0;
        end
        if mh2o_rec < 0
            mh2o_rec = 0;
        end

%-------------------------------------------------------------------------
% System Output and Plots
%-------------------------------------------------------------------------
        Pwr_netwexp = Pwr_e + Pwr_f - avePwr_fan - Pwr_c -
Pwr_pump;  % system power with an expander
        n_syswexp = Pwr_netwexp/(mH2*H2_LHV);
% system efficiency with an expander
        Pwr_net = Pwr_f - avePwr_fan - Pwr_c - Pwr_pump;
% system power without an expander
        n_sys = Pwr_net/(mH2*H2_LHV);
% system efficiency without an expander
        mh2o_bal = mh2o_net + mh2o_rec - mh2o_addh - mvap_H2;
% system water balance

        prtest(hh,index2) = n_sys;             % adiabatic
efficiency without expander
        prtest_exp(hh,index2) = n_syswexp;     % adiabatic
efficiency with expander
        comp_eff(hh,index2) = eff_overall;     % compressor
efficiency
```

```
            press_log(hh,index2) = P2;              % compressor output
pressure (atm)
            press_drop(hh,index2) = Pdrop;          % fuel cell pressure
drop (atm)
            flo_log(hh,index2) = smair;             % mass flow rate
through compressor (kg/s)
            enuf_press(hh,index2) = press_check;    % checks to see if
enough pressure is supplied to overcome pressure drop
        end   % for loop (changes index2)
    end       % for loop (changes hh)
    c_eff(:,:,index) = comp_eff';               % compressor efficiency
    presstest(:,:,index) = prtest';             % adiabatic efficiency
without expander
    presstest_exp(:,:,index) = prtest_exp';     % adiabatic efficiency
with expander
    pressure_log(:,:,index) = press_log';       % compressor output
pressure (atm)
    pressr_drop(:,:,index) = press_drop';       % fuel cell pressure
drop (atm)
    flow_log(:,:,index) = flo_log';             % mass flow rate
through compressor (kg/s)
    enough_press(:,:,index) = enuf_press';      % checks to see if
enough pressure is supplied to overcome pressure drop

% these two for loops find the highest efficiency of the compressor
with and without the expander, as well as the mass flow rate
% (exiting compressor) and the pressure (entering fuel cell) at the
highest efficiency
for x = 1:index
    [Y,I] = max(max(presstest_exp(:,:,x).*enough_press(:,:,x)));
    [store_eff(x),i] = max(presstest_exp(:,I,x).*enough_press(:,I,x));
    store_flow(x) = flow_log(i,I,x);
    store_press(x) = max(pressure_log(i,I,x));
end

for x = 1:index
    [Y,I] = max(max(presstest(:,:,x).*enough_press(:,:,x)));
    [store_eff_exp(x),i] = max(presstest(:,I,x).*enough_press(:,I,x));
    store_flow_exp(x) = flow_log(i,I,x);
    store_press_exp(x) = max(pressure_log(i,I,x));
end
```

This program simulates the fuel cell system with the turbocompressor

PROGRAM 4

```
close all
clear all

% This program simulates a fuel cell system using a curve fit to the
highest efficiency pressure line found in another program

% Assumptions/conditions:  constant specific heats, variable stack
temperature but fixed stack humidity, fixed fan
% and compressor motor efficiency, ideal gases, fixed relative humidity
and temperature at entrance to compressor
```

```
% Variable suffixes
% 1 -- compressor air inlet
% 2 -- humidifier air inlet
% 3 -- fuel cell air inlet
% 4 -- expander air inlet
% 5 -- radiator coolant inlet
% 6 -- fuel cell coolant inlet
% 7 -- humidifier coolant inlet

%-------------------------------------------------------------------------
% Constants and System Parameters
%-------------------------------------------------------------------------

% physical constants
cp_air = 1004.5;                % specific heat of air (J/kg-K)
cp_h2o = 4200;                  % specific heat of water (J/kg-K)
cp_vap = 1900;                  % specific heat of vapor (J/kg-K)
cp_H2 = 14400;                  % specific heat of hydrogen (J/kg-K)
e_vap = 2383000;                % approx. energy of vaporization of
water at 50°C (J/kg) (Shapiro 724)
H2_HHV = 141890000;             % higher heating value of hydrogen
(J/kg)
H2_LHV = 119860000;             % lower heating value of hydrogen
(J/kg)
k = 1.4;                        % specific heat ratio of air
Ma = 28.97;                     % molar mass of air (kmol/kg)
MH2 = 2.016;                    % molar mass of hydrogen (kmol/kg)
Mh2o = 18.02;                   % molar mass of water (kmol/kg)
MO2 = 32;                       % molar mass of oxygen (kmol/kg)

%turbomachine constants

s105K = [0.791 0.854 0.924 0.975 1.034 1.068 1.072;             %
normalized mass flow (SLPS)
   0.997 1 0.983 0.957 0.915 0.849 0.772;                       %
normalized pressure ratio
   0.876 0.896 0.922 0.935 0.926 0.841 0.8;                     %
normalized efficiency
   1 0 0 0 0 0 0];                                              %
normalized speed

s100K = [0.757 0.812 0.872 0.928 0.979 1.017 1.034 1.04 1.042;  %
normalized mass flow (SLPS)
   0.935 0.929 0.92 0.897 0.863 0.809 0.741 0.678 0.595;        %
normalized pressure ratio
   0.913 0.936 0.957 0.962 0.95 0.902 0.852 0.807 0.699;        %
normalized efficiency
   0.952 0 0 0 0 0 0 0 0];                                      %
normalized speed

s90K = [0.658 0.709 0.757 0.804 0.85 0.897 0.932 0.945 0.954;   %
normalized mass flow (SLPS)
   0.781 0.778 0.766 0.749 0.727 0.695 0.65 0.595 0.53;         %
normalized pressure ratio
   0.95 0.977 0.984 0.986 0.974 0.945 0.889 0.812 0.705;        %
normalized efficiency
```

```
   0.857 0 0 0 0 0 0 0 0];                                          %
normalized speed

s80K = [0.551 0.59 0.632 0.671 0.709 0.753 0.782 0.804 0.82;     %
normalized mass flow (SLPS)
   0.655 0.652 0.641 0.63 0.615 0.593 0.567 0.524 0.473;         %
normalized pressure ratio
   0.97 0.987 0.998 1 0.987 0.956 0.902 0.824 0.719;             %
normalized efficiency
   0.762 0 0 0 0 0 0 0 0];                                       %
normalized speed

s70K = [0.458 0.491 0.525 0.555 0.59  0.62 0.624 0.668 0.688;    %
normalized mass flow (SLPS)
   0.544 0.541 0.536 0.527 0.516 0.501 0.499 0.456 0.425;        %
normalized pressure ratio
   0.981 0.991 0.995 0.993 0.981 0.952 0.945 0.838 0.715;        %
normalized efficiency
   0.667 0 0 0 0 0 0 0 0];                                       %
normalized speed

s60K = [0.39 0.418 0.445 0.475 0.509 0.53 0.534 0.572 0.59;      %
normalized mass flow (SLPS)
   0.467 0.464 0.459 0.453 0.444 0.439 0.436 0.407 0.385;        %
normalized pressure ratio
   0.981 0.994 0.998 0.994 0.981 0.956 0.946 0.841 0.719;        %
normalized efficiency
   0.571 0 0 0 0 0 0 0 0];                                       %
normalized speed

s50K = [0.325 0.35 0.368 0.393 0.424 0.443 0.445 0.475 0.491;    %
normalized mass flow (SLPS)
   0.407 0.405 0.402 0.399 0.396 0.388 0.385 0.37 0.356;         %
normalized pressure ratio
   0.981 0.991 0.996 0.995 0.981 0.956 0.95 0.841 0.719;         %
normalized efficiency
   0.476 0 0 0 0 0 0 0 0];                                       %
normalized speed

s40K = [0.261 0.282 0.299 0.317 0.333 0.35 0.354 0.38 0.397;     %
normalized mass flow (SLPS)
   0.359 0.356 0.356 0.353 0.35 0.348 0.346 0.339 0.325;         %
normalized pressure ratio
   0.981 0.991 0.996 0.995 0.981 0.956 0.946 0.841 0.719;        %
normalized efficiency
   0.381 0 0 0 0 0 0 0 0];                                       %
normalized speed

s30K = [0.195 0.207 0.222 0.236 0.253 0.268 0.272 0.287 0.295;   %
normalized mass flow (SLPS)
   0.325 0.325 0.325 0.325 0.322 0.316 0.315 0.311 0.302;        %
normalized pressure ratio
   0.981 0.991 0.996 0.995 0.981 0.956 0.95 0.841 0.717;         %
normalized efficiency
   0.286 0 0 0 0 0 0 0 0];                                       %
normalized speed
```

```
s20K = [0.131 0.14 0.146 0.154 0.162 0.171 0.174 0.187 0.195;   %
normalized mass flow (SLPS)
   0.299 0.299 0.299 0.299 0.298 0.296 0.296 0.291 0.288;       %
normalized pressure ratio
   0.981 0.991 0.995 0.995 0.981 0.952 0.946 0.838 0.715;       %
normalized efficiency
   0.191 0 0 0 0 0 0 0 0];                                      %
normalized speed

s10K = [0.056 0.061 0.066 0.071 0.076 0.081 0.082 0.097 0.103;  %
normalized mass flow (SLPS)
   0.293 0.293 0.293 0.293 0.293 0.291 0.291 0.291 0.288;       %
normalized pressure ratio
   0.981 0.987 0.995 0.995 0.981 0.956 0.95 0.841 0.715;        %
normalized efficiency
   0.095 0 0 0 0 0 0 0 0];                                      %
normalized speed

% approx. 60 kW stack (honeywell)
num_cell = 210;                 % number of cells per stack
area_cell = 678;                % cell effective area (cm^2)
maxfan = 250;                   % max power of fan (W)
maxpump = 900;                  % max power of pump (W) -- determined
from Price Pump CD-100, 3600 RPM rating
maxpumpflow = 3.15;             % max mass flow of pump (kg/s) --
determined from Price Pump (see above)
resist = 3.68;                  % flow resistance for fuel cell system
to achieve 0.4 atm drop at max flow (atm-s/kg)
radair_on = 3.63;               % mass flow of air through radiator
with fan on (m^3/s)
radair_off = 0.34;              % mass flow of air through radiator
with fan off (m^3/s)
UA_on = 5100;                   % radiator UA value with fan on
UA_off = 510;                   % radiator UA value with fan off

% input values
Amax = 475;                     % maximum current for fuel cell (A)
Amin = 5;                       % minimum current for fuel cell (A)
Astep = 5;                      % step size for current loop (A)
T1 = 298;                       % temperature entering compressor (K)
T3 = 353;                       % desired temperature of input to stack
(K)
T3_init = T3;                   % record of original T3 value (T3
changes in program)
T4 = T3;                        % output temperature of stack
T4_init = T4;                   % stores original T4 value
T_h2o = T4+5;                   % assumed temperature of water taken
for humidification (stack coolant outlet)
T_h2o_init = T_h2o;             % stores original T_h2o value
P1 = 100000/101325;             % pressure entering compressor (atm)
RH1 = 0.5;                      % relative humidity entering compressor
RH3 = 0.75;                     % relative humidity entering fuel cell
-- use whole percentage points
RH3_init = RH3;                 % record of original RH3 value
RH_H2 = 1;                      % hydrogen humidity entering fuel cell
SR_min = 2;                     % minimum stoichiometric ratio
SR_max = 2.5;                   % maximum stoichiometric ratio
```

```
nm = 0.88;                      % combined compressor motor and
inverter efficiency
nfan = 0.65;                    % fan efficiency
ne = 0.81;                      % assumed efficiency of expander
expnd = 1;                      % 1 if an expander is desired, 0 if not
ploton = 0;                     % 1 if plots are desired, 0 if no plots
are desired

% minimum parasitics
minpumpflow = 0.2;              % minimum coolant flow (kg/s)
min_smair = 0;                  % minimum standard mass flow of air
(SLPS)

%---------------------------------------------------------------------
% Initial Calculations
%---------------------------------------------------------------------

den_air = Ma*P1*101325/(8314*T1);       % approx. density of dry air
entering compressor (kg/m^3)
den_vap = Mh2o*P1*101325/(8314*T1);     % ideal gas density of water
vapor at input conditions (kg/m^3)
Pg1 = 10^(2.95E-2*(T1-273)-9.18E-5*(T1-273)^2+1.44E-7*(T1-273)^3-2.18);
% inlet saturation pressure (T in K and P_sat in atm)

% this section determines the maximum airflow rate for normalization
purposes
mH2_max = num_cell*475*MH2/(96487*2*1000);                  % kg H2/s
(Fuel Cell Handbook, 8-2)
mO2_max = 0.5*(mH2_max*MO2)/MH2;                            % kg O2/s
(Shapiro, 558, equ. 12.1)
maxair_req = (mO2_max*Ma)/(MO2*0.21);                       % kg dry
air/s required for reaction (ibid, mole fraction of O2 from Shapiro
559)
mair1_max = SR_max*maxair_req;                              % actual kg
dry air/s flowing through compressor
mvap1_max = 0.622*mair1_max*Pg1*RH1/(P1-Pg1*RH1);           % mass flow
of h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
max_smair =(mair1_max/den_air + mvap1_max/den_vap)*1000;    % mass flow
of dry air and vapor(standard l/s)
if max_smair<min_smair                                      % makes
sure maximum is not less than minimum
    max_smair = min_smair;
end
PR_opmax = -3E-06*max_smair^3 + 0.0007*max_smair^2 - 0.0064*max_smair +
1.0382;   % max operating press. for turbocompressor--60 kW

% fuel cell current loop
index = 0;                                                  % place-
keeping variable
for current = Amin:Astep:Amax
    index = index + 1;

% highest efficiency pressure and mass flow lines
%-----------------------------------------------
    %   PR_stack = [1.015 1.015 1.015 1.015 1.0165 1.0215 1.0223 1.044
1.0459 1.0479 1.0499 1.0519 1.0538 1.0558 1.0578 1.0598 1.0638 1.0705
1.0724 1.0742 1.081 1.0828 1.0853 1.0878 1.1153 1.1187 1.1323 1.1364
```

```
1.1263 1.1312 1.1318 1.1228 1.1245 1.1466 1.1303 1.1632 1.172 1.1863
1.2066 1.2263 1.2421 1.2537 1.2657 1.2834 1.2966 1.3062 1.3256 1.35
1.317 1.3877 1.3455 1.3583 1.3773 1.3964 1.4161 1.4316 1.4578 1.4847
1.5222 1.5364 1.5558 1.5808 1.6066 1.6382 1.6657 1.6941 1.7232 1.7489
1.77 1.7921 1.8155 1.8493 1.8741 1.9047 1.9413 1.9738 2.0174 2.0477
2.0784 2.1058 2.1399 2.0313 2.0176 2.0246 2.0469 2.066 2.0912 2.1169
2.1492 2.192 2.2555 2.2617 2.2691 2.2802 2.0975];    %without expander
    %   PR_stack = [1.015 1.015 1.015 1.0165 1.0215 1.0215 1.0223 1.044
1.0459 1.0528 1.0647 1.0765 1.0982 1.1101 1.1367 1.1535 1.1773 1.1939
1.2007 1.2814 1.3178 1.3492 1.3765 1.4529 1.4607 1.5085 1.5172 1.5213
1.5408 1.5556 1.5759 1.5916 1.6426 1.6598 1.6781 1.6961 1.7098 1.7291
1.7445 1.7641 1.7652 1.7768 1.7937 1.8114 1.8197 1.8391 1.8536 1.8681
1.8845 1.8861 1.8932 1.906 1.9201 1.9589 1.9688 1.9794 1.9907 1.9979
2.0157 2.0546 2.0887 2.1088 2.1198 2.1267 2.1395 2.148 2.1722 2.2078
2.2142 2.2016 2.1906 2.1997 2.2343 2.2748 2.3114 2.3489 2.3875 2.4276
2.4682 2.5104 2.5495 2.5544 2.5456 2.5328 2.5206 2.5397 2.5698 2.5906
2.6131 2.6361 2.7983 2.8588 2.9205 2.9908 3.0302];  %with expander
    %   smair_stack = [4.3416 4.3416 4.5594 5.2115 5.4644 6.3173 6.3765
10.6081 10.7169 10.8288 10.9343 11.7345 11.8874 12.3403 12.5931 12.846
13.5989 14.4518 15.0046 15.2575 16.3104 16.5633 16.9161 17.269 17.5219
17.9748 19.0277 19.9805 20.5334 21.3863 21.6392 22.892 23.667 23.0501
24.5368 23.9275 24.3221 25.0208 25.724 26.4316 27.1438 27.8609 28.583
29.3103 30.0429 30.7809 31.5247 32.2742 33.0297 33.7914 34.5594 35.3339
36.1151 36.9031 37.698 38.5001 39.3095 40.1264 40.9509 41.7832 42.6235
43.472 44.3287 45.194 46.0678 46.9505 47.8421 48.7429 49.6529 50.5725
51.5016 52.4406 53.3895 54.3485 55.3179 56.2977 57.2881 58.2893 59.3014
60.3247 61.3593 62.4053 63.4629 64.5323 65.6137 66.7072 67.8129 68.9311
70.0619 71.2055 72.362 73.5317 74.7146 75.9109 77.1209]; %without
expander
    %   smair_stack = [4.3416 4.3416 4.3416 5.0115 6.1556 6.3173 6.3765
10.6081 10.7169 11.0975 11.7499 12.4024 13.5985 14.251 15.592 16.3209
17.3542 18.0745 18.3704 21.3633 22.4501 23.3866 24.1994 26.2409 26.43
27.5881 27.7987 27.8992 28.3706 28.7295 29.2225 29.6029 30.7285 31.0679
31.4287 31.7857 32.0568 32.4372 32.7415 33.1297 33.1512 33.3794 33.7149
34.064 34.2279 34.6122 34.8989 35.1857 35.5094 35.539 35.6723 35.9128
36.1756 36.9048 37.698 38.5001 39.3095 40.1264 40.9509 41.7832 42.6235
43.472 44.3287 45.194 46.0678 46.9505 47.8421 48.7429 49.6529 50.5725
51.5016 52.4406 53.3895 54.3485 55.3179 56.2977 57.2881 58.2893 59.3014
60.3247 61.3593 62.4053 63.4629 64.5323 65.6137 66.7072 67.8129 68.9311
70.0619 71.2055 72.362 73.5317 74.7146 75.9109 77.1209]; %with expander
    %   smair = smair_stack(current/5);                          %
standard mass flow rate of air (slps
    %   PR = PR_stack(current/5);                                %
compressor pressure ratio
    %   hum_ratio = 0.622*Pg1*RH1/(P1-Pg1*RH1);                  %
ambient humidity ratio
    %   mair1 =smair/(1000/den_air + 1000*hum_ratio/den_vap);  % mass
flow rate of air (kg/s)
    %   mvap1 = (smair/1000 - mair1/den_air)*den_vap;          % mass
flow rate of vapor (kg/s)
    %   P2 = PR*P1;                                              %
compressor pressure (atm)
    %   mH2 = num_cell*current*MH2/(96487*2*1000);             % kg
H2/s (Fuel Cell Handbook, 8-2)
% ------------------------------------------------

% curve fit of highest efficiency pressure lines
```

```
% ------------------------------------------------
    SR = 3.381e-6*(current^2)-6e-4*current+2.009;    % operating line
for air stoichiometric ratio--60 kW
                                                     % equation taken
from a curve fit for the following:
                                                     % current:  300
210 120 30
                                                     % SR:       2.5
2.2 2.0 2.0
    if SR<SR_min                                     % makes sure
stoichiometric ratio stays above two and below the maximum
        SR = SR_min;
    elseif SR>SR_max
        SR = SR_max
    end

    % fuel cell fuel and air consumption values
    mH2 = num_cell*current*MH2/(96487*2*1000);        % kg H2/s (Fuel
Cell Handbook, 8-2)
    mO2 = 0.5*(mH2*MO2)/MH2;                          % kg O2/s
(Shapiro, 558, equ. 12.1)
    mair_req = (mO2*Ma)/(MO2*0.21);                   % kg dry air/s
required for reaction (ibid, mole fraction of O2 from Shapiro 559)
    mair1 = SR*mair_req;                              % actual kg dry
air/s flowing through compressor
    mvap1 = 0.622*mair1*Pg1*RH1/(P1-Pg1*RH1);         % mass flow of
h2o (kg/s) entering compressor (Shapiro, 582, equ 12.43)
    smair =(mair1/den_air + mvap1/den_vap)*1000;      % mass flow of
dry air and vapor(standard l/s)
    if smair<min_smair                                % makes sure
standard mass flow does not go below minimum
        smairold = smair;
        smair = min_smair;
        mair1 = smair/smairold*mair1;
        mvap1 = smair/smairold*mvap1;
    end
    if expnd
        PR = -555.56*(current/1000)^5 + 830.58*(current/1000)^4 -
424.16*(current/1000)^3 + 87.855*(current/1000)^2 -
2.6943*(current/1000) + 1.0249;  %pressure line with expander
    else
        PR = 5.6318*(current/1000)^2 - 0.0264*(current/1000) + 1.0152;
%pressure line without expander
    end
    P2 = PR * P1;
%------------------------------------------------

    % initializing variables
    T3 = T3_init;
    T4 = T4_init;
    T_h2o = T_h2o_init;

    p_continue = 0;
    while ~p_continue               % this loop checks to make sure
the pressure of the compressor at least matches the pressure drop
across the system
```

```
        % approximate fuel cell calculations to make guess at Q_stack
and m_cool
        pO2 = P2*0.21;                  % partial pressure of oxygen
in dry air
        J = current/area_cell;          % current density
        V_guess = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4)*J+0.14*log10(pO2)+(T4-333)*0.00041*log10(10^3*J);
      %revised Honeywell curve with low temp. effects (5/2/01)
        Pwr_f_guess = V_guess*num_cell*current;  % stack power (W)
        Q_guess = mH2*H2_HHV-Pwr_f_guess;% initial fuel cell heat
rejection value (W)

        % use of a 50gpm pump with max power of 1.2 bHP (CD-100 from
Pricepump)
        m_cool = Q_guess/(10*cp_h2o);    % coolant flow giving a 10°C
temp diff across stack
        Pwr_pump = maxpump/maxpumpflow*m_cool;  % linear power with max
determined by Price Pump website

        if Pwr_pump>maxpump             % checks to see if the pump
power exceeds the maximum
            Pwr_pump = maxpump;          % if the power does exceed
the max, it is set to the maximum power (W)
            m_cool = maxpumpflow;
            'stack will overheat -- reached max of coolant pump'
        elseif m_cool<minpumpflow
            m_cool = minpumpflow;
            Pwr_pump = maxpump/maxpumpflow*m_cool;          % linear
power with max determined by Price Pump website
            T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;
      %temperature achievable in humidifier air stream (assumes 5° min.
difference btwn coolant and air)
            T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);         %stack exit
coolant temp
        end
%-------------------------------------------------------------------------
% Compressor Section
%-------------------------------------------------------------------------

        % normalize smair based on itself to place in range of
compressor
        smair_norm = smair/max_smair;

        % normalizing values for compressor data
        max_flow = 76;  %normalized for high efficiency (SLPS)
        max_PR = 3.51;
        max_eff = 0.796;
        max_speed = 105000;
        min_speed = 10000;

        PR_norm = PR/max_PR;        %scaled based on compressor maximum

        % first check to make sure the input point is below surge line
        surge = 0;
        if smair_norm <= s20K(1,1)
            P_surge = s20K(2,1) - (s20K(2,1)-s10K(2,1))/(s20K(1,1)-
s10K(1,1))*(s20K(1,1)-smair_norm);
```

```
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s20K(1,1)) & (smair_norm<=s30K(1,1))
            P_surge = s30K(2,1) - (s30K(2,1)-s20K(2,1))/(s30K(1,1)-
s20K(1,1))*(s30K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s30K(1,1)) & (smair_norm<=s40K(1,1))
            P_surge = s40K(2,1) - (s40K(2,1)-s30K(2,1))/(s40K(1,1)-
s30K(1,1))*(s40K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s40K(1,1)) & (smair_norm<=s50K(1,1))
            P_surge = s50K(2,1) - (s50K(2,1)-s40K(2,1))/(s50K(1,1)-
s40K(1,1))*(s50K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s50K(1,1)) & (smair_norm<=s60K(1,1))
            P_surge = s60K(2,1) - (s60K(2,1)-s50K(2,1))/(s60K(1,1)-
s50K(1,1))*(s60K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s60K(1,1)) & (smair_norm<=s70K(1,1))
            P_surge = s70K(2,1) - (s70K(2,1)-s60K(2,1))/(s70K(1,1)-
s60K(1,1))*(s70K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s70K(1,1)) & (smair_norm<=s80K(1,1))
            P_surge = s80K(2,1) - (s80K(2,1)-s70K(2,1))/(s80K(1,1)-
s70K(1,1))*(s80K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s80K(1,1)) & (smair_norm<=s90K(1,1))
            P_surge = s90K(2,1) - (s90K(2,1)-s80K(2,1))/(s90K(1,1)-
s80K(1,1))*(s90K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif (smair_norm>s90K(1,1)) & (smair_norm<=s100K(1,1))
            P_surge = s100K(2,1) - (s100K(2,1)-s90K(2,1))/(s100K(1,1)-
s90K(1,1))*(s100K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
        elseif smair_norm>s100K(1,1)
            P_surge = s105K(2,1) - (s105K(2,1)-s100K(2,1))/(s105K(1,1)-
s100K(1,1))*(s105K(1,1)-smair_norm);
            if PR_norm > P_surge
                surge = 1;
            end
```

```
        end

        if surge  % this will place the operating point directly on the
surge line at the requested pressure
            smairold = smair_norm;
            if PR_norm <= s20K(2,1)
                smair_norm = s20K(1,1) - (s20K(1,1)-
s10K(1,1))/(s20K(2,1)-s10K(2,1))*(s20K(2,1)-PR_norm);
            elseif (PR_norm>s20K(2,1)) & (PR_norm<=s30K(2,1))
                smair_norm = s30K(1,1) - (s30K(1,1)-
s20K(1,1))/(s30K(2,1)-s20K(2,1))*(s30K(2,1)-PR_norm);
            elseif (PR_norm>s30K(2,1)) & (PR_norm<=s40K(2,1))
                smair_norm = s40K(1,1) - (s40K(1,1)-
s30K(1,1))/(s40K(2,1)-s30K(2,1))*(s40K(2,1)-PR_norm);
            elseif (PR_norm>s40K(2,1)) & (PR_norm<=s50K(2,1))
                smair_norm = s50K(1,1) - (s50K(1,1)-
s40K(1,1))/(s50K(2,1)-s40K(2,1))*(s50K(2,1)-PR_norm);
            elseif (PR_norm>s50K(2,1)) & (PR_norm<=s60K(2,1))
                smair_norm = s60K(1,1) - (s60K(1,1)-
s50K(1,1))/(s60K(2,1)-s50K(2,1))*(s60K(2,1)-PR_norm);
            elseif (PR_norm>s60K(2,1)) & (PR_norm<=s70K(2,1))
                smair_norm = s70K(1,1) - (s70K(1,1)-
s60K(1,1))/(s70K(2,1)-s60K(2,1))*(s70K(2,1)-PR_norm);
            elseif (PR_norm>s70K(2,1)) & (PR_norm<=s80K(2,1))
                smair_norm = s80K(1,1) - (s80K(1,1)-
s70K(1,1))/(s80K(2,1)-s70K(2,1))*(s80K(2,1)-PR_norm);
            elseif (PR_norm>s80K(2,1)) & (PR_norm<=s90K(2,1))
                smair_norm = s90K(1,1) - (s90K(1,1)-
s80K(1,1))/(s90K(2,1)-s80K(2,1))*(s90K(2,1)-PR_norm);
            elseif (PR_norm>s90K(2,1)) & (PR_norm<=s100K(2,1))
                smair_norm = s100K(1,1) - (s100K(1,1)-
s90K(1,1))/(s100K(2,1)-s90K(2,1))*(s100K(2,1)-PR_norm);
            elseif PR_norm>s100K(2,1)
                smair_norm = s105K(1,1) - (s105K(1,1)-
s100K(1,1))/(s105K(2,1)-s100K(2,1))*(s105K(2,1)-PR_norm);
            end
            mair1 = smair_norm/smairold*mair1;
            mvap1 = smair_norm/smairold*mvap1;
            smair = smair_norm*max_smair;
            if PR_norm>0.997
                PR_norm = 0.997;
            end
        end

        % flags
        check = 0;        % indicates if input point is within smair
range of speed line
        done = 0;         % indicates if top and bottom speed lines
have been found
        range = 0;        % indicates if the input point was ever in a
smair range
        special1 = 0;     % indicates a special case (operating point
is just below lowest map line)
        special2 = 0;     % indicates a special case (operating point
is left of lowest map line)
```

```
        % This section determines where the input operating point is on
the compressor map

        if and(smair_norm<=max(s10K(1,:)),smair_norm>=min(s10K(1,:)))
% if smair is between max and min of s10K flow values
            x = 1;
            while ~check
                A1 = s10K(1,x);
                A2 = s10K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)   % if smair is
between two selected flows in s10K
                    check = 1;                           % allow to
get out of while loop
                    range = 1;
                    P = s10K(2,x+1) - (s10K(2,x+1) - s10K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s10K at smair
                    if P>=PR_norm
                        done = 1;
                        special1 = 1;
                    end
                else
                    x = x + 1;
                end
            end
        elseif smair_norm<min(s10K(1,:))
            x = 1;
            done = 1;
            special2 = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s20K(1,:)),smair_norm>=min(s20K(1,:))))  % if smair
is between max and min of s20K flow values
            x = 1;
            while ~check
                A1 = s20K(1,x);
                A2 = s20K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s20K
                    check = 1;                           % allow to get
out of while loop
                    range = 1;
                    P = s20K(2,x+1) - (s20K(2,x+1) - s20K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s20K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s20K;
                        bot = s10K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s20K;
            bot = s10K;
```

```
            done = 1;
        elseif and(smair_norm<min(s20K(1,:)), ~done)
            P = s20K(2,2) - (s20K(2,2) - s20K(2,1))/(s20K(1,2) -
s20K(1,1))*(s20K(1,2) - smair_norm);  % interp. of pressure on s20K at
smair
            done = 1;
            top = s20K;
            bot = s10K;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s30K(1,:)),smair_norm>=min(s30K(1,:))))  % if smair
is between max and min of s30K flow values
            x = 1;
            while ~check
                A1 = s30K(1,x);
                A2 = s30K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s30K
                    check = 1;                          % allow to get
out of while loop
                    range = 1;
                    P = s30K(2,x+1) - (s30K(2,x+1) - s30K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s30K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s30K;
                        bot = s20K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s30K;
            bot = s20K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s40K(1,:)),smair_norm>=min(s40K(1,:))))  % if smair
is between max and min of s40K flow values
            x = 1;
            while ~check
                A1 = s40K(1,x);
                A2 = s40K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s40K
                    check = 1;                          % allow to get
out of while loop
                    range = 1;
                    P = s40K(2,x+1) - (s40K(2,x+1) - s40K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s40K at smair
                    if P>=PR_norm
                        done = 1;
```

```
                    top = s40K;
                    bot = s30K;
                end
            else
                x = x + 1;
            end
        end
    elseif and(range, ~done)
        top = s40K;
        bot = s30K;
        done = 1;
    end

    check = 0;
    if and(~done,
and(smair_norm<=max(s50K(1,:)),smair_norm>=min(s50K(1,:))))  % if smair
is between max and min of s50K flow values
        x = 1;
        while ~check
            A1 = s50K(1,x);
            A2 = s50K(1,x+1);
            if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s50K
                check = 1;                          % allow to get
out of while loop
                range = 1;
                P = s50K(2,x+1) - (s50K(2,x+1) - s50K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s50K at smair
                if P>=PR_norm
                    done = 1;
                    top = s50K;
                    bot = s40K;
                end
            else
                x = x + 1;
            end
        end
    elseif and(range, ~done)
        top = s50K;
        bot = s40K;
        done = 1;
    end

    check = 0;
    if and(~done,
and(smair_norm<=max(s60K(1,:)),smair_norm>=min(s60K(1,:))))  % if smair
is between max and min of s60K flow values
        x = 1;
        while ~check
            A1 = s60K(1,x);
            A2 = s60K(1,x+1);
            if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s60K
                check = 1;                          % allow to get
out of while loop
                range = 1;
```

```
                    P = s60K(2,x+1) - (s60K(2,x+1) - s60K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s60K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s60K;
                        bot = s50K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s60K;
            bot = s50K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s70K(1,:)),smair_norm>=min(s70K(1,:))))  % if smair
is between max and min of s70K flow values
            x = 1;
            while ~check
                A1 = s70K(1,x);
                A2 = s70K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s70K
                    check = 1;                           % allow to get
out of while loop
                    range = 1;
                    P = s70K(2,x+1) - (s70K(2,x+1) - s70K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s70K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s70K;
                        bot = s60K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s70K;
            bot = s60K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s80K(1,:)),smair_norm>=min(s80K(1,:))))  % if smair
is between max and min of s80K flow values
            x = 1;
            while ~check
                A1 = s80K(1,x);
                A2 = s80K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s80K
```

```
                    check = 1;                              % allow to get
out of while loop
                    range = 1;
                    P = s80K(2,x+1) - (s80K(2,x+1) - s80K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s80K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s80K;
                        bot = s70K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s80K;
            bot = s70K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s90K(1,:)),smair_norm>=min(s90K(1,:))))  % if smair
is between max and min of s90K flow values
            x = 1;
            while ~check
                A1 = s90K(1,x);
                A2 = s90K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s90K
                    check = 1;                               % allow to get
out of while loop
                    range = 1;
                    P = s90K(2,x+1) - (s90K(2,x+1) - s90K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % interp. of pressure on s90K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s90K;
                        bot = s80K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s90K;
            bot = s80K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s100K(1,:)),smair_norm>=min(s100K(1,:))))  % if
smair is between max and min of s100K flow values
            x = 1;
            while ~check
                A1 = s100K(1,x);
```

```
                A2 = s100K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s100K
                    check = 1;                          % allow to get
out of while loop
                    range = 1;
                    P = s100K(2,x+1) - (s100K(2,x+1) - s100K(2,x))/(A2
- A1)*(A2 - smair_norm);  % interp. of pressure on s100K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s100K;
                        bot = s90K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s100K;
            bot = s90K;
            done = 1;
        end

        check = 0;
        if and(~done,
and(smair_norm<=max(s105K(1,:)),smair_norm>=min(s105K(1,:))))  % if
smair is between max and min of s105K flow values
            x = 1;
            while ~check       % finds points that the input point lies
between
                A1 = s105K(1,x);
                A2 = s105K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s105K
                    check = 1;                          % allow to get
out of while loop
                    P = s105K(2,x+1) - (s105K(2,x+1) - s105K(2,x))/(A2
- A1)*(A2 - smair_norm);  % interp. of pressure on s105K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s105K;
                        bot = s100K;
                    end
                else
                    x = x + 1;
                end
            end
        elseif and(range, ~done)
            top = s105K;
            bot = s100K;
            done = 1;
        end

        while ~done                % op. point is beyond highest speed
line on map, so pressure will be lowered until acceptable
            PR_norm = PR_norm-0.01;  % lowers pressure
            P2 = PR_norm*max_PR*P1;
```

```
            check = 0;
            x = 1;
            while ~check            % finds points that the input
point lies between
                A1 = s105K(1,x);
                A2 = s105K(1,x+1);
                if and(smair_norm<=A2, smair_norm>=A1)  % if smair is
between two selected flows in s105K
                    check = 1;                          % allow to get
out of while loop
                    P = s105K(2,x+1) - (s105K(2,x+1) - s105K(2,x))/(A2
- A1)*(A2 - smair_norm);  % interp. of pressure on s105K at smair
                    if P>=PR_norm
                        done = 1;
                        top = s105K;
                        bot = s100K;
                    end
                else
                    x = x + 1;
                end
            end
        end


        % -- This section defines two special cases and gives the
speed, efficiency, and changes in operating point

        if special1   % this is the special case if the input point is
below the 10k speed line but in smair range
            comp_speed = min_speed;  % compressor speed is set to at
least minimum speed
            PR_norm = s10K(2,x+1) - (s10K(2,x+1) - s10K(2,x))/(A2 -
A1)*(A2 - smair_norm);  % pressure is raised to pressure on 10k map
line at smair
            P2 = PR_norm*max_PR*P1;  % adjustment to P2
            eff_overall = (s10K(3,x+1) - (s10K(3,x+1) - s10K(3,x))/(A2
- A1)*(A2 - smair_norm))*max_eff;  % interp. of efficiency on s10K at
smair

        elseif special2  % this is the special case if the input point
is below the 10k speed line and less than smair range
            comp_speed = min_speed;                % compressor speed
is set to at least minimum speed
            smair_bak = smair;                     % storing old smair
value
            smair = s10K(1,1)*max_smair;           % smair is set to
lowest compressor map smair
            mair1 = smair/smair_bak*mair1;         % mair1 is updated
            mvap1 = smair/smair_bak*mvap1;         % mvap1 is updated
            P2 = s10K(2,1)*max_PR*P1;              % pressure is set to
pressure at new smair
            eff_overall = s10K(3,1)*max_eff;       % efficiency is set
to efficiency at new smair

        else
            % This section finds the two closest points to the input
point on the top speed line
```

```
            % (point1 = closest, point2 = 2nd closest)

            dist1 = sqrt((PR_norm - top(2,1))^2 + (smair_norm -
top(1,1))^2);
            dist2 = sqrt((PR_norm - top(2,2))^2 + (smair_norm -
top(1,2))^2);
            point1 = 1;
            point2 = 2;
            if dist2<dist1           % sets shortest distance at dist1
                dist3 = dist1;
                dist1 = dist2;
                dist2 = dist3;
                point1 = 2;
                point2 = 1;
            end
            for z=3:length(top)
                dist3 = sqrt((PR_norm - top(2,z))^2 + (smair_norm -
top(1,z))^2);
                if dist3<dist2
                    if dist3<dist1
                        dist2 = dist1;
                        point2 = point1;
                        dist1 = dist3;
                        point1 = z;
                    else
                        dist2 = dist3;
                        point2 = z;
                    end
                end
            end

            if PR_norm==P      % this is the special case if the input
point is directly on one of the speed lines
                comp_speed = top(4,1)*max_speed;
                % find percent distance between points, and translate
that into efficiency
                sec1 = sqrt((top(1,point1)-smair_norm)^2 +
(top(2,point1)-PR_norm)^2);
                sec2 = sqrt((top(1,point2)-smair_norm)^2 +
(top(2,point2)-PR_norm)^2);
                pcnt = sec1/(sec1 + sec2);    % percent of distance
that input point is from point1 to point2 on top speed line
                eff_overall = (top(3,point1) + pcnt*(top(3,point2) -
top(3,point1)))*max_eff;
            else
                % This section finds the point on the top speed line
that, if connected to the input point, would make a perpendicular line
                % (x = smair_norm, y = PR_norm)

                check = 0;
                u = ((smair_norm-top(1,point1))*(top(1,point2)-
top(1,point1))+(PR_norm-top(2,point1))*(top(2,point2)-
top(2,point1)))/((sqrt((top(1,point2)-top(1,point1))^2+(top(2,point2)-
top(2,point1))^2))^2);
                smairtop = top(1,point1) + u*(top(1,point2) -
top(1,point1));
```

```
                PRtop = top(2,point1) + u*(top(2,point2) -
top(2,point1));

                % This section finds the point on the bottom speed line
through which the perpendicular to the top speed line goes
                % (1 = input, 2 = point for perpendicular, 3 and 4 =
points on lower line)
                % first, special end cases
                % beginning of bottom speed line
                ua = ((bot(1,2)-bot(1,1))*(PR_norm - bot(2,1)) -
(bot(2,2)-bot(2,1))*(smair_norm-bot(1,1)))/((bot(2,2)-
bot(2,1))*(smairtop-smair_norm) - (bot(1,2)-bot(1,1))*(PRtop -
PR_norm));
                smairbot = smair_norm + ua*(smairtop - smair_norm);
                PRbot = PR_norm + ua*(PRtop - PR_norm);
                if smairbot<=bot(1,2)
                    check = 1;
                end
                % middle of bottom speed line
                x = 1;
                while ~check & x<8
                    x = x + 1;
                    ua = ((bot(1,x+1)-bot(1,x))*(PR_norm - bot(2,x)) -
(bot(2,x+1)-bot(2,x))*(smair_norm-bot(1,x)))/((bot(2,x+1)-
bot(2,x))*(smairtop-smair_norm) - (bot(1,x+1)-bot(1,x))*(PRtop -
PR_norm));
                    smairbot = smair_norm + ua*(smairtop - smair_norm);
                    PRbot = PR_norm + ua*(PRtop - PR_norm);
                    if and(smairbot>=bot(1,x),smairbot<=bot(1,x+1))
                        check = 1;
                    end
                end
                % end of bottom speed line
                if ~check
                    ua = (((bot(1,9))-bot(1,8))*(PR_norm - bot(2,8)) -
(bot(2,9)-bot(2,8))*(smair_norm-bot(1,8)))/((bot(2,9)-
bot(2,8))*(smairtop-smair_norm) - (bot(1,9)-bot(1,8))*(PRtop -
PR_norm));
                    smairbot = smair_norm + ua*(smairtop - smair_norm);
                    PRbot = PR_norm + ua*(PRtop - PR_norm);
                    if smairbot>=bot(1,8)
                        check = 1;
                    end
                end
                % This section finds the compressor speed

                d1 = sqrt((smairtop-smair_norm)^2 + (PRtop -
PR_norm)^2);
                d2 = sqrt((smairbot-smair_norm)^2 + (PRbot -
PR_norm)^2);
                pcnt = d1/(d1 + d2);
                comp_speed = (top(4,1) + pcnt*(bot(4,1) -
top(4,1)))*max_speed;

                % This section develops the efficiency curve for the
given speed and finds the efficiency on that curve
```

```
                eff1 = top(3,1) + pcnt*(bot(3,1) - top(3,1));
                smair1 = top(1,1) + pcnt*(bot(1,1) - top(1,1));
                check = 0;
                x = 1;
                if top(4,1)~=1
                    while and(~check, x<=8)
                        x = x + 1;
                        eff2 = eff1;
                        smair2 = smair1;
                        eff1 = top(3,x) + pcnt*(bot(3,x) - top(3,x));
                        smair1 = top(1,x) + pcnt*(bot(1,x) - top(1,x));
                        if smair_norm<=smair1
                            % now find percent difference of smair
between two points on efficiency curve, and use that to find eff.
                            check = 1;
                            pcnt2 = (smair2 - smair_norm)/(smair2 -
smair1);
                            eff_overall = (eff2 + pcnt2*(eff1 -
eff2))*max_eff;
                        end
                    end
                else
                    while and(~check, x<=6)  % for the special case
where s105K is used as "top" (because of its fewer data points)
                        x = x + 1;
                        eff2 = eff1;
                        smair2 = smair1;
                        eff1 = top(3,x) + pcnt*(bot(3,x) - top(3,x));
                        smair1 = top(1,x) + pcnt*(bot(1,x) - top(1,x));
                        if smair_norm<=smair1
                            % now find percent difference of smair
between two points on efficiency curve, and use that to find eff.
                            check = 1;
                            pcnt2 = (smair2 - smair_norm)/(smair2 -
smair1);
                            eff_overall = (eff2 + pcnt2*(eff1 -
eff2))*max_eff;
                        end
                    end
                end
                if ~check  % takes into account if input point is
beyond smair range of other points on efficiency curve
                    pcnt2 = (smair2 - smair_norm)/(smair2 - smair1);
                    eff_overall = (eff2 + pcnt2*(eff1 - eff2))*max_eff;
                end
            end
        end
        % stagnation temps and pressures are not used due to negligible
difference to static values
        T2 = T1*((P2/P1)^((k-1)/k) - 1)/eff_overall + T1;  %
temperature of outlet
        temp_rise = T2 - T1;
        Pwr_c = ((mair1/Ma + mvap1/Mh2o)*8314*T1*k/(k-1)*((P2/P1)^((k-
1)/k)-1))/(eff_overall*nm);   % compressor power
        vol_eff = 0;  % undefined for turbomachine

%-----------------------------------------------------------------------
```

```
% Humidifier/Heat Exchanger Section
%-----------------------------------------------------------------------

        P3 = P2;  % no pressure drop over humidifier
        cont = 0;
        while ~cont       % this loop checks to make sure the
humidifier doesn't use more heat than is given by stack
            RH3 = RH3_init;   % initializing RH3
            %hydrogen humidification
            Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-273)^2+1.44E-7*(T4-
273)^3-2.18);  % saturation pressure in fuel cell inlet for hydrogen (T
in K and P_sat in atm)
            mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);  %
water vapor in H2 (assumes initially dry H2 and can make it to fc temp)
            %air humidification
            Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-273)^2+1.44E-7*(T3-
273)^3-2.18);  % water sat. pressure for H2
            if T3==T4
                mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);  % vapor mass
going into fuel cell (Shapiro, 582, equ 12.43)
                mh2o_addh = mvap3 - mvap1;                % net water
added in humidifier
                mtot3 = mair1 + mvap3;                    % total mass
flow into fuel cell
            else  % effects of coolant loop minimum
                mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);  % vapor mass
needed to humidify air to correct RH when warmed to fuel cell temp
(Shapiro, 582, equ 12.43)
                mh2o_addh = mvap3 - mvap1;                % net water
added in humidifier
                mtot3 = mair1 + mvap3;                    % total mass
flow into fuel cell
                RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);% relative
humidity of air at humidifier exit
                if RH3>1
                    'lower than requested fuel cell inlet air RH'
                    RH3 = 1;
                    mvap3 = 0.622*mair1*Pg3/(P3-Pg3);       % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;              % net water
added in humidifier
                    mtot3 = mair1 + mvap3;
                    Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);
                    RH3_in = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);
% relative humidity of air after warming up in fuel cell
                end
            end
            Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-T2) +
mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o);  %total heat needed to
warm mass flow into fuel cell

            while Q_needed<Q_guess & T4<T4_init        % raises the
temp of stack if heat is available
                RH3 = RH3_init;  % initializing RH3
                T4 = T4 + .1;
                if T4>T4_init
```

```
                T4 = T4_init;
            end
            T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;% temperature
achievable in humidifier air stream (assumes 5° min. difference btwn
coolant and air)
            T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o); % water temp
same as stack coolant exit temp
            Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-273)^2+1.44E-
7*(T3-273)^3-2.18);  % saturation pressure in fuel cell inlet (T in K
and P_sat in atm)
            Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-273)^2+1.44E-
7*(T4-273)^3-2.18);  % saturation pressure in fuel cell inlet (T in K
and P_sat in atm)
            mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);  %
water vapor in H2 (assumes initially dry H2 and can make it to fc temp)
            if T3==T4
                mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);  % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                mh2o_addh = mvap3 - mvap1;                % net
water added in humidifier
                mtot3 = mair1 + mvap3;
            else  % effects of coolant loop minimum
                mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);  % vapor
mass needed to humidify air to correct RH when warmed to fuel cell temp
(Shapiro, 582, equ 12.43)
                mh2o_addh = mvap3 - mvap1;                % net
water added in humidifier
                mtot3 = mair1 + mvap3;
                RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);%
relative humidity of air at humidifier exit
                if RH3>1
                    'lower than requested fuel cell inlet air RH'
                    RH3 = 1;
                    mvap3 = 0.622*mair1*Pg3/(P3-Pg3);       % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;              % net
water added in humidifier
                    mtot3 = mair1 + mvap3;
                    Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);
                    RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  % relative humidity of air
after warming up in fuel cell
                end
            end
            Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-T2)
+ mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o);
        end

        temp_mod = 0;           % flag to show temperature
modification
        while Q_needed>Q_guess  % lowers stack temperature to allow
humidifier to heat to temperature
            temp_mod = 1;
            RH3 = RH3_init;      % initializing RH3
            T4 = T4 - .1;        % incremental decrease in stack
ave temperature
```

```
                T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5;  % temperature
achievable in humidifier air stream (assumes 5° min. difference btwn
coolant and air)
                T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);   % water temp
same as stack coolant exit temp
                Pg3 = 10^(2.95E-2*(T3-273)-9.18E-5*(T3-273)^2+1.44E-
7*(T3-273)^3-2.18);  % saturation pressure in fuel cell inlet (T in K
and P_sat in atm)
                Pg4 = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-273)^2+1.44E-
7*(T4-273)^3-2.18);  % saturation pressure in fuel cell inlet (T in K
and P_sat in atm)
                mvap_H2 = (Mh2o/MH2)*mH2*Pg4*RH_H2/(P3 - Pg4*RH_H2);  %
water vapor in H2 (assumes initially dry H2 and can make it to fc temp)
                if T3==T4
                    mvap3 = 0.622*mair1*Pg3*RH3/(P3-Pg3*RH3);  % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                % net
water added in humidifier
                    mtot3 = mair1 + mvap3;
                else  % effects of coolant loop minimum
                    mvap3 = 0.622*mair1*Pg4*RH3/(P3-Pg4*RH3);  % vapor
mass needed to humidify air to correct RH when warmed to fuel cell temp
(Shapiro, 582, equ 12.43)
                    mh2o_addh = mvap3 - mvap1;                % net
water added in humidifier
                    mtot3 = mair1 + mvap3;
                    RH3 = (mvap3*P3)/((0.622*mair1+mvap3)*Pg3);%
relative humidity of air at humidifier exit
                    if RH3>1
                        'lower than requested fuel cell inlet air RH'
                        RH3 = 1;
                        mvap3 = 0.622*mair1*Pg3/(P3-Pg3);       % vapor
mass going into fuel cell (Shapiro, 582, equ 12.43)
                        mh2o_addh = mvap3 - mvap1;              % net
water added in humidifier
                        mtot3 = mair1 + mvap3;
                        Pg3_in = 10^(2.95E-2*(T4-273)-9.18E-5*(T4-
273)^2+1.44E-7*(T4-273)^3-2.18);
                        RH3_in =
(mvap3*P3)/((0.622*mair1+mvap3)*Pg3_in);  % relative humidity of air
after warming up in fuel cell
                    end
                end
                Q_needed = mair1*cp_air*(T3-T2) + mvap1*cp_vap*(T3-T2)
+ mh2o_addh*e_vap + mh2o_addh*cp_vap*(T3-T_h2o);
                %Q_needed shows amount of heat needed to warm air
stream to T3
            end

            Q_hum = Q_needed;


%-------------------------------------------------------------------------
% Fuel Cell Section
%-------------------------------------------------------------------------
```

```
            Pdrop = resist*(mair1*den_air + (mtot3-mair1)*den_vap);  %
pressure loss across system
            P4 = P3 - Pdrop;                                         %
pressure at exit of fuel cell
            PO2_3 = (P3 - RH3*Pg3)*0.21;                             %
partial pressure of oxygen at fuel cell inlet
            mO2_used = (num_cell*current*MO2)/(96487*4000);          %
mass flow of oxygen consumed (kg/s)
            mair4 = mair1 - mO2_used;                                %
total mass flow of dry air at fuel cell exit
            O2_ratio4 = (mair1*0.21-mO2_used)/mair4;                 %
ratio of oxygen to dry air at fuel cell exit
            mvapmax = 0.622*mair1*Pg4/(P4-Pg4);                      %
total vapor at 100% RH at fuel cell outlet
            mvap_fc = mvapmax - mvap3;                               %
total vapor acquired by airstream from fuel cell water production
            mh2o_prod = mH2*Mh2o/MH2;                                %
stack water production (kg/s)
            h2o_used = 0;
            if mvap_fc>mh2o_prod
                h2o_used = 1;
                mvap_fc = mh2o_prod;  % if production of water is less
than total that the air can accept for 100% humidity, then air will
only take total production of water
                RH4 = ((mvap3+mvap_fc)*(P3-
Pdrop))/((0.622*mair1+mvap3+mvap_fc)*Pg4);  % relative humidity of air
at fuel cell outlet
            else
                RH4 = 1;
                h2o_used = 0;
            end
            PO2_4 = (P4 - RH4*Pg4)*O2_ratio4;  % partial pressure of
oxygen at fuel cell exit
            pO2 = (PO2_3 + PO2_4)/2;           % average partial
pressure of oxygen
            J = current/area_cell;             % current density
            V = 1.03-0.06*log10(1000*J)-(1.12-2.49e-
3*T4)*J+0.14*log10(pO2)+(T4-333)*0.00041*log10(10^3*J);
      %revised Honeywell curve with temp. effects (5/2/01)
            if V <0.5
                'data for this current level is bad'
            end

            Vstack = V * num_cell;             % stack voltage (V)
            Pwr_f = Vstack*current;            % stack power (W)
            mvap4 = mvap3 + mvap_fc;
            n_stack = Pwr_f/(mH2*H2_LHV);      % stack efficiency
            Q_stack = mH2*(1 - mvap_fc/mh2o_prod)*H2_HHV +
mH2*(mvap_fc/mh2o_prod)*H2_LHV - Pwr_f;               %stack heat
rejection (W)
            mh2o_net = mh2o_prod - mvap_fc;    % water recoverable as
liquid at fuel cell outlet

            if ((Q_guess/Q_stack)>1.01|(Q_guess/Q_stack)<.99)  %
compares actual stack heat rejection with guess
                if (Q_guess/Q_stack)<.99        % this only allows the
initialization of temp if there is more heat available from stack
```

```
                    T4 = T4+.1;
                    if T4 > T4_init
                        T4 = T4_init;
                    end
                    T3 = T4;
                    T_h2o = T4+5;
                end
                Q_guess = Q_stack;
                cont = 0;
                m_cool = Q_guess/(10*cp_h2o);   % coolant flow giving a
10°C temp diff across stack
                Pwr_pump = maxpump/maxpumpflow*m_cool;  % linear power
with max determined by Price Pump website
                if Pwr_pump>maxpump
                    Pwr_pump = maxpump;
                    m_cool = maxpumpflow;
                elseif m_cool<minpumpflow
                    m_cool = minpumpflow;
                    Pwr_pump = maxpump/maxpumpflow*m_cool;   % linear
power with max determined by Price Pump website
                    T_h2o = T4 + Q_guess/(2*m_cool*cp_h2o);  % stack
coolant exit temp
                    T3 = T4 + Q_guess/(2*m_cool*cp_h2o) - 5; % max
temperature achievable in humidifier air stream (assumes 5° min.
difference btwn coolant and air)
                end
            else
                cont = 1;
            end

        end   % while -- heat check (in humidifier)

        if (P2-Pdrop)<P1  % this loop checks if there is enough
pressure to overcome pressure drop
            p_continue = 0;
            P2 = P1 + Pdrop + .001;  % new pressure adds pressure drop
plus a small amount extra to ambient
            PR = P2/P1;
        else
            p_continue = 1;
        end
    end % while -- pressure check (before compressor)

%----------------------------------------------------------------------
% Radiator Section
%----------------------------------------------------------------------

    % the NTU method is used for the radiator calculations
    T5 = T4 - Q_hum/(m_cool*cp_h2o);  % temperature of coolant entering
radiator
    if T5 < T1
        current
        'Warning!  Coolant temp. entering radiator is lower than
ambient'
    end
    Q_rad = Q_stack - Q_hum;
```

```
    % air properties with fan on
    m_on = radair_on*den_air;
    C_on = m_on*cp_air;
    C_h2o = m_cool*cp_h2o;
    C1 = [C_on C_h2o];
    Cr_on = min(C1)/max(C1);
    NTU_on = UA_on/min(C1);
    eff_on = 1-exp((1/Cr_on)*NTU_on^.22*(exp(-Cr_on*NTU_on^.78)-1));
    Qmax_on = min(C1)*(T5-T1);
    Q_on = eff_on*Qmax_on;

    % air properties with fan off
    m_off = radair_off*den_air;
    C_off = m_off*cp_air;
    C_h2o = m_cool*cp_h2o;
    C2 = [C_off C_h2o];
    Cr_off = min(C2)/max(C2);
    NTU_off = UA_off/min(C2);
    eff_off = 1-exp((1/Cr_off)*NTU_off^.22*(exp(-Cr_off*NTU_off^.78)-
1));
    Qmax_off = min(C2)*(T5-T1);
    Q_off = eff_off*Qmax_off;

    bypass_pcnt = 0;
    fan_timeon = (Q_rad-Q_off)/(Q_on-Q_off);
    if fan_timeon < 0
        bypass_pcnt = 1 - Q_rad/Q_off;
        fan_timeon = 0;
    elseif fan_timeon > 1
        fan_timeon = 1;
    end
    avePwr_fan = (fan_timeon*maxfan)/nfan;

    Q_out = Q_on*fan_timeon + Q_off*(1-fan_timeon)*(1-bypass_pcnt);  %
heat rejected from radiator
    T6 = T5 - Q_out/(m_cool*cp_h2o);    % energy balance, temp of
coolant entering fuel cell
    T7 = T6 + Q_stack/(m_cool*cp_h2o);  % energy balance, temp of
coolant exiting fuel cell and entering humidifier

%-----------------------------------------------------------------------
% Expander Section
%-----------------------------------------------------------------------

% stagnation temps and pressures are not used due to negligible
difference to static values

    Pwr_e = ((mair4/Ma + mvap4/Mh2o)*8314*T4*k/(1-k)*((P1/P4)^((k-
1)/k)-1))*ne;
    if Pwr_e <0
        Pwr_e = 0;
    end

    if expnd
        % water recovery with expander
        Tout = T4 - T4*ne*(1-(P1/P4)^((k-1)/k));
```

```
        Pg_out = 10^(2.95E-2*(Tout-273)-9.18E-5*(Tout-273)^2+1.44E-
7*(Tout-273)^3-2.18);  % sat. pressure at fuel cell outlet (T in K and
Pg in atm)
        RH_out = P1*mvap4/((0.622*mair4 + mvap4)*Pg_out);  % Moran and
Shapiro -- p 582, eq. 12.43
        if RH_out>1
            mvap_out = 0.622*mair4*Pg_out/(P1 - Pg_out);  % assumes
Tout remains the same
            mh2o_rec = mvap4 - mvap_out;
        else
            mh2o_rec = 0;
        end
        if mh2o_rec < 0
            mh2o_rec = 0;
        end
    else
        % water recovery without expander
        mh2o_rec = 0;
    end

%-------------------------------------------------------------------------
% System Output and Plots
%-------------------------------------------------------------------------

    Pwr_netwexp = Pwr_e + Pwr_f - avePwr_fan - Pwr_c - Pwr_pump;  %
system power with an expander
    n_syswexp = Pwr_netwexp/(mH2*H2_LHV);                         %
system efficiency with an expander
    Pwr_net = Pwr_f - avePwr_fan - Pwr_c - Pwr_pump;              %
system power without an expander
    n_sys = Pwr_net/(mH2*H2_LHV);                                 %
system efficiency without an expander
    mh2o_bal = mh2o_net + mh2o_rec - mh2o_addh;                   %
system water balance

    current_plot(index) = current;          % Current range for plots
(A)
    stack_pwr(index) = Pwr_f;               % Fuel cell stack power (W)
    stack_eff(index) = n_stack;             % Fuel cell stack
efficiency
    stack_V(index) = Vstack;                % Fuel cell stack voltage
(V)
    stack_T(index) = T3-273;                % Fuel cell stack
temperature (°C)
    stack_Q(index) = Q_stack;               % Fuel cell stack heat
rejection (W)
    net_power(index) = Pwr_net;             % Net fuel cell system
power (W)
    net_powerwexp(index) = Pwr_netwexp;     % Net fuel cell system
power with expander (W)
    net_eff(index) = n_sys;                 % Net fuel cell system
adiabatic efficiency
    net_effwexp(index) = n_syswexp;         % Net fuel cell system
adiabatic efficiency with expander
    compressor_power(index) = Pwr_c;        % Compressor power (W)
    compressor_eff(index) = eff_overall;    % Compressor adiabatic
efficiency
```

```
    compressor_speed(index) = comp_speed;   % Compressor speed (RPM)
    compressor_Trise(index) = temp_rise;    % Compressor temperature
rise from ambient (°C)
    compressor_pressure(index) = P2;        % Compressor pressure (atm)
    expand_pwr(index) = Pwr_e;              % Expander power extracted
(W)
    pressure_ratio(index) = PR;             % Pressure ratio
    pressure_drop(index) = Pdrop;           % Pressure drop over stack
(atm)
    H2O_bal(index) = mh2o_bal;              % Water balance (kg/s)
    addedwater_air(index) = mh2o_addh;      % Water added to air (kg/s)
    addedwater_H2(index) = mvap_H2;         % Water added to hydrogen
(kg/s)
    RH4_log(index) = RH4;                   % Relative humidity of air
at fuel cell exit
    mtotal4(index) = (mvap4 + mair4)*1000;
    smair_in(index) = smair;
    fan_ontime(index)= fan_timeon;
    parasitics(index) = Pwr_c + Pwr_pump + avePwr_fan;
end % current

if expnd         % Tab delimited files
    Z = zeros(length(current_plot),1);
    M= [current_plot' stack_pwr' stack_eff' stack_V' stack_T' stack_Q'
net_powerwexp' net_effwexp' compressor_power' compressor_eff'
compressor_speed' compressor_Trise' compressor_pressure' expand_pwr'
pressure_drop' H2O_bal' addedwater_air' addedwater_H2' RH4_log'
parasitics'];
    dlmwrite('turboout.dat', M,'\t');
else
    Z = zeros(length(current_plot),1);
    M= [current_plot' stack_pwr' stack_eff' stack_V' stack_T' stack_Q'
net_power' net_eff' compressor_power' compressor_eff' compressor_speed'
compressor_Trise' compressor_pressure' pressure_drop' H2O_bal'
addedwater_air' addedwater_H2' RH4_log' parasitics'];
    dlmwrite('turboout.dat', M,'\t');
end

% plotted results
if ploton
figure(1)
plot(current_plot,stack_pwr/1000)
title('Figure 1: Stack Power')
xlabel('Current (A)')
ylabel('Power (kW)')
grid

figure(2)
plot(current_plot,stack_eff)
title('Figure 2: Stack Efficiency')
xlabel('Current (A)')
ylabel('Efficiency')
grid

figure(3)
plot(current_plot,stack_Q/1000)
title('Figure 3: Stack Heat Rejection')
```

```
xlabel('Current (A)')
ylabel('Heat Rejected (kW)')
grid

figure(4)
plot(current_plot,stack_V)
title('Figure 4: Stack Voltage')
xlabel('Current (A)')
ylabel('Voltage (V)')
grid

figure(5)
plot(current_plot,net_powerwexp/1000,current_plot,net_power/1000)
title('Figure 5: Net System Power')
xlabel('Current (A)')
ylabel('Power (kW)')
legend('with expander', 'no expander',0)
grid

figure(6)
plot(current_plot,H2O_bal)
title('Figure 6: Net Water Production')
xlabel('Current (A)')
ylabel('Water (kg/s)')
grid

figure(7)
plot(current_plot,net_effwexp,current_plot,net_eff)
title('Figure 7: System Efficiency')
xlabel('Current (A)')
ylabel('Efficiency')
legend('with expander', 'no expander',0)
grid

figure(8)
plot(current_plot,fan_ontime*100)
title('Figure 8: Percentage of Time that the Fan is On')
xlabel('Current (A)')
ylabel('Percent')
grid

figure(9)
plot(current_plot, expand_pwr/1000)
title('Figure 9: Expander Power')
xlabel('Current (A)')
ylabel('Power (kW)')
grid

figure(10)
plot(current_plot,compressor_power)
title('Figure 10: Power Required from Compressor')
xlabel('Current (A)')
ylabel('Power (W)')
grid

figure(11)
plot(current_plot, compressor_speed)
```

```
title('Figure 11:  Compressor Speed');
xlabel('Current (A)')
ylabel('speed (rpm)')
grid

figure(12)
plot(current_plot, compressor_Trise)
title('Figure 12:  Temperature Rise in Compressor')
xlabel('Current (A)')
ylabel('Degrees (°C)')
grid

figure(13)
plot(current_plot,compressor_pressure)
title('Figure 13: Output Pressure of Compressor')
xlabel('Current (A)')
ylabel('Pressure (atm)')
grid

figure(14)
plot(current_plot, compressor_eff)
title('Figure 14:  Adiabatic Efficiency of Compressor');
xlabel('Current (A)')
ylabel('Efficiency')
grid

figure(15)
plot(current_plot,pressure_drop)
title('Figure 15: Pressure Drop Across System')
xlabel('Current (A)')
ylabel('Pressure (atm)')
grid

figure(16)
plot(current_plot,parasitics/1000,current_plot,stack_pwr/1000)
title('Figure 16: Parasitic Power Compared to Fuel Cell Power')
xlabel('Current (A)')
ylabel('Power (kW)')
legend('Parasitic','Fuel Cell',0)
grid

figure(17)
plot(net_powerwexp/1000, stack_T)
title('Figure 17:  Stack Temperature')
xlabel('Total System Power (kW)')
ylabel('Temperature (°C)')
grid

figure(18)
plot(current_plot, smair_in/.8445, current_plot, mtotal4)
title('Figure 18:  Air Mass Flow');
xlabel('Current (A)')
ylabel('mass flow (g/s)')
legend('inlet air','outlet air',0)
grid

figure(19)
```

```
plot(current_plot, RH4_log)
title('Figure 19:  Fuel Cell Exit Humidity');
xlabel('Current (A)')
ylabel('Relative Humidity')
grid

figure(20)
plot(current_plot,addedwater_air*1000, current_plot,
addedwater_H2*1000)
title('Figure 20:  Water Added to Air and Hydrogen Streams');
xlabel('Current (A)')
ylabel('water (g/s)')
legend('Air Stream','H2 Stream',0)
grid
end
```

Subroutines

Here are the subroutines called by the primary programs:

LINEAR

```
function [A,B] = linear(X1, Y1, X2, Y2)

%function for finding a point inside the data range of the system that
is very close to an edge point and is linear
%with a point outside of the data range.  (X2,Y2) must be the edge
point of the data, and (X1,Y1) must be the data
%point outside of range

if X1==X2;
    X3 = X2;
    if Y1>Y2
        Y3 = Y2 - 0.0001;
    else
        Y3 = Y2 + 0.0001;
    end
elseif Y1==Y2
    Y3 = Y2;
    if X1>X2
        X3 = X2 - 0.0001;
    else
        X3 = X2 + 0.0001;
    end
else
    m = (Y1-Y2)/(X1-X2);
    %using y = mx + b
    b = Y1-m*X1;
    if m>1                      %slope greater than one means Y changes
faster than X
        if Y2>Y1                %if Y2 is greater than Y1, a higher
value of Y will put data in range
            Y3 = Y2 + 0.0001;
            X3 = (Y3 - b)/m;
```

```
        else
            Y3 = Y2 - 0.0001;
            X3 = (Y3 - b)/m;
        end
    else
        if X2>X1                %if X2 is greater than X1, a higher
value of X will put data in range
            X3 = X2 + 0.0001;
            Y3 = b + m*X3;
        else
            X3 = X2 - 0.0001;
            Y3 = b + m*X3;
        end
    end
end
A = X3;
B = Y3;
```

LINEAR2

```
function output = linear2(X,Y,Z,XI,YI,alt_index)        %X = pressure,
Y = smair, Z = data

%this function finds the value of the particular data of interest by
linear interpolation when it is outside of
%the data range

for index = 1:length(X(:,1))
    if X(index,1)==X(index,2)
        output(index) = Z(index,2) - (Z(index,2)-
Z(index,1))/(Y(index,2)-Y(index,1))*(Y(index,2)-YI);
    else
        output(index) = Z(index,2) - (Z(index,2)-
Z(index,1))/(X(index,2)-X(index,1))*(X(index,2)-XI(alt_index(index)));
    end
end
```

LINEAR22

```
function output = linear22(X,Y,Z,XI,YI)        %X = pressure, Y =
smair, Z = data

%this function finds the value of the particular data of interest by
linear interpolation when it is outside of
%the data range

if X(1)==X(2)
    output = Z(2) - (Z(2)-Z(1))/(Y(2)-Y(1))*(Y(2)-YI);
else
    output = Z(2) - (Z(2)-Z(1))/(X(2)-X(1))*(X(2)-XI);
end
```

LINEAR3

```
function output = linear3(X,Z,XI,alt_index)         %X = pressure, Z =
temp_rise

%this function finds the value of temp_rise at those points outside of
the data range

for index = 1:length(X(:,1))
    if X(index,1)==X(index,2)
        output(index) = Z(index,1);
    else
        output(index) = Z(index,2) - (Z(index,2)-
Z(index,1))/(X(index,2)-X(index,1))*(X(index,2)-XI(alt_index(index)));
    end
end
```

LINEAR32

```
function output = linear32(X,Z,XI)         %X = pressure, Z = temp_rise

%this function finds the value of temp_rise at those points outside of
the data range

    if X(1)==X(2)
        output = Z(1);
    else
        output = Z(2) - (Z(2)-Z(1))/(X(2)-X(1))*(X(2)-XI);
    end
```

LINEAR4

```
function output = linear4(X,Y,Z,XI,YI,alt_index)        %X = pressure,
Y = smair, Z = data

%this function finds the value of the particular data of interest by
linear interpolation when it is outside of
%the data range

for index = 1:length(X(:,1))
    if X(index,1)==X(index,2)
        output(index) = Z(index,2) - (Z(index,2)-
Z(index,1))/(Y(index,2)-Y(index,1))*(Y(index,2)-YI(alt_index(index)));
    else
        output(index) = Z(index,2) - (Z(index,2)-
Z(index,1))/(X(index,2)-X(index,1))*(X(index,2)-XI(alt_index(index)));
    end
end
```

Vita

My name is Galen W. Kulp. My educational experience includes studies at Eastern Mennonite University, Pennsylvania State University, and Virginia Polytechnic Institute and State University. I joined the 3-2 program at Eastern Mennonite, which is a program of study where 3 years are spent at Eastern Mennonite and 2 years are spent at Penn State. At the end of the program, I earned a Bachelor of Science in Liberal Arts from Eastern Mennonite, and a Bachelor of Science in Mechanical Engineering at Penn State.

This paper is my primary work at Virginia Tech. I've published one paper with the Future Technology of Transportation conference, and another one of my papers is soon to be published with the Future Car Congress. My degree at Virginia Tech is a Masters in Mechanical Engineering.